Resourceful Dependent Types

Andreas Abel

Department of Computer Science and Engineering
Gothenburg University

Quantitative typing integrates resource usage tracking into the type system. The prime example is linear lambda-calculus that requires that each variable is used exactly once. Following Girard's initial proposal of linear logic, an abundance of substructural type systems have been proposed that allow fine control over resources. These type systems find their applications in many areas:

1. Compilation: Cardinality analyses such as strictness, absence (dead code), linearity, or affine usage [Verstoep and Hage, 2015] enable diverse optimizations in the generated code.
2. Security is an instance of absence analysis: Classified information may not flow to unclassified output.
3. Differential privacy [Reed and Pierce, 2010]: Fuzziness of values is captured by a metric type system, and sufficient fuzz guarantees privacy in information aggregation.

These type systems annotate variables in the typing context with usage information, i. e., the context is a finite maps from variables x to their type A together with a resource value q. In the judgment $\Gamma \vdash t : C$, term t of type C can refer to variables x in Γ according to their usage description q.

The question how to extend quantitative typing to dependent types had been lacking a satisfactory answer for a long time. The problem can be traced back to the invariant of type theory that if $\Gamma \vdash t : C$, then $\Gamma \vdash C : \mathsf{Type}$, i. e., a well-typed term requires a well-formed type, *in the same context*. A judgement like $x :_1 A \vdash \mathsf{refl}\, x : (x \equiv_A x)$ violates this property, since x is used twice on the type side.

McBride [2016] cut the Gordian knot by decreeing that types are mental objects that do not consume resources. In particular, any reference to a variable in a type should count as 0. The invariant now only stipulates $0\Gamma \vdash_0 C : \mathsf{Type}$, which erases usage information by multiplication with 0, and can be seen as the vanilla, resource free typing judgement. McBride's approach has been refined by Atkey [2017] into two judgements $\vdash_1$ (resourceful) $\vdash_0$ (resourceless) where the latter is used for type constructors.

The radical solution however also has some drawbacks. By discounting resources in types in general,

- we cannot utilize usage information for optimizing the computation of types during typechecking; and
- we cannot interpret types as values, in particular, we lose the option to case on types. However, *type case* is useful at least in typed intermediate languages to derive specialized implementations of data structures and their operations for particular types.

We propose a fresh look at the problem of dependent resource typing, but decoupling resource information from the typing context Γ. Instead, we track usage information in terms and types separately by *resource contexts* γ and δ which map the variables x of Γ to a resource quantities q. The invariant becomes $\Gamma \vdash t^{\gamma} : A^{\delta}$ implies $\Gamma \vdash A^{\delta} : \mathsf{Type}^{\emptyset}$, and no extra trickery is needed. Dependent function types $\Pi^{q,r}\, A\, F$ are indexed by two quantities: q describes how

the function uses its argument of type A to produce the result; r describes how the codomain F uses that argument to produce the result type.

Resources q, r are drawn from a partially ordered commutative semiring which is positive ($q+r=0$ implies $q=r=0$) and free of zero dividers ($qr=0$ implies $q=0$ or $r=0$). Ordering $q \leq r$ expresses that q is more precise as r, in the same sense as subtyping $A \leq B$ states that A is more specific than B. An resource semiring to track linearity would be $\{\omega, 0, 1\}$ with $\omega \leq q$. It is a subsemiring of $\mathcal{P}(\mathbb{N})$ [Abel, 2015] with $\omega = \mathbb{N}$, $0 = \{0\}$, $1 = \{1\}$, pointwise addition and multiplication, and $\leq \,=\, \supseteq$.

In the following, we list the inference rules for judgment $\Gamma \vdash t^{\gamma} : A^{\delta}$ of a dependently typed lambda-calculus with a predicative universe hierarchy U_ℓ.

$$\text{UNIV}\ \frac{\vdash \Gamma}{\Gamma \vdash \mathsf{U}_\ell^{\emptyset} : \mathsf{U}_{\ell'}^{\emptyset}}\ \ell{<}\ell' \qquad \text{PI}\ \frac{\Gamma \vdash A^{\delta_1} : \mathsf{U}_\ell^{\emptyset} \qquad \Gamma \vdash F^{\delta_2} : A \xrightarrow{r} \mathsf{U}_\ell^{\delta_1}}{\Gamma \vdash (\Pi^{q,r}\, A\, F)^{\delta_1+\delta_2} : \mathsf{U}_\ell^{\emptyset}}$$

$$\text{VAR}\ \frac{\vdash \Gamma}{\Gamma \vdash x^{x:1} : A^{\delta}}\ \Gamma(x) = A^{\delta} \qquad \text{ABS}\ \frac{\Gamma,\, x{:}A^{\delta_1} \vdash t^{\gamma, x:q} : (F \cdot^{r} x)^{\delta_2, x:r}}{\Gamma \vdash (\lambda^{q} x.\, t)^{\gamma} : (\Pi^{q,r}\, A\, F)^{\delta_1+\delta_2}}$$

$$\text{APP}\ \frac{\Gamma \vdash t^{\gamma_1} : (\Pi^{q,r}\, A\, F)^{\delta_1+\delta_2} \qquad \Gamma \vdash u^{\gamma_2} : A^{\delta_1}}{\Gamma \vdash (t \cdot^{q} u)^{\gamma_1 + q\gamma_2} : (F \cdot^{r} u)^{\delta_2 + r\gamma_2}} \qquad \text{SUB}\ \frac{\Gamma \vdash t^{\gamma} : A^{\delta} \qquad \Gamma \vdash A^{\delta} \leq B^{\delta}}{\Gamma \vdash t^{\gamma} : B^{\delta}}$$

Subtyping $\Gamma \vdash A^{\delta} \leq A'^{\delta}$ takes imprecision in the resource information into account and is contravariant for function domains, as usual.

$$\frac{\Gamma \vdash A^{\delta} = A'^{\delta} : \mathsf{U}_\ell^{\emptyset}}{\Gamma \vdash A^{\delta} \leq A'^{\delta}} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \mathsf{U}_\ell^{\emptyset} \leq \mathsf{U}_{\ell'}^{\emptyset}}\ \ell \leq \ell'$$

$$\frac{\Gamma \vdash A'^{\delta_1} \leq A^{\delta_1} \qquad \Gamma,\, x{:}A'^{\delta_1} \vdash (F \cdot^{r} x)^{\delta_2, x:r} \leq (F' \cdot^{r} x)^{\delta_2, x:r}}{\Gamma \vdash (\Pi^{q,r}\, A\, F)^{\delta_1+\delta_2} \leq (\Pi^{q',r}\, A'\, F')^{\delta_1+\delta_2}}\ q' \leq q$$

Acknowledgments. This work was supported by Vetenskapsrådet under Grant No. 621-2014-4864 *Termination Certificates for Dependently-Typed Programs and Proofs via Refinement Types* and by the EU Cost Action CA15123 *Types for programming and verification.*

References

A. Abel. The next 700 modal type assignment systems. In T. Uustalu, editor, *21st Int. Conf. on Types for Proofs and Programs, TYPES 2015, Abstracts*. Institute of Cybernetics at Tallinn University of Technology, 2015. URL http://cs.ioc.ee/types15/abstracts-book/.

R. Atkey. The syntax and semantics of quantitative type theory. Under submission, 2017. URL https://bentnib.org/quantitative-type-theory.html.

C. McBride. I got plenty o' nuttin'. In S. Lindley, C. McBride, P. W. Trinder, and D. Sannella, editors, *A List of Successes That Can Change the World – Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*, volume 9600 of *Lect. Notes in Comput. Sci.*, pages 207–233. Springer, 2016. URL http://dx.doi.org/10.1007/978-3-319-30936-1_12.

J. Reed and B. C. Pierce. Distance makes the types grow stronger: a calculus for differential privacy. In P. Hudak and S. Weirich, editors, *Proc. of the 15th ACM SIGPLAN Int. Conf.*

on Functional Programming, ICFP 2010, pages 157–168. ACM Press, 2010. URL http://doi.acm.org/10.1145/1863543.1863568.

H. Verstoep and J. Hage. Polyvariant cardinality analysis for non-strict higher-order functional languages: Brief announcement. In K. Asai and K. Sagonas, editors, *Proc. of the 2015 Wksh. on Partial Evaluation and Program Manipulation, PEPM, 2015*, pages 139–142. ACM Press, 2015. URL http://doi.acm.org/10.1145/2678015.2682536.