Towards Quantitative Inductive Families

Yulong Huang and Jeremy Yallop

University of Cambridge, UK

We present the on-going work to extend Quantitative Type Theory (QTT) [2] with inductive families [6] whose constructors are user-annotated. We give the general scheme for defining lists with quantities, which we believe can be extended to arbitrary inductive families, subsuming instances of datatypes like dependent pairs [1, 2, 4], unit [2, 4], Boolean [2], natural numbers [1], and lists [3] scattered in recent work.

Quantitative Type Theory. Quantitative type theory extends MLTT with runtime usage annotations on variables, ranging from 0 (unused), and 1 (used linearly), to ω (used unlimitedly). Our judgements are in the form of $\Gamma \vdash M \overset{\sigma}{:} A \ ; \ \underline{m}$ (inspired by [1]), which says that M is well-typed in Γ and $\underline{m} = q_1, \cdots, q_n$ is a *quantity assignment* to variables in the context.

$$\textsc{ty-Pi}\ \frac{\Gamma \vdash A \overset{0}{:} \mathrm{Type} \ ; \ \underline{0} \qquad \Gamma, x{:}A \vdash B \overset{0}{:} \mathrm{Type} \ ; \ \underline{0}}{\Gamma \vdash \Pi x \overset{q}{:} A.\, B \overset{0}{:} \mathrm{Type} \ ; \ \underline{0}} \qquad \textsc{ty-Var}\ \frac{x : A \in \Gamma}{\Gamma \vdash x \overset{\sigma}{:} A \ ; \ \sigma_x}$$

The typing rules are standard except for quantity information. The parameter σ in the judgement (above the colon) indicates the *mode* of type checking, either 0 (where variable usage is ignored) or 1 (where variable usage is counted). Terms checked in mode 0 are runtime-irrelevant and require no resource, and the rules ensure that they will not appear at runtime. Types are runtime irrelevant, so Π is judged with $\sigma = 0$, takes erased arguments, and is assigned a vector of zero quantities. Variables are assigned σ_x, denoting σ for x and 0 for other variables (note the notation abuse here to implicitly coerce modes to quantities).

$$\textsc{ty-Lam}\ \frac{\Gamma, x{:}A \vdash M \overset{\sigma}{:} B \ ; \ \underline{m}, q}{\Gamma \vdash \lambda x \overset{q}{:} A.\, M \overset{\sigma}{:} \Pi x \overset{q}{:} A.\, B \ ; \ \underline{m}} \qquad \textsc{ty-App}\ \frac{\sigma' = 0 \Leftrightarrow (\sigma = 0 \vee q = 0) \qquad \Gamma \vdash M \overset{\sigma}{:} \Pi x \overset{q}{:} A.\, B \ ; \ \underline{m} \qquad \Gamma \vdash N \overset{\sigma'}{:} A \ ; \ \underline{n}}{\Gamma \vdash M\ N \overset{\sigma}{:} B[M/x] \ ; \ \underline{m} + q\underline{n}}$$

The lambda case is straightforward. For an application $M\ N$, M uses resources $\underline{m}$ and uses its argument q times, while N uses resources $\underline{n}$, so the total resource of the application is $\underline{m} + q\underline{n}$ (operations on quantities, like addition and multiplication, extend pointwise to assignments). The side condition ensures that erased terms cannot appear at runtime: a function accepts an erased argument ($\sigma' = 0$) only if the entire application is runtime irrelevant ($\sigma = 0$), or if the function does not use its argument at all ($q = 0$).

QTT terms are subject to the usual $\beta\eta$-equality and conversion. The quantities have a partial order of $0 \leq \omega \geq 1$. QTT supports a sub-usaging rule for over-approximating the resource usage: if $\Gamma \vdash M \overset{\sigma}{:} A \ ; \ \underline{m}$ and $\underline{m} \leq \underline{m}'$ then we can also assign $\underline{m}'$ to M.

QTT with linear lists. We extend QTT with linear lists, an instance of the general scheme for lists. Here is the inductive family signature with each constructor argument marked with a quantity (both 1 here), specifying its runtime usage:

$$\texttt{data}\ \mathbf{List}^{11}\ (A{:}\mathrm{Type}) : \mathrm{Type}\ \texttt{where}\ [\,] : \mathbf{List}^{11} A \mid \_::\_ : \Pi x \overset{1}{:} A.\, \Pi xs \overset{1}{:} \mathbf{List}^{11} A.\, \mathbf{List}^{11} A$$

This extends our type theory with a type formation rule and one introduction rule for each constructor. Type former $\mathbf{List}^{11}$ is judged with $\sigma = 0$, since "types need nothing". The introduction rules are similar to rule TY-APP, where we sum the usage of all the arguments.

$$\textsc{ty-List}\ \frac{\Gamma \vdash A \overset{0}{:} \mathrm{Type}\ ;\ \underline{0}}{\Gamma \vdash \mathbf{List}^{11}\, A \overset{0}{:} \mathrm{Type}\ ;\ \underline{0}} \qquad \textsc{ty-Nil}\ \frac{\Gamma \vdash A \overset{0}{:} \mathrm{Type}\ ;\ \underline{0}}{\Gamma \vdash [\,] \overset{\sigma}{:} \mathbf{List}^{11}\, A\ ;\ \underline{0}} \qquad \textsc{ty-Cons}\ \frac{\Gamma \vdash M \overset{\sigma}{:} A\ ;\ \underline{m} \quad \Gamma \vdash N \overset{\sigma}{:} \mathbf{List}^{11}\, A\ ;\ \underline{n}}{\Gamma \vdash M :: N \overset{\sigma}{:} \mathbf{List}^{11}\, A\ ;\ \underline{m} + \underline{n}}$$

Given a predicate P, list L, and branches M and N for the empty and non-empty cases, we get a term of type $P[L/ls]$ from the eliminator. The typing and β-reduction (omitted here) are conventional. The two branches M and N use resources $\underline{m}$ and $\underline{n}$ respectively. The last premise says that N must use the head argument x and the inductive hypothesis r exactly once, while the tail argument xs is for typing only (hence unused).

$$\textsc{ty-ElimList}\ \frac{\begin{array}{c}\Gamma, ls{:}\mathbf{List}^{11}\, A \vdash P \overset{0}{:} \mathrm{Type}\ ;\ \underline{0} \quad \Gamma \vdash L \overset{\sigma}{:} \mathbf{List}^{11}\, A\ ;\ \underline{l} \quad \Gamma \vdash M \overset{\sigma}{:} P[[\,]/ls]\ ;\ \underline{m} \\ \Gamma, x{:}A, xs{:}\mathbf{List}^{11}\, A, r{:}P[xs/ls] \vdash N \overset{\sigma}{:} P[x :: xs/ls]\ ;\ \underline{n}, 1, 0, 1\end{array}}{\Gamma \vdash \mathit{Elim}_{list}(P, L, M, (x, xs, r).N) \overset{\sigma}{:} P[L/ls]\ ;\ \underline{l} + \underline{m} + \omega\underline{n}}$$

The eliminator reduces to M if L is empty, using resources $\underline{m}$. Otherwise, it evaluates N many times until it hits the base case M, using resources $\underline{m} + \omega\underline{n}$ (we do not know the exact number of recursions). We take the join of the quantities in these two cases (since we do not know which branch the eliminater reduces to) and add the resource used by L to obtain the quantity assignment for the eliminator, $\underline{l} + (\underline{m} \sqcup (\underline{m} + \omega\underline{n}))$, which simplifies to $\underline{l} + \underline{m} + \omega\underline{n}$.

Lists with quantities. For any two fixed quantities p and q, we can give a inductive family signature similar to that of $\mathbf{List}^{11}$, except the constructor arguments are marked with p and q instead of 1. Again, our type theory is extended with type formation and introduction rules. The type former for $\mathbf{List}^{pq}$ is judged with $\sigma = 0$ as before. Constructor applications are treated like function applications – we sum the resource assigned to each argument multiplied by its designated usage. As in TY-APP, the side conditions say that an argument is erased only if the entire expression is runtime irrelevant or it is never used.

$$\textsc{ty-List-pq}\ \frac{\Gamma \vdash A \overset{0}{:} \mathrm{Type}\ ;\ \underline{0}}{\Gamma \vdash \mathbf{List}^{pq}\, A \overset{0}{:} \mathrm{Type}\ ;\ \underline{0}} \qquad \textsc{ty-Cons-pq}\ \frac{\sigma_1 = 0 \Leftrightarrow (\sigma = 0 \vee p = 0) \quad \sigma_2 = 0 \Leftrightarrow (\sigma = 0 \vee q = 0) \quad \Gamma \vdash M \overset{\sigma_1}{:} A\ ;\ \underline{m} \quad \Gamma \vdash N \overset{\sigma_2}{:} \mathbf{List}^{pq}\, A\ ;\ \underline{n}}{\Gamma \vdash M :: N \overset{\sigma}{:} \mathbf{List}^{pq}\, A\ ;\ p\underline{m} + q\underline{n}}$$

The eliminator for $\mathbf{List}^{pq}$ has the same structure and typing rules as in rule TY-ELIMLIST with a more general premiss for N. The list's head should be used p times in N. The tail is used q_1 times directly and q_2 times in the recursion in N, so the combined usage $q_1 + q_2$ should be no more than q, the specified usage of the tail. For example, if $p = q = 1$, we have a more general eliminator of $\mathbf{List}^{11}$ that can only use either xs or r once.

$$\textsc{ty-ElimList-pq}\ \frac{\begin{array}{c}\Gamma, ls{:}\mathbf{List}^{pq}\, A \vdash P \overset{0}{:} \mathrm{Type}\ ;\ \underline{0} \quad \Gamma \vdash L \overset{\sigma}{:} \mathbf{List}^{pq}\, A\ ;\ \underline{l} \quad \Gamma \vdash M \overset{\sigma}{:} P[[\,]/ls]\ ;\ \underline{m} \\ q_1 + q_2 \le q \quad \Gamma, x{:}A, xs{:}\mathbf{List}^{pq}\, A, r{:}P[xs/ls] \vdash N \overset{\sigma}{:} P[x :: xs/ls]\ ;\ \underline{n}, p, q_1, q_2\end{array}}{\Gamma \vdash \mathit{Elim}_{list}(P, L, M, (x, xs, r).N) \overset{\sigma}{:} P[L/ls]\ ;\ \underline{l} + (\underline{m} \sqcup (q_2\underline{m} + (q_2 + 1)\underline{n}))}$$

Calculation of the quantity assignment is also similar to rule TY-ELIMLIST. We join the usage of the eliminator's two cases and add the resource for creating the list. The base case M has usage m. The usage of N depends on the number of times it uses the induction hypothesis r, i.e. the value of q_2: the base case is evaluated q_2 times and the inductive case $q_2 + 1$ times, so quantity assignment for rule ELIMLIST-PQ is $\underline{l} + (\underline{m} \sqcup (q_2\underline{m} + (q_2 + 1)\underline{n}))$.

Our extension is sound because erasing quantity-related information gives the usual inductive families [6], whose soundness is well known [5]. We have shown that the extension respects QTT's syntactic properties, e.g. substitution and subject reduction.

Lemma 1.1 (Substitution)**.** *The following rule for substitution is admissible:*

$$\textsc{ty-Subst}\quad \frac{\Gamma, x{:}A \vdash M \overset{\sigma}{:} B \;;\; \underline{m}, q \qquad \Gamma \vdash N \overset{\sigma'}{:} A \;;\; \underline{n} \qquad \sigma' = 0 \Leftrightarrow q = 0}{\Gamma \vdash M[N/x] \overset{\sigma}{:} B[N/x] \;;\; \underline{m} + q\underline{n}}$$

Lemma 1.2 (Reduction)**.** *If* $\Gamma \vdash M \overset{\sigma}{:} A \;;\; \underline{m}$ *and* M *reduces to* M'*, then* $\Gamma \vdash M' \overset{\sigma}{:} A \;;\; \underline{m}$ *is derivable.*

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):920–954, 2023.

[2] Robert Atkey. Syntax and semantics of quantitative type theory. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 56–65. ACM, 2018.

[3] Robert Atkey. Polynomial time and dependent types. *Proc. ACM Program. Lang.*, 8(POPL):2288–2317, 2024.

[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):1–32, 2021.

[5] Peter Dybjer. Inductive sets and families in martin-lof's type theory and their set-theoretic semantics. In *Logical frameworks*, pages 280–306. 1991.

[6] Peter Dybjer. Inductive families. *Formal aspects of computing*, 6:440–465, 1994.