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 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 Γ ⊢ M : A ; m (inspired by [1]), which says that M is well-typed in Γ and m = q₁, · · · , qₙ is a quantity assignment to variables in the context.

\[ \frac{\Gamma \vdash A : \text{Type} ; q}{\Gamma, x : A \vdash B : \text{Type} ; q'} \]

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 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 "types need nothing". The introduction rules are similar to rule ty-App, where we sum the usage of all the arguments.

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 Γ ⊢ M : A ; m (inspired by [1]), which says that M is well-typed in Γ and m = q₁, · · · , qₙ is a quantity assignment to variables in the context.

\[ \frac{\Gamma \vdash A : \text{Type} ; q}{\Gamma, x : A \vdash B : \text{Type} ; q'} \]

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.

The side condition ensures that erased terms cannot appear at runtime: a function accepts an erased argument (σ = 0 or 1) only if the entire application is runtime irrelevant (σ = 0), or if the function does not use its argument at all (q = 0).

QTT terms are subject to the usual βη-equality and conversion. The quantities have a partial order of 0 ≤ ω ≥ 1. QTT supports a sub-usaging rule for over-approximating the resource usage: if Γ ⊢ M : A ; m then we can also assign 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:

\[ \text{data } \text{List}^{11} (A : \text{Type}) : \text{Type } \text{where } [ ] : \text{List}^{11} A \mid \_ \_ : \Pi x : A. \Pi n : \text{List}^{11} A. \text{List}^{11} A \]

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

\[ \frac{\Gamma \vdash M : \text{Type} ; n}{\Gamma \vdash \text{List}^{11} M : \text{Type} ; n} \]
Towards Quantitative Inductive Families

Huang, Yallop

ty-List

Γ ⊢A
0
: Type ; 0
Γ ⊢List
11
A
0
: Type ; 0
ty-Nil
Γ ... [5]. We have shown that the extension respects
QTT’s syntactic properties, e.g. substitution and subject reduction.

2

Γ ⊢ (we do not know the exact
Γ ⊢ (respectively. The last premise
Γ ⊢ (we do not
Γ ⊢ (we do not
Γ ⊢ (we do not

ElimList-pq

usage of the eliminator’s two cases and add the resource for creating the list. The base case
xs
is evaluated
2
times directly and
2
should be no more than
2
the specified usage of the tail. For example, if
p = q = 1, we have a more
general eliminator of
List
11
that can only use either
xs
or
r
once.

Lists with quantities. For any two fixed quantities
p
and
q,
we can give a inductive family signature similar to that of
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
List
p,q
is judged with
σ = 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.

TY-List-PQ

Γ ⊢ A
0
: Type ; 0
Γ ⊢ List
p,q
A
0
: Type ; 0

The eliminator for
List
p,q
has the same structure and typing rules as in rule
TY-ElimList
with a more general premise for
N.
The list’s head should be used
p
times in
N.
The tail is used
q
and
q
times in the recursion in
N,
so the combined usage
q + q
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
List
11
that can only use either
xs
or
r
once.

TY-ElimList-PQ

Γ, ls: List
p,q
A
0
⊢ P
0
: Type ; 0
Γ ⊢ List
p,q
A
0
: Type ; 0

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:
the base case is evaluated
q
times and the inductive case
q
+ 1 times, so quantity assignment for rule
ElimList-PQ
is
l + (m \cup (q_2m + (q_2 + 1)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.

85
Lemma 1.1 (Substitution). The following rule for substitution is admissible:

\[\begin{align*}
\Gamma, x &: A \vdash M^\sigma : B; \ m, q & \quad \Gamma \vdash N^\sigma' : A; \ n \\
\sigma' = 0 & \iff q = 0
\end{align*}\]

\[\Gamma \vdash M[N/x]^\sigma : B[N/x]; \ m + q n\]

Lemma 1.2 (Reduction). If \(\Gamma \vdash M^\sigma : A; \ m\) and \(M\) reduces to \(M'\), then \(\Gamma \vdash M'^\sigma : A; \ m\) is derivable.

References