A Quantitative Dependent Type Theory with Recursion

Oskar Eriksson, Nils Anders Danielsson, Andreas Abel

Chalmers University of Technology and University of Gothenburg, Sweden

In graded modal type theories the type system is parameterized by an algebraic structure, typically some form of semiring, containing *grades*. One use of grades, which we focus on, is to encode quantitative aspects like erasure, linear types, and affine types [3, 4, 5, 6].

In previous work, we have developed an Agda formalization of a graded modal type theory with dependent types [1] supporting Π-types, strong and weak Σ-types, an empty type, natural numbers, and a universe. Certain parts of the syntax, notably lambda abstractions and applications, are annotated with grades corresponding to how many times some part of the program, in this case the function argument, is used. This is checked primarily by a form of typing for grades, the *usage relation* $\gamma \blacktriangleright t$ between a grading context γ and an expression t. Here is a selection of rules (note that our formalization uses de Bruijn indices instead of variables):

$$\frac{}{\mathbf{0}.\,x{:}1.\,\mathbf{0} \blacktriangleright x} \qquad \frac{\gamma.\,x{:}p \blacktriangleright t}{\gamma \blacktriangleright \lambda^p x.t} \qquad \frac{\gamma \blacktriangleright t \qquad \delta \blacktriangleright u}{\gamma + p\delta \blacktriangleright t^p u} \qquad \frac{}{\mathbf{0} \blacktriangleright \mathsf{zero}} \qquad \frac{\gamma \blacktriangleright t}{\gamma \blacktriangleright \mathsf{suc}\, t} \qquad \frac{\gamma \blacktriangleright t}{\delta \blacktriangleright t}\ \delta \leq \gamma$$

In essence, the relation counts the uses of the free variables of t in terms of the operators of the semiring, lifted pointwise to contexts γ. Usage counting is not necessarily exact: the semiring is equipped with an order relation interpreted as usage subsumption or *compatibility*—a variable that is used p times can also be considered to be used q times if $q \leq p$. We proved key properties for usage such as a substitution lemma and subject reduction, and showed correctness of erasure, in the sense that parts marked as erasable can be safely removed during compilation.

Building on this work, we extend the formalization [2] with a correctness proof that goes beyond erasure: we prove that variables are used as many times as specified when a program is evaluated. Our approach is based on the work of Choudhury et al. [4] and involves a heap-based abstract machine with the capability to track how many times lookups may be performed. In contrast to their formulation, our machine uses an explicit stack of continuations. We also fix a problem with the usage relation in our previous work [1]: variable usage was sometimes handled incorrectly for natrec, the eliminator for natural numbers. As an example, our previous method allowed the program $\lambda^1 n.\texttt{plus}\ n\ n$ to be considered linear (here `plus` denotes addition for natural numbers, defined in a standard way).

In $\mathsf{natrec}^r_p\ (x_n.A)\ z\ (x_p.x_r.s)\ n$, the arguments z and s are the zero and suc branches, respectively, and n is the natural number argument. The s argument binds two variables, x_p, corresponding to the predecessor of the natural number, and x_r, corresponding to the recursive call. These are assigned the grades p and r, respectively ($p$redecessor and $r$ecursive call). There is also an explicit motive A dependent on the eliminated number (bound as x_n).

In order to find a correct usage count for natrec we make the following ansatz (which comes from following the structure of e.g. the eliminator for pairs):

$$\frac{\gamma_{\mathsf{z}} \blacktriangleright z \qquad \gamma_{\mathsf{s}}.\,x_p{:}p.\,x_r{:}r \blacktriangleright s \qquad \gamma_{\mathsf{n}} \blacktriangleright n}{q\gamma_{\mathsf{n}} + \delta \blacktriangleright \mathsf{natrec}^r_p\ (x_n.A)\ z\ (x_p.x_r.s)\ n}$$

Here q is some grade indicating the number of copies of n that are used and δ is a context representing the usage contributions from z and s. The main difficulty in finding proper values for q and δ comes from the recursion: whatever usage counts we choose to assign, they should work for all values of n, i.e. regardless of how deep the recursion is.

Let δ_i be the usage contribution from z and s for the number i. First, δ_0 corresponds to the zero case where only z is used, so we get $\delta_0 = \gamma_{\mathsf{z}}$. Similarly, δ_{i+1} corresponds to a successor case which uses s once (with γ_{s} resources) and the recursive call r times (up to subsumption). A single recursive call uses δ_i resources, so we get $\delta_{i+1} = \gamma_{\mathsf{s}} + r\delta_i$. Now we let δ be the approximation that we get by taking the infimum of the δ_i (for $i \in \mathbb{N}$): if such an infimum does not exist then $\mathsf{natrec}^r_p\,(x_n.A)\;z\;(x_p.x_r.s)\;n$ is not considered to be well-resourced.

Analogously, we obtain q as the infimum of the sequence q_i, where $q_0 = 1$ since in the base case n is "consumed fully" by the match, and $q_{i+1} = p + rq_i$ since n is used p times by s and the recursive call is again used r times.

With this usage rule, we can show the same substitution, subject reduction and erasure correctness properties as before [1] (for semirings with well-behaved greatest lower bounds). However, it also allows us to show a more precise form of resource correctness, stated using an abstract machine with a heap and a stack.

A machine state $\langle H;\, t;\, \rho;\, S\rangle$ consists of a heap H, head t, environment ρ, and stack S. Environments map variable names to pointers (again, note that our formalization uses de Bruijn indices instead of variables, we have not proved that the rules involving variables below actually work). Heap entries $x \mapsto^p (t, \rho)$ are associated with a grade p, indicating how many times lookup may be performed. They also contain an environment ρ that maps the variable names of t to the pointers of the heap. The evaluation stack S is a list of *continuations* that represent the context "around" the term that is currently being evaluated (the head). Like entries, each continuation contains an environment mapping variables to pointers. Reduction is performed in a call-by-name style; these are the rules for the lambda calculus part:

$$\begin{aligned}
\langle H.\, \rho(x) \mapsto^p (t, \rho').\, H';\, x;\, \rho;\, S\rangle &\to \langle H.\, \rho(x) \mapsto^q (t, \rho').\, H';\, t;\, \rho';\, S\rangle && \text{if } p - |S| = q\\
\langle H;\, t^p u;\, \rho;\, S\rangle &\to \langle H;\, t;\, \rho;\, \bullet^p u[\rho].\, S\rangle\\
\langle H;\, \lambda^p x.t;\, \rho;\, \bullet^p u[\rho'].\, S\rangle &\to \langle H.\, y \mapsto^{|S|p} (u, \rho');\, t;\, \rho[x \mapsto y];\, S\rangle && y \notin H
\end{aligned}$$

For variables, a lookup is performed and the grade of the corresponding entry is updated, indicating that some allowed lookups have been "used". We do not require the semiring to be equipped with subtraction (i.e. to be a ring) and instead define a form of "partial" subtraction via the relation $p - q = r$, defined to hold if r is the least grade such that $p \leq r + q$ (we require that if there is such an r then there is a least one). Because an eliminator is not necessarily linear in its scrutinee, a given lookup might "consume" multiple copies. For instance, for $\mathsf{natrec}^r_p\,(x_n.A)\,z\,(x_p.x_r.s)\,n$ the grade q is assigned to the argument n (where q is the infimum of 1, $r+p$, r^2+rp+p, and so on), so the machine is set up so that when $\mathsf{natrec}^r_p\,(x_n.A)\,z\,(x_p.x_r.s)\,x$ is evaluated (in an empty stack) the lookup of x consumes q copies. The stack could contain several eliminators, so the variable rule uses $|S|$, a grade associated to all the eliminators in S (note that this grade might not be defined if some infimum does not exist).

The non-variable cases are mainly concerned with the stack. For eliminators like applications and natrec, a corresponding continuation is put on the stack and evaluation continues with the scrutinee. If the result is a compatible value, then the continuation is removed from the stack and zero or more new entries are added to the heap. The number of allowed lookups $|S|p$ for a new heap entry is given by a corresponding annotation p on the syntax, scaled by the stack multiplicity $|S|$.

We show termination: evaluation of a well-resourced term of type $\mathbb{N}$, starting with an empty heap and stack, will reach a final state with a numeral in head position and an empty stack. This implies that evaluation does not use variables more times than specified, since doing so is prevented by the reduction semantics. We also show that variables are not used fewer times than specified: the final heap does not contain any entries for which lookups are "necessary". For instance, if the linearity semiring is used, then there is no heap entry with the grade 1.

Acknowledgements. We would like to thank the reviewers. Andreas Abel and Oskar Eriksson acknowledge support by *Vetenskapsrådet* (the Swedish Research Council) via project 2019-04216 *Modal typteori med beroende typer* (Modal Dependent Type Theory). Oskar Eriksson additionally acknowledges support by *Knut and Alice Wallenberg Foundation* via project 2019.0116. Nils Anders Danielsson acknowledges support from Vetenskapsrådet (2023-04538).

References

[1] Andreas Abel, Nils Anders Danielsson, and Oskar Eriksson. A graded modal dependent type theory with a universe and erasure, formalized. *Proc. ACM Program. Lang.*, 7(ICFP), August 2023. doi:10.1145/3607862.

[2] Andreas Abel, Nils Anders Danielsson, Oskar Eriksson, Gaëtan Gilbert, Ondřej Kubánek, Wojciech Nawrocki, Joakim Öhman, and Andrea Vezzosi. An Agda Formalization of a Graded Modal Type Theory with a Universe Hierarchy and Erasure, 2025. URL: https://github.com/graded-type-theory/graded-type-theory.

[3] Robert Atkey. Syntax and semantics of quantitative type theory. In *LICS '18: Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 56–65, 2018. doi:10.1145/3209108.3209189.

[4] Pritam Choudhury, Harley Eades III, Richard A. Eisenberg, and Stephanie Weirich. A graded dependent type system with a usage-aware semantics. *Proc. ACM Program. Lang.*, 5(POPL), January 2021. doi:10.1145/3434331.

[5] Conor McBride. I got plenty o' nuttin'. In *A List of Successes That Can Change the World*, volume 9600 of *LNCS*, pages 207–233, 2016. doi:10.1007/978-3-319-30936-1_12.

[6] Benjamin Moon, Harley Eades III, and Dominic Orchard. Graded modal dependent type theory. In *Programming Languages and Systems, 30th European Symposium on Programming, ESOP 2021*, volume 12648 of *LNCS*, pages 462–490, 2021. doi:10.1007/978-3-030-72019-3_17.