Linear Exponentials as Graded Modal Types

Jack Hughes, Daniel Marshall, James Wood, Dominic Orchard

From the proceedings of Trends in Linear Logic and Applications 2021 (TLLA 2021, eds E. Pimentel, Ch. Retoré)
https://hal.archives-ouvertes.fr/TLLA2021

Linear Exponentials as Graded Modal Types

Jack Hughes* Daniel Marshall* James Wood†
Dominic Orchard*

Abstract

Graded type systems, which allow resource usage in programs to be tracked and reasoned about, are proliferating in recent works. These systems generalise ideas from Bounded Linear Logic (BLL), which allows the exponential modality ! from linear logic to be indexed by a bound on the number of times a formula is used. Graded systems generalise BLL's indexed modality to an arbitrary semiring of grades. Despite their relation to linear logic, particular choices in most graded type systems mean they cannot faithfully model ! due to the interaction between ! and $\otimes$; there are certain distributive laws that can be derived in graded type systems but cannot be derived in linear logic. We remedy this by enriching the structure of the grading with an additional operation, and show how this recovers the expressive power of Intuitionistic Multiplicative Exponential Linear Logic (IMELL) in a system with graded modal types. We briefly discuss our implementation involving this new operation for the graded modal language Granule.

1 Introduction

Linear logic separates linear formulas (those that must be used exactly once) from non-linear formulas (those that can be used arbitrarily often) by treating linearity as the default and using the 'exponential' modality ! to mark non-linear formulas. Bounded Linear Logic [GSS92] generalises ! to an indexed family of modalities $!_r A$ where r is a polynomial term over natural numbers capturing the maximum number of times that A can be used. Graded modal logics have generalised this idea further so that $\Box_r A$ captures a formula A which can be used according to a usage constraint r which is an element of an arbitrary pre-ordered semiring $\mathscr{R}$. One natural question is: can we recover the ! of linear logic via a particular choice of semiring? It turns out that many graded modal type systems bake in extra properties regarding the interaction of graded modalities and linear products $\otimes$ (multiplicative conjunction) such that ! cannot be precisely captured. In particular, whilst graded type theories (e.g. [OLEI19, AB20, BBN$^+$17, Bra21]) can derive a bi-implication

*University of Kent
†University of Strathclyde

between $\Box_r A \otimes \Box_r B$ and $\Box_r (A \otimes B)$ for any r, linear logic (LL) only derives $!A \otimes !B \multimap !(A \otimes B)$, i.e., for graded theories, denoted GR:

$$\vdash_{\text{GR}} \mathit{pull}_{\Box} : \Box_r A \otimes \Box_r B \multimap \Box_r (A \otimes B) \qquad \vdash_{\text{GR}} \mathit{push}_{\Box} : \Box_r (A \otimes B) \multimap \Box_r A \otimes \Box_r B$$
$$\vdash_{\text{LL}} \mathit{pull}_{!} : !A \otimes !B \multimap !(A \otimes B) \qquad \not\vdash_{\text{LL}} \mathit{push}_{!} : !(A \otimes B) \multimap !A \otimes !B$$

For example, $\mathit{push}_{\Box}$ can be defined in Granule (a linear language with graded modalities [OLEI19]) as follows (where `(a, b) [r]` is syntax for $\Box_r (A \otimes B)$):

```
push : ∀ {a b : Type, s : Semiring, r : s} . (a, b) [r] → (a [r], b [r])
push [(x, y)] = ([x], [y])
```
Granule

However, the power of the pattern matching used here does not translate to linear logic, and thus $\Box_r$ and $!$ are different. We propose a technique to restrict *push* by augmenting the grading semiring with additional structure such that the derivability of *push* can be controlled (or 'turned off'), enabling the intuitionistic $!$ to be recovered in a graded modal type system. This has applications to practical implementations of graded type systems such as Linear Haskell [BBN+17], Granule [OLEI19], and Idris 2 [Bra21], and on the theories of Grad [CEIEW21], Λ^p [AB20] and GranuleCore [OLEI19].

While we would like to be able to embed intuitionistic linear logic, we would like in other situations to have $\mathit{push}_{\Box}$. Graded systems have useful models [AB20] and a simple syntax for mixed linear and non-linear settings [BBN+17, Bra21]. This motivates accommodating both choices in the same system.

2 Core calculus

To explore this issue and formulate a solution, we focus on a core calculus based on the linear λ-calculus extended with products, units, pattern matching, and a *semiring graded necessity modality* [OLEI19], where for a pre-ordered semiring $(\mathscr{R}, *, 1, +, 0, \sqsubseteq)$, there is a family of types $\{\Box_r A\}_{r \in \mathscr{R}}$. This language constitutes a simplified monomorphic subset of Granule [OLEI19] which we will call GrCore, and closely resembles other related graded systems [AB20, Atk18, BBN+17, Bra21, CEIEW21].

Typing judgements are of the form $\Gamma \vdash t : A$, assigning type A to a term t under context Γ. Contexts Γ can contain linear and graded assumptions $\Gamma ::= \varnothing \mid \Gamma, x : A \mid \Gamma, y : [A]_r$, where x is a linear assumption and y is a graded assumption with grade $r \in \mathscr{R}$ describing how y can be used. Syntax and typing are given by the rules:

$$\frac{}{x : A \vdash x : A}\ \textsc{Var} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B}\ \textsc{Abs} \qquad \frac{\Gamma_1 \vdash t_1 : A \multimap B \qquad \Gamma_2 \vdash t_2 : A}{\Gamma_1 + \Gamma_2 \vdash t_1\, t_2 : B}\ \textsc{App} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma, x : [A]_1 \vdash t : B}\ \textsc{Der}$$

$$\frac{\Gamma \vdash t : A}{\Gamma + [\Delta]_0 \vdash t : A}\ \textsc{Weak} \qquad \frac{\Gamma, x : [A]_r, \Gamma' \vdash t : A \qquad r \sqsubseteq s}{\Gamma, x : [A]_s, \Gamma' \vdash t : A}\ \textsc{Approx} \qquad \frac{[\Gamma] \vdash t : A}{r * [\Gamma] \vdash [t] : \Box_r A}\ \textsc{Pr} \qquad \frac{}{\varnothing \vdash \mathsf{unit} : \mathbf{1}}\ \textsc{Unit}$$

$$\frac{\Gamma_1 \vdash t_1 : A \qquad \Gamma_2 \vdash t_2 : B}{\Gamma_1 + \Gamma_2 \vdash (t_1, t_2) : A \otimes B}\ \textsc{Prod} \qquad \frac{\Gamma_1 \vdash t_1 : A \quad \cdot \vdash p : A \rhd \Delta \quad \Gamma_2, \Delta \vdash t_2 : B}{\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\ p = t_1\ \mathbf{in}\ t_2 : B}\ \textsc{Let}$$

The (VAR), (ABS), and (APP) rules are the standard rules of the linear λ-calculus, augmented with a partial *context addition* operation $\Gamma + \Gamma'$ defined via semiring addition on graded assumptions which appear in both Γ and Γ' (where Γ and Γ' are disjoint in their linear assumptions): $(\Gamma, x : [A]_r) + (\Gamma', x : [A]_s) = (\Gamma + \Gamma'), x : [A]_{r+s}$.

The (WEAK) rule captures structural weakening of assumptions, where $[\Delta]_0$ denotes a context containing only assumptions graded by 0. A linear assumption may be converted to a graded assumption with the grade 1 via the rule for dereliction (DER). Grade approximation is provided by the (APPROX) rule, which allows a grade r to be converted to another grade s, provided that s *approximates* r (where $\sqsubseteq$ is the pre-order given by the grades' semiring). Graded modalities are introduced by the promotion (PR) rule, scaling the assumptions in the context $[\Gamma]$ (where a $[\Gamma]$ denotes a context containing only graded assumptions) via semiring multiplication. Products are typed by the (PROD) rule, where the contexts used to type the pair's constituent subterms are added together.

Elimination of products, units, and graded modal types is achieved via the (LET) rule, which applies pattern matching to a term t_1 to produce a context of typed binders Δ to be used in t_2. Patterns p are typed by judgements of the form $?r \vdash p : A \rhd \Delta$, meaning that a pattern p has type A and produces a context of typed binders Δ. Optional grade information is denoted via $?r$, defined syntactically as $?r ::= \cdot \mid r$, where r indicates that the rule takes place under an unboxing pattern. We then define the rules for pattern typing as:

$$\frac{}{\cdot \vdash x : A \rhd x : A}\ \text{PVAR} \qquad \frac{\cdot \vdash p_1 : A \rhd \Gamma_1 \quad \cdot \vdash p_2 : B \rhd \Gamma_2}{\cdot \vdash (p_1, p_2) : A \otimes B \rhd \Gamma_1, \Gamma_2}\ \text{PPROD} \qquad \frac{r \vdash p : A \rhd \Gamma}{\cdot \vdash [p] : \Box_r A \rhd \Gamma}\ \text{PBOX}$$

$$\frac{}{r \vdash x : A \rhd x : [A]_r}\ [\text{PVAR}] \qquad \frac{r \vdash p_1 : A \rhd \Gamma_1 \quad r \vdash p_2 : B \rhd \Gamma_2}{r \vdash (p_1, p_2) : A \otimes B \rhd \Gamma_1, \Gamma_2}\ [\text{PPROD}] \qquad \frac{}{?r \vdash \mathsf{unit} : \mathbf{1} \rhd \varnothing}\ \text{PUNIT}$$

The (PBOX) rule types 'unboxing' patterns by propagating grade information r into the typing of the sub-pattern p. We therefore require two rules each for the typing of variable and product patterns: a rule for linear patterns (PVAR and PPROD) and a rule for patterns which take place inside an unboxing ([PVAR] and [PPROD]). In the case of the latter [PVAR] rule, a binding is produced with the grade of the enclosing box's grade r. Likewise in [PPROD], grade information is propagated to the typing of the product's constituent sub-patterns. Unit patterns are not affected by the enclosing grade and so have one rule PUNIT above.

3 Dissecting the problem

In the above system, we can construct the following derivation, nesting a tensor pattern inside an unboxing pattern:

$$\dfrac{\Gamma_1 \vdash t_1 : \Box_r(A \otimes B) \quad \dfrac{\dfrac{\dfrac{}{r \vdash x : A \rhd x : [A]_r}\,[\text{PVAR}] \quad \dfrac{}{r \vdash y : B \rhd y : [B]_r}\,[\text{PVAR}]}{r \vdash (x,y) : A \otimes B \rhd x : [A]_r, y : [B]_r}\,[\text{PPROD}]}{\cdot \vdash [(x,y)] : \Box_r(A \otimes B) \rhd x : [A]_r, y : [B]_r}\,\text{PBOX} \quad \Gamma_2, x : [A]_r, y : [B]_r \vdash t_2 : C}{\Gamma_1 + \Gamma_2 \vdash \textbf{let}\ [(x,y)] = t_1\ \textbf{in}\ t_2 : C}\,\text{LET}$$

This prevents us capturing linear logic's ! as some $\Box_r$ in a graded system, since from the above we can derive $push_{\Box} : \Box_r(A \otimes B) \multimap \Box_r A \otimes \Box_r B$ with $t_2 = ([x],[y])$. We briefly demonstrate how this problem affects languages other than Granule.

Linear Haskell As of GHC 9, Haskell implements a graded type system [BBN+17]. Function types `(a %r -> b)` now have a multiplicity annotation `r`. The annotation `r` is either `'One` or `'Many`, corresponding to linear and unrestricted use of the argument, respectively. Following the presentation in [HVO20], we can define a graded modality in Linear Haskell via the `Box` data type:

```
data Box r a where { Box :: a %r -> Box r a }
```
Haskell

We might hope to define $!A = $ `Box %'Many A`, but this does not provide an accurate definition of the exponential modality as we can define the *push* distributive law:

```
push :: Box r (a, b) %1 -> (Box r a, Box r b)
push (Box (x, y)) = (Box x, Box y)
```
Haskell

The `Box %'Many A` type is equivalent to a wrapper data type `Unrestricted` which is described as being "very much like $!a$ in linear logic" [BBN+17]. The existence of `push` shows one way in which $!a$ and `Unrestricted a` behave differently.

Idris 2 Idris 2 is based on Quantitative Type Theory (QTT) [Bra21, Atk18], though, as we discuss below, QTT does not admit *push*, while Idris 2 does. In both Idris 2 and QTT, every variable has a *quantity* associated with it, either 0, 1 or ω (unrestricted). Idris 2 does not support direct abstraction over multiplicity, so instead of a general $\Box_r$ type, we define a type corresponding to $\Box_\omega$ and the Haskell `Unrestricted` data type:

```
data Unrestricted : Type -> Type  where  Box : a -> Unrestricted a
```
Idris 2

(Note here that, as in Haskell, variables have an unrestricted multiplicity by default.) Similarly to Haskell, we might hope that we can define $!A = $ `Unrestricted A`. However, again this is not possible as we can redefine the same `push` law which is not derivable in linear logic.

```
push : Unrestricted (a, b) -> (Unrestricted a, Unrestricted b)
push (Box (x, y)) = (Box x, Box y)
```

Idris 2

QTT Quantitative Type Theory [Atk18, McB16] is an example of a graded type system without the *push* behaviour, thus being a conservative extension of linear logic. It does this via a more restricted rule for tensor products than the other systems mentioned in this paper. Below, we give the rule for tensor products as it appears in the original paper, and also a simplified version comparable with propositional linear logic.

In QTT, conclusions are annotated either 1 or 0. Rules are given for both possibilities simultaneously, with σ ranging over $\{0,1\}$. Conclusions annotated 0 are used only for constructing (dependent) types, so we need only consider the $\sigma = 1$ case. Further simplifying to get the elimination rule for a simple tensor product, we ignore all of the dependency in T and U, and set π (the grade of the first component of the pair) to 1. All of these simplifications get us the second rule below. Essentially the same simplified rule appears in the system $\lambda\mathscr{R}$, which is equivalent to intuitionistic linear logic [WA21].

$$\frac{0\Gamma_1, z \overset{0}{:} (x \overset{\pi}{:} S) \otimes T \vdash U \qquad \Gamma_1 \vdash M \overset{\sigma}{:} (x \overset{\pi}{:} S) \otimes T \qquad \Gamma_2, x \overset{\sigma\pi}{:} S, y \overset{\sigma}{:} T \vdash N \overset{\sigma}{:} U[(x,y)/z] \qquad 0\Gamma_1 = 0\Gamma_2}{\Gamma_1 + \Gamma_2 \vdash \text{let}_{x \overset{\pi}{:} S.T}\ (x,y) = M \text{ in } N \overset{\sigma}{:} U[M/z]}$$

$$\frac{0\Gamma_1 = 0\Gamma_2 \qquad \Gamma_1 \vdash M : S \otimes T \qquad \Gamma_2, x \overset{1}{:} S, y \overset{1}{:} T \vdash N : U}{\Gamma_1 + \Gamma_2 \vdash \text{let } (x,y) = M \text{ in } N : U}$$

Abel & Bernardy's Λ^p Abel and Bernardy defined a graded type system Λ^p and denotational semantics admitting *push* [AB20]. They have the following rule for pattern matching on products:

$$\frac{\gamma\Gamma \vdash t : A \times B \qquad \delta\Gamma, x : {}^q A, y : {}^q B \vdash u : C}{(q\gamma + \delta)\Gamma \vdash \text{let } (x,y) = {}^q t \text{ in } u : C}$$

This is akin to the elimination behaviour underneath a graded modality in GRCORE as shown at the start of this section. Compared to the simplified QTT pattern matching rule, the essential difference with QTT becomes apparent: the Λ^p rule has the extra grade q, allowing not just one $A \times B$ to yield one A and one B, but q-many inhabitants of $A \times B$ (signified by the fact that γ is multiplied by q in the conclusion) to yield q-many As and q-many Bs. This stronger elimination rule requires a semantic version of *push* in the models, but such models still include the one used to show that every linear term is a permutation.

4 Solution

The key to controlling the 'push' behaviour is to control pattern matching on tensor products underneath a graded modality. We thus extend the pre-ordered semiring

with a partial commutative and associative binary operator $\curlywedge : \mathscr{R} \times \mathscr{R} \nrightarrow \mathscr{R}$ (pronounced 'hsup') used in the typing of patterns for tensor products as:

$$\frac{r \vdash p_1 : A \rhd \Gamma_1 \quad s \vdash p_2 : B \rhd \Gamma_2}{r \curlywedge s \vdash (p_1, p_2) : A \otimes B \rhd \Gamma_1, \Gamma_2} \text{ [PPROD]}$$

Thus, to pattern match on a pair inside of an unboxing pattern, we must be able to compute $r \curlywedge s$, where r and s are the grades used to type the pair's constituent sub-patterns. For the semirings where we wish to permit the full $\otimes$ distributive law we can set $\curlywedge$ to be the (partial) least-upper bound $\sqcup$ derived from the pre-ordering which recovers the existing pattern matching typing, e.g., for the exact usage semiring ($\mathbb{N}$ with pre-ordering as equality) this means that $\curlywedge$ is only defined when $r = s$.

For graded modalities where we wish to disallow the 'push' behaviour we can leave $r \curlywedge s$ undefined for the relevant grades. In this way, we can recover the power of linear logic's ! in our calculus via the pre-ordered semiring $\{0, 1, \omega\}$ (none-one-tons) with $!A = \Box_\omega A$. The semiring is defined with $r + s = r$ if $s = 0$, $r + s = s$ if $r = 0$ and otherwise ω, and $r * 0 = 0 * r = 0$, $r * \omega = \omega * r = \omega$ (for $r \neq 0$), and $r * 1 = 1 * r = r$ with ordering $0 \sqsubseteq \omega$ and $1 \sqsubseteq \omega$. This semiring allows us to represent both linear and non-linear use: variables graded with 1 must be used linearly, with 0 must be discarded, and a grade of ω permits unconstrained use à la linear logic's !. We then define $r \curlywedge s$ for this semiring as:

$$r \curlywedge s = \begin{cases} 1 & r = 1 \wedge s = 1 \\ \bot & \text{otherwise} \end{cases} \tag{1}$$

i.e., if either of the grades is not 1, then $\curlywedge$ is undefined and we cannot apply the [PPROD] pattern typing rule, thus disallowing $!(A \otimes B) \to !A \otimes !B$ and recovering the strength of ! for intuitionistic multiplicative exponential linear logic (IMELL):

Theorem 1 (Equivalent expressivity)**.** *The adjusted* GRCORE *calculus, for the none-one-tons semiring with $!A = \Box_\omega A$, has the same expressive power as IMELL.*

The equivalence in expressivity follows by a translation. In the case of the GRCORE $\to$ *IMELL direction, we interpret contexts and types directly, but a type-directed translation is then required for the term. In the opposite direction, a more syntactic translation is possible, since we follow Benton et al.'s full term assignment for IMELL [BBdPH93]. The appendix [HMWO21] provides the proof.*

- *(*GRCORE *into IMELL)* $\Gamma \vdash t : A \quad \Rightarrow \quad \exists M.\, [\![\Gamma]\!] \vdash_{\mathsf{IMELL}} M : [\![A]\!]$
- *(IMELL into* GRCORE*)* $\Gamma \vdash_{\mathsf{IMELL}} M : T \quad \Rightarrow \quad [\![\Gamma]\!] \vdash [\![M]\!] : [\![T]\!]$

The translation from GRCORE to IMELL maps both $\Box_0 A$ and $\Box_\omega A$ to $![\![A]\!]$ and $\Box_1 A$ to just $[\![A]\!]$, and similarly for graded assumptions (e.g., $x : [A]_\omega$ to $x :![\![A]\!]$).

The operation $\curlywedge$ is inspired by the coeffect calculus of Petricek et al. [POM14] whose graded type system includes $\curlywedge$ as an operation to control splitting of resources to subterms, modelled by colax monoidality of a graded comonad $n_{r,s,A,B} : D_{r \curlywedge s}(A \otimes B) \to (D_r A \otimes D_s B)$.

5 Implementation in Granule

The partial operation $\asymp$ is implemented within the Granule compiler as part of the type checker. Granule has a more general notion of type constructors than we defined in our calculus here, including user defined algebraic data types (ADTs) as well as generalised algebraic data types (GADTs). Thus, instead of occurring solely in the presence of product patterns, $\asymp$ is applied whenever a pattern match occurs against a constructor with at least one sub-pattern.

We implement $\asymp$ via a constraint on the grades given by the data constructor pattern (inside a box pattern) which is then discharged via an SMT solver. If such a constraint is satisfiable, then the type checking of the pattern may proceed. The nature of information flow in the type checking of a Granule pattern match is such that the grades r and s in $r \asymp s$ are necessarily the same, i.e. $r \asymp r$, and we require that this is then defined at r. For the 0-1-ω semiring (called `LNL` in Granule and written with elements `0`, `1`, `Many`), where ω is used to model the exponential modality, $r \asymp r$ is then defined only when $r = 1$. We can therefore no longer write the (previously type checkable) Granule program:

```
1 push : ∀ {a b : Type} . (a, b) [Many] → (a [Many], b [Many])
2 push [(x, y)] = ([x], [y])
```

Ill-typed Granule

6 Conclusions

Graded types trace their origins from intuitionistic linear logic. The inability in many of these systems to capture the strength of ! therefore represents an interesting divergence in their expressive power from linear logic. The solution described here allows our calculus to retain both the power of *push* and of !, by defining when grades can be 'pushed inside a tensor' via the algebraic structure of grades. This approach is readily adaptable to the other graded systems described here. This has implications for other ongoing work on the automatic derivation of distributive laws for arbitrary types in a graded type system [HVO20]. The ability to define where *push* is possible for a given semiring imposes an additional constraint on when the *push* distributive law for a type can be automatically calculated.

Acknowledgements Thanks to Benjamin Moon and Harley Eades III for their comments and discussion, and to the TLLA anonymous reviewers for their helpful comments. This work was supported by EPSRC Doctoral Training Awards (Hughes, Marshall, Wood) and by EPSRC grant EP/T013516/1 (*Verifying Resource-like Data Use in Programs via Types*).

References

[AB20] Andreas Abel and Jean-Philippe Bernardy. A unified view of modalities in type systems. *Proc. ACM Program. Lang.*, 4(ICFP), 2020.

[Atk18] Robert Atkey. Syntax and semantics of quantitative type theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, page 56–65, 2018.

[BBdPH93] P. N. Benton, Gavin M. Bierman, Valeria de Paiva, and Martin Hyland. A term calculus for intuitionistic linear logic. In *Typed Lambda Calculi and Applications, TLCA '93*, volume 664 of *Lecture Notes in Computer Science*, pages 75–90. Springer, 1993.

[BBN+17] Jean-Philippe Bernardy, Mathieu Boespflug, Ryan R. Newton, Simon Peyton Jones, and Arnaud Spiwack. Linear Haskell: Practical linearity in a higher-order polymorphic language. *Proc. ACM Program. Lang.*, 2(POPL), December 2017.

[Bra21] Edwin Brady. Idris 2: Quantitative type theory in practice, 2021. To appear at ECOOP 2021.

[CEIEW21] 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), 2021.

[GSS92] Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. Bounded linear logic: a modular approach to polynomial-time computability. *Theoretical Computer Science*, 97(1):1–66, 1992.

[HMWO21] Jack Hughes, Daniel Marshall, James Wood, and Dominic Orchard. Linear Exponentials as Graded Modal Types (Appendix). Zenodo, `https://doi.org/10.5281/zenodo.4976702`, June 2021.

[HVO20] Jack Hughes, Michael Vollmer, and Dominic Orchard. Deriving distributive laws for graded linear types. *TLLA 2020*, 2020.

[McB16] Conor McBride. I Got Plenty o' Nuttin'. In Sam Lindley, Conor McBride, Philip W. Trinder, and Donald 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 *Lecture Notes in Computer Science*, pages 207–233. Springer, 2016.

[OLEI19] Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. Quantitative program reasoning with graded modal types. *Proc. ACM Program. Lang.*, 3(ICFP), July 2019.

[POM14] Tomas Petricek, Dominic A. Orchard, and Alan Mycroft. Coeffects: a calculus of context-dependent computation. In *Proceedings of the ICFP 2014*, pages 123–135. ACM, 2014.

[WA21] James Wood and Robert Atkey. A linear algebra approach to linear metatheory. *CoRR*, abs/2005.02247, 2021.