Graded Modalities as Linear Types

ASHER FROST

We present an approach that models graded modal bindings using only linear ones. Inspired by Granule, we present an approach in Idris 2 to model limited grades using a construction, called "mu" (M). We present the construction of M, related ones, operations on them, and some useful properties

Todo list

Update everything

1 Introduction

Idris shows the practical applications of QTT, a branch of type theory based around the notion of *usage*[2]. In particular, the ability for multiplicities to model both modality in logic and resource usage in programs adds yet another way to reason about programs, and is utilized by Linear Haskell (as an extension to the language) and as part of Idris 2 (as a core language) [2, 1]

Idris 2's use of QTT as a core language has been one of its strong selling points. There are quite a few things that can be easily done with restricted bindings that cannot be done any other way. For instance, they provide a good model for type variables, proof irrelevance at runtime, a less magical system of IO, among other properties.

Recently, however, there has been interest in GrTT, which allows for more dynamic restrictions on the usage of types, particularly allowing for usages greater than 1 [6]. Rather than arguing for the use of GrTT in Idris (which would add complexity), we instead propose a construction, M, that can model graded quantities using only linear types.

Furthermore, a system is described that *only* relies on types with multiplicities 0 and 1 to build a system (almost) as powerful as such as system with ω. We do this by using the exponential construction, here called M, to model arbitrary multiplicities.

2 M types

The core construction here is M, or, in Idris, `Mu`, type. This is made to model a "source" of a given value. It is indexed by a natural number, a type, and an erased value of that type. The definitions of this in Idris are given at 1

Definition 2.1. M is a polymorphic type with signature $M : (n : \mathbb{N}) \to_0 (t : *) \to_0 (w : t) \to_0 *$

In addition, M has two constructors

Author's Contact Information: Asher Frost.

```
data Mu : (n : Nat) -> (t : Type) -> (w : t) -> Type where
        MZ :
                Mu Z t w
        MS :
                (1 w : t) ->
                (1 xs : (Mu n t w)) ->
                Mu (S n) t w
```

Listing 1. The definition of M in Idris

Definition 2.2. There are two constructors of M, $\diamond$ `MZ`, and $\odot$, `MS`, Which have the singatures $\diamond : M\ n\ t\ w$ and $\odot : (w : t) \to_1 (\_ : M\ n\ t\ w) \to_1 M\ (Sn)\ t\ w$

Firstly, it should be noted taht the names were chose due to tthe fact that the indices of them are related. That is, $\diamond$ or "mu zero" will always be indexed by 0, and $\odot$, "mu successor", is always indexed by the succesor of whatever is given in.

Remark 2.3. $M\ 0\ t\ w$ can only be constructed by $\diamond$.

Intuitivly, M represents n copies of t, all with the value w, very much inspired by the paper "How to Take the Inverse of a Type". For instance, if we want to construct $x : M\ 2$ `String "value"`, we can only construct this thorugh $\odot$, an we know, bu nothing but the first arguement of the value, that we must take as a initial arguement `value`, and as a second value of the form $M\ 1$ *String*`"value"`, which we then repeat one more time and get another `"value"` an finally we match $M\ 0$ `String "value"` and get that we must have $\diamond$. We can then say that we kneow that the only constructors of $M\ 2$ `String "String"` is `"value"`$\odot$`"value"`$\odot\diamond$. Note the similarity between the natural number and the constructors. Just as we get 2 by applying Stwice to Z, we get $M\ 2$ `String "value"` by applying $\odot$ twice to $\diamond$. This relationship betweeen M types and numbers is far more extensive, as we will cover later

The windex or "witness" is the value being copied. Notably, if we remove the windex, we would have $\diamond : Mnt$, and $\odot : t \to_1 Mnt \to_1 M(Sn)t$, which is simply `LVect`. The reason that this is undesirable is that we don't want this as it allows for M to have hetrogeneous elements. However, if we are talking about "copies" of something, we know that should all be the exact same.

There are two very basic functions that bear mention with respect to M. The first of these is `witness`, which is of the form $\texttt{witness} :_0 \{w : t\} \to_0 M\ n\ t\ w \to t$. Note that this is an *erased* function. Its implementation is quite simple, just being $\texttt{witness}\{w\}\_ := w$, which we can create, as erased functions can return erased values

Need citation

. In addition, we also have $\texttt{drop} : (\_ : M\ 0\ t\ w) \to_1 ()$, which allows us to drop a value. Its signature is simple $\texttt{drop}\diamond := ()$.

Finally, we have once, which takes a value of form $M1$ and extracts it into the value itself. It has a signature $\texttt{once} : (\_ : M\ 1\ t\ w) \to_1 t$, where we have $\texttt{once}(x\odot\diamond) := x$

2.1 Uniqueness

While wserves a greatj purpose in the interpretation of M it perhasp serves an even greater purpose in terms of values of M. We can, using w,provate that there exists exactly one inhabitatent of the type $M\ n\ t\ w$, so long as that type is well formed.

Segway into proof

Lemma 2.4. *$x : M\ n\ t\ w$ only if the concrete value of x contains n applicationso f $\odot$.*

Proof. Induct on n,the first case, $M\ 0\ t\ w$, contains zero applications of $\odot$, as it is $\diamond$. The second one splits has $x : M(Sn')tw$, and we can destruct on the only possible constructor of M for S, $\odot$, and get $y :_1 t$ and $z :_1 Mntw$ where $x =_? (y \odot z)$. We know by the induction hypothesis that z contains exactly n' uses of $\odot$, and we therefore know that the constructor occurs one more times then that, or Sn' □

This codifies the relationship between M types and natural numbers. Next we prove that we can establish equality between any two elements of a given M instance. This is equivalent to the statement "there exists at most one M"

Need citation

.

Lemma 2.5. *If both x and y are of type $M\ n\ t\ w$, then $x =_? y$.*

Proof. Induct on n.

- The first case, where $x$and $y$are both $M\ 0\ t\ w$, is trivial because, as per 2.3 they both must be $\diamond$
- The inductive case, where, from the fact that for any a and b (both of $M\ n\ '\ tw$) we have $a =_? b$, we prove that we have, for $x$and $y$of $M\ (Sn')\ t\ w$ that $x =_? y$. We note that we can destruct both of these, with $x_1 : M\ n\ '\ tw$ and $y_1 : M\ n\ '\ tw$, into $x = x_0 \odot x_1$ and $y = y_0 \odot y_1$, where we note that both x_0 and y_0 must be equal to w,and then we just have the induction hypthesis, $x_1 =_? y_1$

We thereby can simply this to the fact that $M\ n\ '\ tw$ has the above property, which is the induction hypothesis. □

Finish proof

However, we can make a even more specific statement, given the fact we know that w is an inhabitent of t.

Theorem 2.6 (Uniqueness). *If $M\ n\ t\ w$ is well formed, then it must have* exactly *one inhabitent.*

Proof. We know by 2.5 that there is *at most* one inhabitent of $M\ n\ t\ w$. We then induct on n to show that there must also exist *at least* one inhabitent of the type.

- The first case is that $M\ 0\ t\ w$ is always constructible, which is trivial, as this is just $\diamond$.
- The second case, that $M\ n\ '\ tw$ provides $M\ (Sn')\ t\ w$ being constructible is also trivial, namely, if we have the construction on $M\ n\ '\ tw$ as x, we *know* that the provided value must be w.

Given that we can prove that there must be at least and at most one inhabitent, we can prove that there is exactly one inhabitent. □

This is very important for proofs on M. It corresponds to the fact that there is only one way to copy something, to provide another value that is the exact same as the first.

2.2 Graded Modalities With M

A claim was made earlier that M types can be used to model graded modalities within QTT. The way it does this, however, is not by directly equating M types with [] types [6]. It instead does this in a similar way to how others have embedded QTT in Agda [4, 3].

Namely, rather than viewing $M\ n\ t\ w$ as the *type* $[t]_n$, we instead view it as the *judgment* $[w] : [t]_r$. Fortunately, due to the fact that this is Idris 2, we don't need a separate $\Vdash$, as M is a type, which, like any other, can be bound linearly, so, the equivalent to the GrTT statement $\Gamma \vdash [x] : [a]_r$ is $\Gamma \vdash \phi : M\ r\ a\ x$, which can be manipulated like any other type.

$$\diamond \otimes x := x \tag{1}$$
$$Z + x := x \tag{2}$$
$$(a \odot b) \otimes x := a \odot (b \otimes x) \tag{3}$$
$$(Sn) + x := S(n + x) \tag{4}$$

This is incredibly powerful. Not only can we reason about graded modalities in Idris, we can reason about them in the language itself, rather than as part of the syntax, which allows us to employ regular proofs on them. This is very apparent in the way constructions are devised. For instance, while Granule requires separate rules for dereliction, we do not, and per as a matter of fact we just define it as `once`; a similar relationship exists between weakening and `drop`.

Remark 2.7. We assume that $Mntw$ is equivalent to $[w] : [t]_n$.

Unfortunately, there is no way to prove this in either language, as M can't be constructed in Granule, and graded modal types don't exist in Idris 2.

2.3 Operations on M

There are number of operations that are very important on M. The first of these that we will discuss is combination. We define this as $\otimes : (\_ : M\ m\ t\ w) \to_1 (\_ : M\ n\ t\ w) \to_1 M\ (m + n)\ t\ w$

Need code

, and we define it inductivly as $\odot \otimes x := x$, and $(a \odot b) \otimes x := a \odot (b \otimes x)$. Note the similarity between the natural number indicies and the values, where $0 + x := x$ and $(Sn) + x := S(n + x)$

Fix the alignment of these

In addition, using the assumption of 2.7, we can liken the function $\otimes$ to context concatenation [6]. However, unlike Granule, we define this in the language itself. Per as a matter of fact, it isn't actually possible to construct $\otimes$in Granule, as it would require a way to reason about type level equality, which isn't possible.

Lemma 2.8. *$\otimes$ is commutative[1], that is, $x \otimes y =_? y \otimes x$.*

Proof. These types are the same, and by 2.6, they are equal. □

Given that we can "add" (Or, as we will see later, multiply) two M, it would seem natural that we could also subtract them. We can this function, which is the inverse of `combine`, `split`, which has the signature $\texttt{split} : (n : \mathbb{N}) \to_1 (\_ : M\ (m + n)\ t\ w) \to_1 M\ m\ t\ w \times^1 M\ n\ t\ w$. We actually use this a fair bit more than we use `combine`, as `split`doesn't impose any restrictions on the witness, which is very useful for when we discuss ^types.

In a similar manner to how we proved (in 2.8) the commutativity of $\otimes$, we can prove that these are inverses.

Lemma 2.9. *Given that we have $f := \texttt{split}$, and $g := \texttt{uncurry}^1(\otimes)^2$, then f and g are inverses.*

Proof. The type of f is $(\_ : M\ (m + n)\ t\ w) \to_1 M\ m\ t\ w \times^1 M\ n\ t\ w$, and that of g is $(\_ : M\ m\ t\ w \times^1 M\ n\ t\ w$ $n)\ t\ w$, so there composition $f \circ g : (\_ : M\ (m + n)\ t\ w) \to_1 M\ (m + n)\ t\ w$ (the same for $g \circ f$). Any function from a unique object and that same unique object is an identity, thereby these are inverses

Need citation

□

[1] You can also prove associativity and related properties, the proof is the same

[2] Given that we have $uncurry^1 : ((\_ : a) \to_1 (\_ : b) \to_1 c) \to (\_ : (a \times^1 b)) \to_1 c$

One of the more notable constructions

3 ω and Ω Types

4 Exponential and Existential Types

5 Using M and Friends

6 Conclusion

Related Work

Acknowledgements

Thank you to the Idris team for helping provide guidance and review for this. In particular, I would like to thank Constantine for his help in the creation of M types

Artifacts

All Idris code mentioned here is either directly from or derived from the code in the Idris library `idris-mult`, which may be found at .

Add link

References

[1] Jean-Philippe Bernardy et al. "Linear Haskell: practical linearity in a higher-order polymorphic language". In: *Proc. ACM Program. Lang.* 2.POPL (Dec. 2017). DOI: 10.1145/3158093. URL: https://doi.org/10.1145/3158093.

[2] Edwin Brady. "Idris 2: Quantitative Type Theory in Practice". In: 2021, pp. 32460–33960. DOI: 10.4230/LIPICS.ECOOP.2021.9.

[3] Maximilian Doré. *Dependent Multiplicities in Dependent Linear Type Theory*. 2025. arXiv: 2507.08759 [cs.PL]. URL: https://arxiv.org/abs/2507.08759.

[4] Maximilian Doré. "Linear Types with Dynamic Multiplicities in Dependent Type Theory (Functional Pearl)". In: *Proc. ACM Program. Lang.* 9.ICFP (Aug. 2025). DOI: 10.1145/3747531. URL: https://doi.org/10.1145/3747531.

[5] Danielle Marshall and Dominic Orchard. "How to Take the Inverse of a Type". In: *36th European Conference on Object-Oriented Programming (ECOOP 2022)*. Ed. by Karim Ali and Jan Vitek. Vol. 222. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 5:1–5:27. ISBN: 978-3-95977-225-9. DOI: 10.4230/LIPIcs.ECOOP.2022.5. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECOOP.2022.5.

[6] Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. "Quantitative program reasoning with graded modal types". In: *Proc. ACM Program. Lang.* 3.ICFP (July 2019). DOI: 10.1145/3341714. URL: https://doi.org/10.1145/3341714.