



Kent Academic Repository

Liepelt, Vilem, Marshall, Danielle and Orchard, Dominic (2026) *Same coeffect, different base: Connecting two dominant approaches to graded types*. Proceedings of the ACM on Programming Languages, 10 (ICFP). pp. 721-750.

Downloaded from

<https://kar.kent.ac.uk/116035/> The University of Kent's Academic Repository KAR

The version of record is available from

<https://doi.org/10.1145/3828697>

This document version

Publisher pdf

DOI for this version

Licence for this version

CC BY (Attribution)

Additional information

Versions of research works

Versions of Record

If this version is the version of record, it is the same as the published version available on the publisher's web site. Cite as the published version.

Author Accepted Manuscripts

If this document is identified as the Author Accepted Manuscript it is the version after peer review but before type setting, copy editing or publisher branding. Cite as Surname, Initial. (Year) 'Title of article'. To be published in **Title of Journal**, Volume and issue numbers [peer-reviewed accepted version]. Available at: DOI or URL (Accessed: date).

Enquiries

If you have questions about this document contact ResearchSupport@kent.ac.uk. Please include the URL of the record in KAR. If you believe that your, or a third party's rights have been compromised through this document please see our [Take Down policy](https://www.kent.ac.uk/guides/kar-the-kent-academic-repository#policies) (available from <https://www.kent.ac.uk/guides/kar-the-kent-academic-repository#policies>).

Same Coeffect, Different Base: Connecting Two Dominant Approaches to Graded Types

VILEM LIEPELT, University of Kent, UK

DANIELLE MARSHALL, Royal Holloway, University of London, UK

DOMINIC ORCHARD, University of Kent, UK and University of Cambridge, UK

Graded types provide a way to augment a type system with fine-grained information, e.g., to track side effects or context dependence and resource use (called *coeffects*). Graded types for coeffects have found their way into languages such as Haskell, Idris, and Granule, enabling resourceful reasoning via coeffect analysis with varying levels of generality. Two separate lineages of graded coeffect system have emerged in the last decade: those in which coeffect annotations are pervasive, requiring annotations on function types (which we call *graded-base*) and those in which coeffects are added by way of a graded modal type operator atop linear types (which we call *linear-base*). The latter has its origins in Girard’s Linear Logic which has been a rich humus for programming language research focused on resources, whereas the graded-base approach emerged in the mid-2010s, seeing rapid adoption in programming language theory and practice, e.g. in QTT and Linear Haskell. The relationship between these two styles has however remained an open question. We answer this question by giving translations between pairs of calculi of both lineages that we prove type-, grade- and operational-semantics preserving. We show that the same notions of context dependence can be expressed in either style, building a bridge between the two lineages that enables transfer of results and ideas, while helping language designers to make better informed choices.

CCS Concepts: • **Theory of computation** → **Linear logic; Modal and temporal logics; Program specifications; Program verification; Functional constructs; Type theory.**

Additional Key Words and Phrases: graded modal types, graded types, linear types, coeffects

ACM Reference Format:

Vilem Liepelt, Danielle Marshall, and Dominic Orchard. 2026. Same Coeffect, Different Base: Connecting Two Dominant Approaches to Graded Types. *Proc. ACM Program. Lang.* 10, ICFP, Article 299 (August 2026), 30 pages. <https://doi.org/10.1145/3828697>

1 Introduction

The behaviour of useful programs often depends on execution context—whether through low-level factors like hardware capabilities (e.g., the available I/O devices) or higher-level properties like information flow security. Recent work has explored tracking such *coeffects* via the type system [Brunel et al. 2014; Ghica and Smith 2014; Petricek et al. 2014, 2013], providing an analysis of context dependence, reified into the type system as an intrinsic part of the language’s meaning.

There exist in the literature two parallel threads on coeffect types systems: those which build atop Linear Logic via the linear λ -calculus, which we call *linear-base* systems, and those which build atop the simply-typed λ -calculus or System F but with pervasive annotations on all assumptions and function arrows, which we refer to as *graded-base* systems (for reasons which will become clear

Authors’ Contact Information: Vilem Liepelt, University of Kent, School of Computing, UK, mail@vilem.net; Danielle Marshall, Royal Holloway, University of London, Department of Computer Science, UK, Danielle.Marshall@rhul.ac.uk; Dominic Orchard, University of Kent, School of Computing, UK and University of Cambridge, Department of Computer Science and Technology, UK, D.A.Orchard@kent.ac.uk.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2026 Copyright held by the owner/author(s).

ACM 2475-1421/2026/8-ART299

<https://doi.org/10.1145/3828697>

later). How these two approaches precisely relate is a long-standing open question that this paper answers, giving type system and language designers a map of the graded type system landscape.

Generalising linearity to graded modal types. Girard [1987] introduced Linear Logic, seeking more fine-grained control over the structural rules implicitly present in classical and intuitionistic logic. The relevance to computer science in particular was very much at the forefront of the motivations for Girard [1987]: “One of the main outputs of Linear Logic seems to be in computer science”.

The more conscious use of structural properties in Linear Logic allows for reasoning about the cost of computations and other properties, which are referred to under the umbrella term *resources*. As predicted by Girard himself, Linear Logic has become the mother to a profusion of type systems for reasoning about various notions of resource [Walker 2005]. Examples in the literature include access to hardware devices [Petricek et al. 2013], timing [Ghica and Smith 2014] and differential privacy [Gaborardi et al. 2013]. These developments stand on the shoulders of a refinement of Linear Logic, in the form of Bounded Linear Logic (BLL), due to Girard et al. [1992] in the early 1990s. While Linear Logic delineates the world into ‘use strictly once’ (A) and ‘arbitrary usage’ ($!A$), i.e. zero or more times, BLL indexes the *of course* modality with a polynomial bound, where $!_n A$ is usable up to n times. For example, the following type describes some higher-order function which takes a function and composes it with itself, thus using it twice: $!_2(A \multimap A) \multimap (A \multimap A)$.

A significant evolution came in 2014, when Brunel et al. [2014] and Ghica and Smith [2014] independently generalised the natural number indices in BLL to elements of a semiring $\mathcal{R}$ to capture more properties than just variable (re)use, including privacy levels and hardware schedules. This development then allowed other strands of research on *coeffects* (general context analysis) to be expressible in this linear-logic-derived setting.

The generalisation of BLL gave rise to a more general paradigm of *graded modal types*. Gaborardi et al. [2016] considered a calculus with a semiring-graded modality $\Box_r A$ for coeffects alongside a monoid-graded modality for effects $\Diamond_f A$, modelled respectively by graded comonads and graded monads. Orchard et al. [2019] crystallised the idea of graded modal types as one in which “graded modal types carry information about semantic structure” where the algebraic structure of the indices reflects underlying program structure. Others have followed a similar approach [Hughes and Orchard 2020; Hughes et al. 2020; Wood and Atkey 2020, 2022]. In all these systems, Linear Logic has been at the core. We thus refer to these as *linear-base systems*.

Coeffect systems with pervasive grading and without inherent linearity. In parallel, Petricek et al. [2014, 2013] dualised type-and-effect systems to *coeffect systems*, capturing how programs depend on their context. Their approach built on top of the simply-typed rather than the linear λ -calculus. Every assumption has an associated coeffect (grade) and function arrows must also be annotated with a coeffect describing how the parameter is used. In the case of the ‘apply twice’ higher-order function, the analogous coeffect type is $(A \xrightarrow{1} A) \xrightarrow{2} (A \xrightarrow{1} A)$. If the parameter function is generalised from coeffect 1 to some arbitrary use r , then the type would be $(A \xrightarrow{r} A) \xrightarrow{2} (A \xrightarrow{r*r} A)$.

Coeffect systems were noted by Brunel et al. [2014] to relate to a generalised BLL system. Since then, many other graded systems have taken a similar approach with no base notion of linearity, but where every assumption has a grade and function arrows are annotated, e.g., Linear Haskell (GHC/Haskell with the LinearTypes extension) [Bernardy et al. 2018], the general system of Abel and Bernardy [2020], Quantitative Type Theory, first due to McBride [2016] then refined by Atkey [2018] and used as the basis for Idris 2 [Brady 2021a], the GRAD calculus of Choudhury et al. [2021], Graded Modal Type Theory [Moon et al. 2021], and various others [Abel et al. 2023; Hughes and Orchard 2024; Liepelt et al. 2026b]. This style has even been adapted to an imperative setting in the coeffectful Multi-Graded Featherweight Java [Bianchini et al. 2023a, 2022]. A similar type system

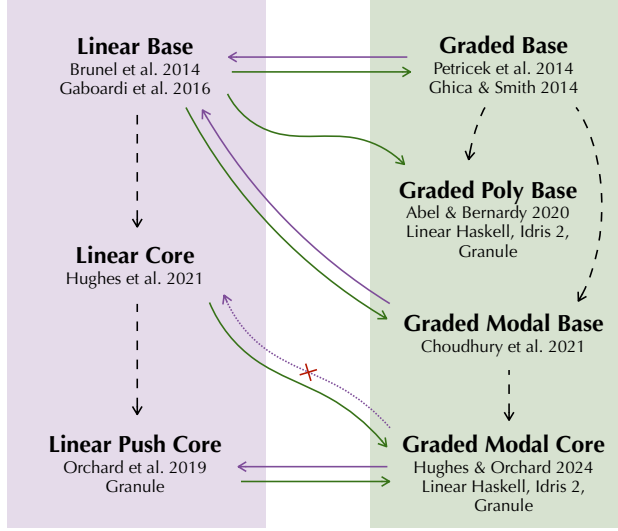


Fig. 1. Relationship between graded calculi studied in this paper. For each calculus, a selection of representative papers and programming systems is given (that typically also extend the given calculus with orthogonal features, such as dependent types, which are not relevant to this paper). A dashed arrow denotes a subset relationship. A solid arrow denotes a semantics-preserving translation which we define in the present paper. A crossed-out arrow with a dotted shaft refers to a translation that is not possible.

structuring arose in a separate thread of work from [Wright and Baker-Finch \[1993\]](#) whose lineage has been recently been traced and summarised [[Wright 2026](#)]. In these approaches, grading is pervasive and there is no inherent notion of linearity as in Linear Logic (although linearity can be achieved as an emergent phenomenon). We refer to these as *graded-base systems*.

So what is the relationship? Are graded coeffects with a linear basis equally expressive as those without an inherent notion of linearity? This is the open question that has become ever more glaring as these works have developed. We seek a more clear understanding of their relationship and relative expressivity in order to inform future developments and choices.

Contributions and roadmap. Our aim is to elucidate the connection between the two distinct strands of coeffect reasoning that have emerged in the literature and in programming language implementations, which we call *linear-base* vs. *graded-base* systems. We show that, depending on the substrate, both systems can be given mutual translations to each other in a sound way.

Figure 1 gives an overview of these two strands, instantiated as several concrete calculi with differing features which we summarise here with relevant section numbers:

Linear-base systems are shown on the left. Here **Linear Base** (2.1) represents a linear λ -calculus extended with graded modal necessity $\Box_r A$, corresponding to a generalised *of course* modality. **Linear Core** (5.1) augments this with products and sums (multiplicative conjunction and additive disjunction). **Linear Push Core** (5.5) further adds constructs for allowing distributivity of graded modalities over products and sums, thereby losing the correspondence to Linear Logic, as shown by [Hughes et al. \[2021\]](#). This system can model the interaction between grades and products/sums in Granule as presented by [Orchard et al. \[2019\]](#).

On the right then we have graded-base systems, for all of which **Graded Base** (2.2) is the underlying calculus, corresponding to pervasively-graded simply-typed λ -calculus in the style

of Petricek et al. [2014], Ghica and Smith [2014], and Linear Haskell as presented by Bernardy et al. [2018]. **Graded Poly Base** (4.1) augments this with System-F-style type abstraction as in the work of Abel and Bernardy [2020], presenting a substrate modelling more closely the actual implementation of Linear Haskell, as well as Idris 2 (where the dependent types have no interaction with grades) [Brady 2021a] and also Granule with the GradedBase language extension [Hughes and Orchard 2024]. **Graded Modal Base** (3.1) instead augments Graded Base by adding a graded modal necessity akin to the one in Linear Base. **Graded Modal Core** (5.2) augments the latter system with products and sums and thereby captures the interaction between grades and algebraic datatypes available e.g. in Granule with the GradedBase extension, and Linear Haskell and Idris 2 (where a modal operator can be encoded via an algebraic datatype).

The paper is then structured as follows:

- Section 2 introduces Linear Base and Graded Base. Section 2.3 introduces syntax for translation, gives a **translation from Graded Base to Linear Base** (2.3.1) with soundness results, and then the converse **from Linear Base to Graded Base** via a CPS-like translation (2.3.2).
- Section 3 extends Graded Base to Graded Modal Base (3.1), gives a **translation from Linear Base to Graded Modal Base** (3.2.1) with soundness results, gives a **translation from Graded Modal Base to Linear Base** (3.2.3) with soundness results, investigates the nature of the relationship between both translations (3.2.4), and gives a possible generalisation of elimination for the graded modality that appears in the literature, along with tradeoffs (3.3).
- Section 4 gives an alternative strategy for adding a graded modality to Graded Base, via an encoding inspired by Linear Haskell, extends Graded Base to Graded Poly Base (4.1), and gives a **translation from Linear Base to Graded Poly Base** (4.1.1) with soundness results.
- Section 5 adds product and sum types, allowing us to study the interaction of coeffects with data structures. It extends Linear Base to Linear Core (5.1), extends Graded Modal Base to Graded Modal Core (5.2), gives a **translation from Linear Core to Graded Modal Core** (5.3) with soundness results, shows **there is no sound translation from Graded Modal Core to Linear Core** (5.4), extends Linear Core to Linear Push Core (5.5), and gives a **translation from Graded Modal Core to Linear Push Core** with soundness results.
- Section 6 asks: *is one approach better than the other?* (6.1), gives some simple example programs in languages that implement features from the two lineages studied (6.2, considers some related ideas (6.3-6.6), and concludes (6.7).

2 The Two Calculi: Linear and Graded Base

We present in turn two representative calculi for the two dominant styles of graded coeffects: Linear Base (Section 2.1) and Graded Base (Section 2.2).

2.1 Linear Base

As discussed in the introduction, linear-base systems take linear λ -calculus as their core. Such systems include the core quantitative coeffect calculus of Brunel et al. [2014], the combined graded system for effects and coeffects of Gaboardi et al. [2016], $\lambda\mathcal{R}$ of Wood and Atkey [2020], the core of Granule by Orchard et al. [2019], and associated work [Hughes et al. 2020].

The term syntax of our idealised system, Linear Base, is as follows:

$$t ::= x \mid t_1 t_2 \mid \lambda x. t \mid [t] \mid \text{let } [x] = t_1 \text{ in } t_2 \quad (\text{terms})$$

Terms comprise variables, applications, and abstraction as standard. Graded modal introduction (formally *promotion*, informally *boxing*) is via $[t]$, which for now we can think of as wrapping

a term in a constructor. Graded modal elimination eliminates this constructor via a *let* binding, unwrapping the boxed value, which we refer to as *unboxing*.

Linear Base types include a linear function space, a *graded modality* which can be understood as generalising Linear Logic's exponential modality $!$, and a base type K so that types are well-founded:

$$A ::= A \multimap B \mid \Box_r A \mid K \quad (\text{types})$$

Here r is drawn from a pre-ordered semiring $(\mathcal{R}, \cdot, 1, +, 0, \sqsubseteq)$ (where addition $+$ and multiplication $\cdot$ must be monotonic wrt. to $\sqsubseteq$) parameterising the calculus.

An intuition for $\Box_r A$ is that it describes some capability r with which terms of type A contained inside can be used. For example, for the natural numbers semiring $\mathbb{N}$ then $\Box_2 A$ denotes two copies of an A , or in essence that its contents can be used twice.

Typing judgments are of the form $\Gamma \vdash_L t : A$, relating a context Γ , term t , and type A . Contexts comprise linear assumptions $x : A$ and graded assumptions $x : [A]_r$ with grade $r \in \mathcal{R}$, of the form:

$$\Gamma ::= \emptyset \mid \Gamma, x : A \mid \Gamma, x : [A]_r \quad (\text{contexts})$$

Contexts can be *added* if they are disjoint in their linear assumptions and using semiring addition to combine grades of any graded assumptions shared between two contexts:

Definition 2.1 (Context addition). For contexts Γ_1, Γ_2 , then $\Gamma_1 + \Gamma_2$ is a partial operation computing the combined context providing *contraction* on graded assumptions. Context addition is specified:

$$\begin{aligned} \Gamma + \emptyset &= \Gamma & (\Gamma_1, x : [A]_r) + (\Gamma_2, x : [A]_s) &= (\Gamma_1 + \Gamma_2), x : [A]_{r+s} \\ \emptyset + \Gamma &= \Gamma & (\Gamma_1, x : [A]_r) + \Gamma_2 &= (\Gamma_1 + \Gamma_2), x : [A]_r & \text{if } x \notin \text{dom}(\Gamma_2) \\ & & (\Gamma_1, x : A) + \Gamma_2 &= (\Gamma_1 + \Gamma_2), x : A & \text{if } x \notin \text{dom}(\Gamma_2) \end{aligned}$$

That is, we union the disjoint parts of the contexts, and for the non-disjoint parts of the contexts we merge with semiring addition. If the two contexts share linear assumptions or if the two contexts share graded assumptions that do not agree on the type, then the addition is undefined (in practical terms, this amounts to a linearity error). Contexts are unordered, i.e., *exchange* is permitted.

We define two predicates for denoting contexts containing only graded assumptions and those containing graded assumptions all with the same grade:

Definition 2.2 (Graded contexts). A context Γ has property $\text{graded}(\Gamma)$ if it contains only graded assumptions, and $\text{graded}(\Gamma, r)$ if it contains only graded assumptions which are of grade r . These predicates are defined inductively:

$$\frac{}{\text{graded}(\emptyset)} \quad \frac{\text{graded}(\Gamma)}{\text{graded}((\Gamma, x : [A]_r))} \quad \frac{}{\text{graded}(\emptyset, r)} \quad \frac{\text{graded}(\Gamma, r)}{\text{graded}((\Gamma, x : [A]_r), r)}$$

Lastly, contexts which contain only graded assumptions can be *scaled* by a grade:

Definition 2.3 (Scalar multiplication). A context Γ where $\text{graded}(\Gamma)$ can be *scaled* $r \cdot \Gamma$, defined:

$$r \cdot \emptyset \triangleq \emptyset \quad r \cdot (\Gamma', x : [A]_s) \triangleq (r \cdot \Gamma'), x : [A]_{r \cdot s}$$

Figure 2 defines typing rules. The *variable* (VAR_L) and *abstraction* (ABS_L) rules are standard for the linear λ -calculus. The *application* (APP_L) rule is close to that of the λ -calculus, but the separate contexts of the premises are added in the conclusion. *Weakening* (WEAK_L) allows unused assumptions to be added, as long as they are graded at the zero of the semiring. *Dereliction* (DER_L) treats a linear assumption as a graded assumption at the one element of the semiring, which connects the linear and non-linear worlds. *Promotion* (PR_L) introduces grading to terms, making use of scalar context multiplication to propagate the resource requirements to the assumptions. The *let*

$\Gamma \vdash_L t : A$

 Typing rules

$$\begin{array}{c}
\frac{}{x : A \vdash_L x : A} \text{VAR}_L \quad \frac{\Gamma, x : A \vdash_L t : B}{\Gamma \vdash_L \lambda x. t : A \multimap B} \text{ABS}_L \quad \frac{\Gamma_1 \vdash_L t_1 : A \multimap B \quad \Gamma_2 \vdash_L t_2 : A}{\Gamma_1 + \Gamma_2 \vdash_L t_1 t_2 : B} \text{APP}_L \\
\\
\frac{\Gamma \vdash_L t : A \quad \text{graded}(\Gamma', 0)}{\Gamma, \Gamma' \vdash_L t : A} \text{WEAK}_L \quad \frac{\Gamma, x : A \vdash_L t : B}{\Gamma, x : [A]_1 \vdash_L t : B} \text{DER}_L \quad \frac{\Gamma \vdash_L t : A \quad \text{graded}(\Gamma)}{r \cdot \Gamma \vdash_L [t] : \Box_r A} \text{PR}_L \\
\\
\frac{\Gamma_1 \vdash_L t_1 : \Box_r A \quad \Gamma_2, x : [A]_r \vdash_L t_2 : B}{\Gamma_1 + \Gamma_2 \vdash_L \text{let } [x] = t_1 \text{ in } t_2 : B} \text{LET}_L \quad \frac{\Gamma, x : [B]_r \vdash_L t : A \quad r \sqsubseteq s}{\Gamma, x : [B]_s \vdash_L t : A} \text{APPROX}_L
\end{array}$$

Fig. 2. Linear Base typing.

or *unboxing* rule (LET_L) eliminates an r -graded modality, making the unboxed value of t_1 available in the body of t_2 according to the capabilities afforded by r . *Approximation* (APPROX_L) allows treating an r -graded assumption as s -graded if $r \sqsubseteq s$. Note the direction of the graded inequality in the premise is according to [Petricek et al. \[2014\]](#). The opposite polarity is also used in the literature, for example by [Brunel et al. \[2014\]](#), without any semantic difference.

Example 2.4. The natural numbers semiring $(\mathbb{N}, *, 1, +, 0, \equiv)$ whose ordering is equality (i.e., a discrete ordering) captures a notion of exact usage. For example, the following captures the typing of a linear function f that takes two inputs of type A . The inputs are provided by first eliminating an assumption $x : \Box_2 A$ into a graded assumption $y : [A]_2$ which is then used twice to apply f :

$$\begin{array}{c}
\frac{\dots}{f : A \multimap A \multimap B, y : A \vdash_L f y : A \multimap B} \text{APP}_L \quad \frac{\dots}{y : A \vdash_L y : A} \text{VAR}_L \\
\frac{f : A \multimap A \multimap B, y : [A]_1 \vdash_L f y : A \multimap B}{f : A \multimap A \multimap B, y : [A]_1 \vdash_L y : A} \text{DER}_L \quad \frac{y : [A]_1 \vdash_L y : A}{y : [A]_1 \vdash_L y : A} \text{APP}_L \\
\frac{\dots \quad f : A \multimap A \multimap B, y : [A]_2 \vdash_L f y y : B}{f : A \multimap A \multimap B, x : \Box_2 A \vdash_L \text{let } [y] = x \text{ in } f y y : B} \text{LET}_L
\end{array}$$

Example 2.5 (None-one-tons [Atkey 2018; McBride 2016], a.k.a. the Linearity Semiring). Linearity can be captured with the semiring $\mathcal{R} = \{0, 1, \omega\}$, with the required laws for 0 and 1, and $1 + 1 = \omega$ and $\omega + r = \omega = r + \omega$ and $\omega \cdot \omega = \omega$, and ordering $0 \sqsubseteq \omega$ and $1 \sqsubseteq \omega$. This captures linear reasoning in a graded modal setting [\[Abel and Bernardy 2020; Abel et al. 2023\]](#), with 1 denoting linearity and ω for non-linearity, and 0 providing a slightly finer-grained view when non-linearly discarding.

Example 2.6 (Security [Abel and Bernardy 2020; Gaboardi et al. 2016; Liepelt et al. 2026b]). Consider a semi-lattice of security levels $\mathcal{L} = \{\text{Lo}, \text{Hi}\}$ with ordering such that $\text{Hi} \geq \text{Lo}$. This forms a semiring with multiplication as join $\cdot = \vee$ with unit $1 = \text{Lo}$, and addition as meet $+$ with unit $0 = \text{Hi}$. We can type a function which maps low-security to high-security values, using approximation where we set $\sqsubseteq = \geq$:

$$\begin{array}{c}
\frac{}{x : A \vdash_L x : A} \text{VAR}_L \\
\frac{x : A \vdash_L x : A}{x : [A]_{\text{Lo}} \vdash_L x : A} \text{DER}_L \\
\frac{x : [A]_{\text{Hi}} \vdash_L [x] : \Box_{\text{Hi}} A \quad \text{Hi} \geq \text{Lo}}{x : [A]_{\text{Lo}} \vdash_L [x] : \Box_{\text{Hi}} A} \text{APPROX}_L
\end{array}$$

A function mapping from Hi to Lo values is derivable but cannot use its input. This *non-interference* property is proved by [Abel and Bernardy \[2020\]](#) and [Liepelt et al. \[2026b\]](#).

$t \rightsquigarrow_L t$ Operational semantics (Call-By-Name)

$$\frac{}{(\lambda x. t_2) t_1 \rightsquigarrow_L [t_1/x] t_2} \beta_L \quad \frac{t_1 \rightsquigarrow_L t'_1}{t_1 t_2 \rightsquigarrow_L t'_1 t_2} \text{CONGAPP}_L$$

$$\frac{}{\text{let } [x] = [t_1] \text{ in } t_2 \rightsquigarrow_L [t_1/x] t_2} \square\beta_L \quad \frac{t_1 \rightsquigarrow_L t'_1}{\text{let } [x] = t_1 \text{ in } t_2 \rightsquigarrow_L \text{let } [x] = t'_1 \text{ in } t_2} \text{CONGLETL}_L$$

$t \equiv_L t'$ Equational theory (congruence rules omitted).

$$\frac{}{(\lambda x. t_2) t_1 \equiv_L [t_1/x] t_2} \equiv_{L-\beta} \quad \frac{x \# t}{\lambda x. t x \equiv_L t} \equiv_{L-\eta} \quad \frac{}{\text{let } [x] = [t_1] \text{ in } t_2 \equiv_L [t_1/x] t_2} \equiv_{L-\square\beta}$$

$$\frac{}{\text{let } [x] = t \text{ in } [x] \equiv_L t} \equiv_{L-\square\eta} \quad \frac{}{\text{let } [x] = [t_1] \text{ in } [t_2] \equiv_L [\text{let } [x] = [t_1] \text{ in } t_2]} \equiv_{L-\text{let}\square}$$

$$\frac{x \# t_0}{t_0 (\text{let } [x] = t_1 \text{ in } t_2) \equiv_L \text{let } [x] = t_1 \text{ in } (t_2 t_0)} \equiv_{L-\text{let}\eta} \quad \frac{x \# t_0}{(\text{let } [x] = t_1 \text{ in } t_2) t_0 \equiv_L \text{let } [x] = t_1 \text{ in } (t_2 t_0)} \equiv_{L-\text{let}\eta}$$

Fig. 3. Linear Base - Operational semantics and equations

Example 2.7 (Product). Given two pre-ordered semirings $\mathcal{R}$ and $\mathcal{S}$, their product $\mathcal{R} \times \mathcal{S}$ is a semiring, with zero element $\langle 0, 0 \rangle \in (\mathcal{R} \times \mathcal{S})$, one element $\langle 1, 1 \rangle \in (\mathcal{R} \times \mathcal{S})$, and operations and ordering acting pointwise. This is a standard algebraic construction which has been used in graded systems to combine analyses (e.g. [Orchard et al. 2019]).

Section 3.2.2 uses this construction to compose the linearity semiring (Example 2.5) with some other given semiring, thereby making explicit the linearity analysis inherent to Linear Base.

Bianchini et al. [2023b] present the *smash product* construction as a variation on the product construction which has some preconditions and filters out the elements of the product that have either the left or the right = 0, instead providing a single canonical 0 element.

LEMMA 2.8 (ADMISSIBILITY OF SUBSTITUTION). *There are two admissible substitutions:*

- (1) (Linear) If $\Gamma_1 \vdash_L t_1 : A$ and $\Gamma_2, x : A \vdash_L t_2 : B$ then $\Gamma_1 + \Gamma_2 \vdash_L [t_1/x] t_2 : B$;
- (2) (Graded) If $\Gamma_1 \vdash_L t_1 : A$ and $\text{graded}(\Gamma_1)$ and $\Gamma_2, x : [A]_r \vdash_L t_2 : B$ then $r \cdot \Gamma_1 + \Gamma_2 \vdash_L [t_1/x] t_2 : B$.

Remark 2.9. In the literature (e.g. Brunel et al. [2014]; Gaboardi et al. [2016]; Orchard et al. [2019]), the graded context predicates (2.2) are often presented instead as partial identity functions on contexts: written $[\Gamma]$, the function is identity when the context contains only graded assumptions and is undefined otherwise, preventing a rule from being applied (and similarly $[\Gamma]_r$ for contexts containing only assumptions graded by r). We chose to make the predicates explicit for clarity.

Operational Semantics. Figure 3 (first group) gives the operational semantics, which is that of the call-by-name λ -calculus with an additional congruence rule for graded modal elimination and β -reduction of graded modal introduction followed by elimination.

Note that the operational semantics we consider for each calculus in this paper represents the standard models given in the literature in question, with progress and preservation theorems, but these do not track specific resource behaviour operationally (see discussion of resource-aware operational models in Section 6.6).

Appendix A [Liepelt et al. 2026a] gives the definition of syntactic substitution which is standard.

Equational theory. Figure 3 (second group) defines an equational theory, giving the key rules. For brevity, we omit the congruence rules which apply to all sub-term positions. Note that $\rightsquigarrow_L \subseteq \equiv_L$.

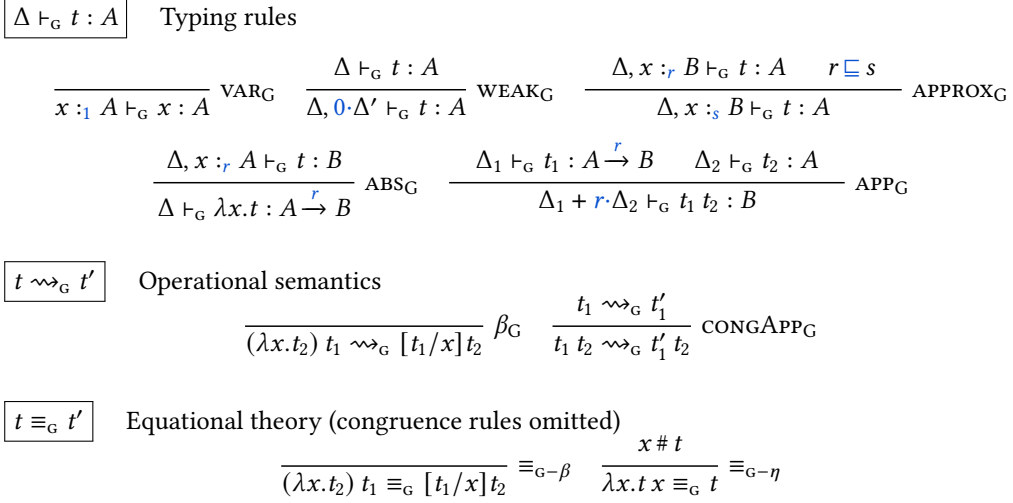


Fig. 4. Graded Base system

Definition 2.10 (Freshness). We make use of freshness here and throughout. It is defined as follows:

$$x_1, \dots, x_n \# t \triangleq \{x_1, \dots, x_n\} \cap \text{fv}(t) = \emptyset$$

2.2 Graded Base

The next calculus, Graded Base, is representative of the coeffect systems of [Petricek et al. \[2014, 2013\]](#), the generalised bounded linear type system of [Ghica and Smith \[2014\]](#), the simply-typed core of work by [Abel and Bernardy \[2020\]](#), [Choudhury et al. \[2021\]](#), and [Abel et al. \[2023\]](#), the OO-based approach of [Bianchini et al. \[2022\]](#), and others. The syntax of terms is that of the λ -calculus:

$$t ::= x \mid t_1 t_2 \mid \lambda x. t \quad (\text{terms})$$

We use t to also range over Graded Base terms but disambiguate which calculus terms belong to via the context of their later use (in translations). Type comprise graded functions and a base type K :

$$A ::= A \xrightarrow{r} B \mid K \quad (\text{types})$$

$$\Delta ::= \emptyset \mid \Delta, x :_r A \quad (\text{contexts})$$

Instead of a linear function type and graded modality, Graded Base has just a *graded function type*, where $r \in \mathcal{R}$. Contexts contain only graded assumptions which we instead denote as $x :_r A$.

Figure 4 (first group) gives the typing with judgments of the form $\Delta \vdash_G t : A$. Note the singleton context graded at 1 in the *variable* rule. The *abstraction* rule introduces its binding into the context graded at r , per the graded function arrow. In the *application* rule, the function arrow propagates the function's resource requirements onto its argument t_2 via scalar multiplication on Δ_2 (i.e., the context of t_2). Weakening scales by 0 all the grades in an additional context Δ' .

Graded substitution is admissible, akin to the second clause of Linear Base substitution (Lemma 2.8):

LEMMA 2.11 (ADMISSIBILITY OF SUBSTITUTION). *If $\Delta_1 \vdash_G t_1 : A$ and $\Delta_2, x :_r A \vdash_G t_2 : B$ then $r \cdot \Delta_1 + \Delta_2 \vdash_G [t_1/x] t_2 : B$.*

Example 2.12. The simply-typed λ -calculus is recovered by instantiating Graded Base with the single-point semiring $\mathcal{R} = \{\bullet\}$.

Example 2.13. The higher-order function $h = \lambda f. \lambda x. f(f\ x)$ mentioned in the introduction, composing a function with itself twice, has the following derivation for its body:

$$\frac{\frac{f :_1 (A \xrightarrow{r} A) \vdash_G f : A \xrightarrow{r} A}{f :_1 (A \xrightarrow{r} A) \vdash_G f : A \xrightarrow{r} A} \text{VAR}_G \quad \frac{\frac{f :_1 (A \xrightarrow{r} A) \vdash_G f : A \xrightarrow{r} A}{x :_1 A \vdash_G x : A} \text{VAR}_G}{f :_1 (A \xrightarrow{r} A), x :_{(r-r)} A \vdash_G f\ x : A} \text{APP}_G}{f :_{(1+1)} (A \xrightarrow{r} A), x :_{(r-r)} A \vdash_G f(f\ x) : A} \text{APP}_G$$

All the semirings mentioned in Section 2.1 (Linear Base) can also be used with Graded Base.

Operational Semantics and Equational theory. Figure 4 (second group) gives the operational semantics, which is that of the CBN λ -calculus. Appendix A [Liepelt et al. 2026a] gives the definition of syntactic substitution which is standard.

Figure 4 (third group) defines an equational theory for the calculus, giving the key rules but omitting the congruence rules which apply to all sub-term positions. Note $\rightsquigarrow_G \subseteq \equiv_G$.

2.3 Translation

In the course of the paper, we define translations between several calculi of the Linear and Graded strands, according to the overview in Figure 1. As for the term and type syntax of the different calculi, we use the same interpretation brackets across different variations of linear- and graded-base calculi respectively, making clear from the context which interpretation we are referring to.

The interpretation $\llbracket - \rrbracket$ takes graded-base terms to linear-base terms and $\langle\!\langle - \rangle\!\rangle$ takes linear-base terms to graded-base terms. The following mnemonics have proven useful to the authors:

- Given a graded term t , $\llbracket t \rrbracket$ is its translation, into a Linear term, with the square Lilac brackets mirroring the shape of the letter L.
- Given a linear term t , $\langle\!\langle t \rangle\!\rangle$ is its translation into a Graded term, with the rounded Green parentheses mirroring the shape of the letter G.

The same notation is used for type and context translations.

2.3.1 Graded Base to Linear Base translation.

$$\begin{aligned} \llbracket x \rrbracket &\triangleq x \\ \llbracket \lambda x. t \rrbracket &\triangleq \lambda y. \text{let } \llbracket x \rrbracket = y \text{ in } \llbracket t \rrbracket & y \# t \\ \llbracket t_1\ t_2 \rrbracket &\triangleq \llbracket t_1 \rrbracket\ \llbracket t_2 \rrbracket & \text{(Term translation)} \\ \llbracket K \rrbracket &\triangleq K \\ \llbracket A \xrightarrow{r} B \rrbracket &\triangleq \llbracket A \rrbracket \multimap \llbracket B \rrbracket & \text{(Type translation)} \\ \langle\!\langle \emptyset \rangle\!\rangle &\triangleq \emptyset \\ \langle\!\langle \Delta, x :_r A \rangle\!\rangle &\triangleq \langle\!\langle \Delta \rangle\!\rangle, x : \langle\!\langle A \rangle\!\rangle_r & \text{(Context translation)} \end{aligned}$$

The key idea is to use the graded modality to capture the grading of assumptions and functions. The graded function type translates to the Linear Base function type $\multimap$, with its left-hand side under $\llbracket \cdot \rrbracket$. This translation leads to the insight that Graded Base effectively wraps a graded modality around function inputs, capturing the usage of the parameter in the body. Since every term in Graded Base has an associated grading in the context, every abstraction in the translation from Graded to Linear needs to do an unboxing (let) and every application needs to do a boxing (promotion).

Contexts are translated straightforwardly. The translation can never produce a linear assumption.

Example 2.14. The judgment $f :_{(1+1)} (A \xrightarrow{r} A), x :_{(r,r)} A \vdash_G f(f x) : A$ in Graded Base translates to $f : [\Box_r[A]] \multimap [A]]_{(1+1)}, x : [[A]]_{(r,r)} \vdash_L f[f[x]] : [A]$ in Linear Base. Note the right-hand sides of an application find themselves under a promotion.

Remark 2.15. We do not translate $A \xrightarrow{1} B$ to a linear function $[A] \multimap [B]$; this is because $A \xrightarrow{1} B$ does not necessarily mean that the function uses its argument linearly. Some choices of semiring mean that 1 does not mean ‘used exactly once’, the simplest example being the single-point (trivial) semiring in which $0 = 1$ and so 1 would mean any usage.

We now establish key soundness results. Firstly, that the translation is type preserving, i.e., translation of terms, contexts, and types is such that a well-typed derivation in Graded Base is well-typed under its translation to Linear Base. This requires various small lemmas that the translation is a homomorphism between graded-base contexts and linear-base contexts which we elide here. Secondly, the translation preserves the operational semantics, and lastly it preserves the equational theory. Appendix B.1 [Liepelt et al. 2026a] provides the full proofs and supporting lemmas.

Theorem 1. [Soundness of the Graded Base to Linear Base translation]

- Type preservation: $\Delta \vdash_G t : A \implies [[\Delta]] \vdash_L [[t]] : [[A]]$
- Operational correspondence: $t \rightsquigarrow_G t' \implies [[t]] \rightsquigarrow_L^* [[t']]$
- Equation preservation: $t \equiv_G t' \implies [[t]] \equiv_L [[t']]$

2.3.2 Linear Base to Graded Base translation. A type-preserving translation from Linear Base to Graded Base is however more difficult since we need to find some encoding of the graded modal type $(\Box_r A)$. From the introduction rule of the graded modality (promotion), such a translation on terms should incur a multiplication of the context by r which in Graded Base is only provided through application of a $A \xrightarrow{r} B$ function. Thus a translation must have such a function in a negative position, suggesting a continuation-passing style (CPS) or Church-encoding-like form.

Taking inspiration from Lindley [2025] on encoding products, we extend Plotkin’s CPS translation of the call-by-name λ -calculus [Plotkin 1975] with a CPS encoding of the graded modalities of Linear Base into Graded Base:

$$\begin{array}{ll}
 \langle A \multimap B \rangle & \triangleq ((\langle A \rangle \xrightarrow{1} \langle B \rangle) \xrightarrow{1} K) \xrightarrow{1} K \\
 \langle \Box_r A \rangle & \triangleq (\langle A \rangle \xrightarrow{r} K) \xrightarrow{1} K \\
 \langle K \rangle & \triangleq (K \xrightarrow{1} K) \xrightarrow{1} K & \text{(Type translation)} \\
 \langle x \rangle & \triangleq \lambda k. x k \\
 \langle \lambda x. t \rangle & \triangleq \lambda k. k (\lambda x. \langle t \rangle) \\
 \langle t_1 t_2 \rangle & \triangleq \lambda k. (\langle t_1 \rangle (\lambda f. f \langle t_2 \rangle k)) \\
 \langle [t] \rangle & \triangleq \lambda k. k \langle t \rangle \\
 \langle \text{let } [x] = t_1 \text{ in } t_2 \rangle & \triangleq \lambda k. (\langle t_1 \rangle (\lambda x. \langle t_2 \rangle k)) & \text{(Term translation)} \\
 \langle \emptyset \rangle & \triangleq \emptyset \\
 \langle \Gamma, x : A \rangle & \triangleq \langle \Gamma \rangle, x :_1 \langle A \rangle \\
 \langle \Gamma, x : [\Box_r A]_r \rangle & \triangleq \langle \Gamma \rangle, x :_r \langle A \rangle & \text{(Context translation)}
 \end{array}$$

The key point is to (recursively) encode $\Box_r A$ as $(A \xrightarrow{r} K) \xrightarrow{1} K$ so that in the case of promotion $[t]$ we can use application to scale the grades of t . Note that linear assumptions get mapped to graded assumptions with grade 1, though this may not mean ‘linear use’ (see Section 3.2.2 for a solution).

Theorem 2. [Soundness of the Linear Base to Graded Base translation (proofs in Appendix B.2 [Liepelt et al. 2026a]).]

- Type preservation: $\Gamma \vdash_L t : A \implies \langle \Gamma \rangle \vdash_G \langle t \rangle : \langle A \rangle$
- Operational correspondence: $t \rightsquigarrow_L t' \implies \langle t \rangle \rightsquigarrow_{\beta_G}^* \langle t' \rangle$
- Equation preservation: $t \equiv_L^{\beta \rightarrow \beta \eta \square} t' \implies \langle t \rangle \equiv_G \langle t' \rangle$

As we use Plotkin’s CBN CPS translation, there are two notable implications: Firstly, regarding operational correspondence, our source uses CBN but we need full β -reduction (via the CONGABS_G rule) in the target to prove simulation. Computational adequacy then follows from the λ -calculus standardisation theorem. Secondly, regarding equation preservation: we preserve only β -equality for functions and not η -equality—a well-known limitation of Plotkin’s translation. However, we preserve both β - and η -equality for the graded modality, hence we label the equational theory here as $\equiv_L^{\beta \rightarrow \beta \eta \square}$. Section 4 considers an alternate encoding in a polymorphic Graded Base setting.

3 Extending Graded Base with a Graded Modality

We now add a graded modality to Graded Base thereby arriving at **Graded Modal Base**. This captures the type system features of calculi like GRAD [Choudhury et al. 2021] and a monomorphic subset of Abel and Bernardy [2020], and permits a simple translation to Linear Base.

3.1 Graded Modal Base

Graded Modal Base can now share the syntax (and operational semantics) of Linear Base:

$$t ::= x \mid t_1 \ t_2 \mid \lambda x. t \mid [t] \mid \text{let } [x] = t_1 \text{ in } t_2 \quad (\text{shared terms})$$

$$A ::= A \multimap_r B \mid \square_r A \mid K \quad (\text{extended types})$$

Typing judgments in Graded Modal Base are of the form $\Delta \vdash_{G_\square} t : A$. The typing rules are the same as Graded Base with addition of typing for graded modal introduction and elimination:

$$\frac{\Delta \vdash_{G_\square} t : A}{r \cdot \Delta \vdash_{G_\square} [t] : \square_r A} \text{PR}_G \quad \frac{\Delta_1 \vdash_{G_\square} t_1 : \square_r A \quad \Delta_2, x :_r A \vdash_{G_\square} t_2 : B}{\Delta_1 + \Delta_2 \vdash_{G_\square} \text{let } [x] = t_1 \text{ in } t_2 : B} \text{LET}_G$$

The equational theory extends to include $\beta\eta$ -equality on graded modalities (see Appendix A.5 [Liepelt et al. 2026a]).

3.2 Translation

3.2.1 Linear Base to Graded Modal Base translation. We can now translate Linear Base to Graded Modal Base as a homomorphism on terms, rather than the CPS encoding of Section 2.3:

$$\begin{aligned} \langle x \rangle &\triangleq x \\ \langle \lambda x. t \rangle &\triangleq \lambda x. \langle t \rangle \\ \langle t_1 \ t_2 \rangle &\triangleq \langle t_1 \rangle \ \langle t_2 \rangle \\ \langle [t] \rangle &\triangleq [\langle t \rangle] \\ \langle \text{let } [x] = t_1 \text{ in } t_2 \rangle &\triangleq \text{let } [x] = \langle t_1 \rangle \text{ in } \langle t_2 \rangle \end{aligned} \quad (\text{Term translation})$$

$$\begin{aligned} \langle A \multimap B \rangle &\triangleq \langle A \rangle \xrightarrow{1} \langle B \rangle \\ \langle \square_r A \rangle &\triangleq \square_r \langle A \rangle \\ \langle K \rangle &\triangleq K \end{aligned} \quad (\text{Type translation})$$

$$\begin{aligned} \langle \emptyset \rangle &\triangleq \emptyset \\ \langle \Gamma, x : A \rangle &\triangleq \langle \Gamma \rangle, x :_1 \langle A \rangle \\ \langle \Gamma, x : [A]_r \rangle &\triangleq \langle \Gamma \rangle, x :_r \langle A \rangle \end{aligned} \quad (\text{Context translation})$$

For types, we translate $\Box_r$ in the straightforward way, directly to the Graded Modal Base graded modality. To translate the linear function space we grade the resulting function arrow in Graded Modal Base at $1 \in \mathcal{R}$ and similarly for linear assumptions.

Example 3.1. The following Linear Base judgment types a K -combinator-like expression:

$$x : B \vdash_L \lambda y. \text{let } [z] = y \text{ in } x : \Box_0 A \multimap B$$

The parameter y is used by eliminating the graded modality, binding to z but then discarding the term. Its translation to Graded Modal Base results in an identical term but with translated types:

$$x :_1 \langle B \rangle \vdash_G \lambda y. \text{let } [z] = y \text{ in } x : \Box_0 \langle A \rangle \xrightarrow{1} \langle B \rangle$$

Theorem 3. [Soundness of the Linear Base to Graded Modal Base translation]

- Type preservation: $\Gamma \vdash_L t : A \implies \langle \Gamma \rangle \vdash_{G\Box} \langle t \rangle : \langle A \rangle$
- Operational correspondence: $t \rightsquigarrow_L t' \implies \langle t \rangle \rightsquigarrow_{G\Box}^* \langle t' \rangle$
- Equation preservation: $t \equiv_L t' \implies \langle t \rangle \equiv_{G\Box} \langle t' \rangle$

Appendix B.3 [Liepelt et al. 2026a] provides the proof along with its supporting lemmas.

Remark 3.2. We might consider whether there is a dual to the above, a kind of completeness result, i.e., that given $\Delta \vdash_G t : A$ there exists Γ, t', A' such that $\Gamma \vdash_L t' : A'$ and $\langle \Gamma \rangle \vdash_G \langle t' \rangle : \langle A' \rangle$ and $\Delta \equiv \langle \Gamma \rangle$ and $t \equiv \langle t' \rangle$ and $A \equiv \langle A' \rangle$. However, an easy counterexample is the following judgment:

$$x :_1 B \vdash_G \lambda y. x : A \xrightarrow{0} B.$$

There is no type in Linear Base which translates to $A \xrightarrow{0} B$ as we always translate the linear function type to $A \xrightarrow{1} B$. Indeed, the analogous term in Linear Base was already shown in Example 3.1, where we see that the result of the translation has type $\Box_0 \langle A \rangle \xrightarrow{1} \langle B \rangle$.

Thus, the translation from Linear Base to Graded Modal Base is not surjective. Neither is it injective: both the linear assumption $x : A$ and the graded assumption $x : [A]_1$ are translated to $x :_1 \langle A \rangle$. For those semirings in which 1 does not capture linearity (e.g., the singleton semiring, any semiring where $0 \leq 1$, and others), linear and non-linear use is conflated by this translation.

3.2.2 A Linearity-Preserving Linear Base to Graded Modal Base Translation. Instead, we can give an alternate translation which is injective and *linearity preserving*: keeping the translation of terms as above, we can translate types and grades from Linear Base with semiring $\mathcal{R}$ into types and grades in Graded Modal Base with semiring $(\mathcal{R} \times \{0, 1, \omega\})$, i.e., the product semiring (Example 2.7) of $\mathcal{R}$ with the linearity-capturing semiring (Example 2.5).

$$\begin{aligned} \langle A \multimap B \rangle' &\triangleq \langle A \rangle' \xrightarrow{\langle 1, 1 \rangle} \langle B \rangle' \\ \langle \Box_r A \rangle' &\triangleq \Box_{\langle r, \omega \rangle} \langle A \rangle' \\ \langle K \rangle' &\triangleq K \end{aligned} \quad \text{(Type translation)}$$

$$\begin{aligned} \langle \Gamma, x : A \rangle' &\triangleq \langle \Gamma \rangle', x : \langle 1, 1 \rangle \langle A \rangle' \\ \langle \Gamma, x : [A]_r \rangle' &\triangleq \langle \Gamma \rangle', x : \langle r, \omega \rangle \langle A \rangle' \end{aligned} \quad \text{(Context translation)}$$

Thus, the translation is now injective in the types since we translate a linear assumption $x : A$ to have grade $\langle 1, 1 \rangle$ but a graded assumption $x : [A]_r$ to $\langle r, \omega \rangle$ which is a different grade even if $r = 1$. By taking the pairing with the linearity semiring, we can capture the combination of linearity and grading that occurs in Linear Base, but now in the Graded Modal Base setting.

Operational correspondence and equation preservation hold as before in Theorem 3 since the term translation is unchanged. We then re-establish type preservation of the translation:

Theorem 4. [Soundness of the Linearity-Preserving Linear Base to Graded Modal Base translation]

- Type preservation: $\Gamma \vdash_L t : A \implies (\Gamma)' \vdash_{G_\square} ([t]) : ([A])'$

Appendix B.4 [Liepelt et al. 2026a] provides the proof.

The key insight in this translation is that Linear Base essentially comprises two parallel forms of analysis: a graded analysis and a latent linear analysis witnessed by the corresponding two forms of typing assumption in Linear Base contexts. Our translation here captures these two aspects systematically, by taking the product of the same grading semiring and the linearity semiring.

A question may arise here: *What is the meaning of a grade such as $\langle 2, 1 \rangle$, where the first component looks non-linear in $\mathcal{R}$ but the second represents linearity?* Such a grade is not in the image of our translation but we may still consider what it is like to type terms in Graded (Modal) Base with $\mathcal{R} \times \{0, 1, \omega\}$. First, let us decompose $\langle 2, 1 \rangle$ into $\langle 1 + 1, 1 \rangle$ for clarity, i.e., for any semiring $\mathcal{R}$ in the first component. Such grades could be formed via approximation if $\mathcal{R}$ permits $1 \sqsubseteq 1 + 1$ and thus $\langle 1, 1 \rangle \sqsubseteq \langle 1 + 1, 1 \rangle$ is an over-approximation of grades, e.g., useful for when calculating the grade across multiple program branches. Without such an approximation, $\langle 1 + 1, 1 \rangle$ cannot be derived from variable use and contraction of variables alone since in $(\mathcal{R} \times \{0, 1, \omega\})$ we have $\langle 1, 1 \rangle + \langle 1, 1 \rangle = \langle 1 + 1, \omega \rangle$. Such a grade could only arise from an open term applying a function, e.g.,

$$f :_1 A \xrightarrow{\langle 1+1, 1 \rangle} B, x :_{\langle 1+1, 1 \rangle} A \vdash_G f x : B.$$

However there is no well-typed closed term that can be formed to substitute into f since we cannot form $\langle 1 + 1, 1 \rangle$ without use of such an open term. Consequently, for closed terms, we cannot form $\langle 1 + 1, 1 \rangle$ unless it is permitted by approximation in $\mathcal{R}$. In the approximation case, it is up to the choice (or design) of semiring to permit approximation via the definition of $\sqsubseteq$.

More generally, we may consider the advantage of pairing semirings with the linearity semiring. Consider a semiring such as the security lattice semiring (Example 2.6) where $1 = \text{Lo}$. Since $\text{Lo} + \text{Lo} = \text{Lo}$ and $0 \sqsubseteq \text{Lo}$ then a Lo-graded variable can represent both linear or non-linear usage. The pairing $\langle \text{Lo}, 1 \rangle$ then distinguishes a linear low-security variable from $\langle \text{Lo}, \omega \rangle$ which instead represents a non-linear low-security variable. Conversely, recall that $0 = \text{Hi}$, giving the translation $([\square_{\text{Hi}} A])' \equiv \square_{\langle \text{Hi}, \omega \rangle} A$, which allows unlimited usage, but limited to Hi security contexts.

Our approach here is not necessarily the only sound and injective translation; however, it seems particularly illuminating, in that the product construction clearly demarcates and represents the two analyses present within Linear Base: the explicit coeffect in the grades on the one hand and the latent linearity coeffect of the system (not present in Graded Base) on the other.

3.2.3 Graded Modal Base to Linear Base Translation. We can now extend the previous translation from Graded Base to Linear Base (Section 2.3) to translate from Graded Modal Base to Linear Base:

$$\begin{array}{ll} \llbracket x \rrbracket & \triangleq x \\ \llbracket \lambda x. t \rrbracket & \triangleq \lambda y. \text{let } [x] = y \text{ in } \llbracket t \rrbracket \quad y \# t \\ \llbracket t_1 t_2 \rrbracket & \triangleq \llbracket t_1 \rrbracket \llbracket t_2 \rrbracket \\ \llbracket [t] \rrbracket & \triangleq \llbracket \llbracket t \rrbracket \rrbracket \\ \llbracket \text{let } [x] = t_1 \text{ in } t_2 \rrbracket & \triangleq \text{let } [x] = \llbracket t_1 \rrbracket \text{ in } \llbracket t_2 \rrbracket \end{array} \quad (\text{Term translation})$$

$$\begin{array}{ll} \llbracket A \xrightarrow{r} B \rrbracket & \triangleq \square_r \llbracket A \rrbracket \multimap \llbracket B \rrbracket \\ \llbracket \square_r A \rrbracket & \triangleq \square_r \llbracket A \rrbracket \\ \llbracket K \rrbracket & \triangleq K \end{array} \quad (\text{Type translation})$$

$$\begin{array}{ll} \llbracket \emptyset \rrbracket & \triangleq \emptyset \\ \llbracket \Delta, x :_r A \rrbracket & \triangleq \llbracket \Delta \rrbracket, x : \llbracket [A] \rrbracket_r \end{array} \quad (\text{Context translation})$$

The only new parts are the translation of the graded modality which maps directly onto the graded modality in Linear Base. We then (re)establish soundness. Appendix B.5 [Liepelt et al. 2026a] provides the proofs.

Theorem 5. [Soundness of the Graded Modal Base to Linear Base translation]

- Type preservation: $\Delta \vdash_{\text{G}\Box} t : A \implies \llbracket \Delta \rrbracket \vdash_{\text{L}} \llbracket t \rrbracket : \llbracket A \rrbracket$
- Operational correspondence: $t \rightsquigarrow_{\text{G}\Box} t' \implies \llbracket t \rrbracket \rightsquigarrow_{\text{L}}^* \llbracket t' \rrbracket$
- Equation preservation: $t \equiv_{\text{G}\Box} t' \implies \llbracket t \rrbracket \equiv_{\text{L}} \llbracket t' \rrbracket$.

3.2.4 Mutual relationship. Given the pair of translations here, we might wonder: what is the relationship between the translations? Are they mutually inverse? By considering just the translation of function types we can quickly see that the translations are not inverse:

$$\begin{aligned} \llbracket (A \multimap B) \rrbracket &= \llbracket A \xrightarrow{1} B \rrbracket \equiv \Box_1 \llbracket (A) \rrbracket \multimap \llbracket (B) \rrbracket \\ \llbracket (A \xrightarrow{r} B) \rrbracket &= \llbracket \Box_r A \multimap B \rrbracket \equiv \Box_r \llbracket (A) \rrbracket \xrightarrow{1} \llbracket (B) \rrbracket \end{aligned}$$

Similarly, given $x : A \vdash_{\text{L}} x : A$ in Linear Base then its translation to Graded Modal Base is $x : \Box_1 A \vdash_{\text{G}\Box} x : A$ which is translated back to Linear Base as $x : \llbracket A \rrbracket_1 \vdash_{\text{L}} x : A$.

However, by composing operational correspondence theorems, then round-tripping the translations yields a term that corresponds operationally to the original, in both directions:

COROLLARY 3.3 (OBSERVATIONAL ISOMORPHISM). *If $t \rightsquigarrow_{\text{L}} t'$ then $\llbracket (t) \rrbracket \rightsquigarrow_{\text{L}}^* \llbracket (t') \rrbracket$ and if $t \rightsquigarrow_{\text{G}\Box} t'$ then $\llbracket (t) \rrbracket \rightsquigarrow_{\text{G}\Box}^* \llbracket (t') \rrbracket$.*

We might also wonder if there is an adjoint relationship between the translations. If $\llbracket - \rrbracket \dashv \langle - \rangle$ then we would expect to be able to construct a mapping in the category of types (whose morphisms are derivations of judgments with source as inputs and the target as the result):

$$\eta_A : A \rightarrow \llbracket (A) \rrbracket$$

However, there is no such mapping for $A \multimap B$ to $\llbracket (A \multimap B) \rrbracket \equiv \Box_1 \llbracket (A) \rrbracket \multimap \llbracket (B) \rrbracket$ in general since this is undefined for a higher-order type.

Consider $(K \multimap K) \multimap K$ for the base type K whose translation between the systems is the identity, and thus $\llbracket (K) \rrbracket = K$. Then η would need to provide a derivation of the judgment:

$$(K \multimap K) \multimap K \vdash \Box_1(\Box_1 K \multimap K) \multimap K$$

Such a derivation would require us to map a value of type $\Box_1(\Box_1 K \multimap K)$ to one of type $K \multimap K$. We can certainly derive $\Box_1(\Box_1 K \multimap K) \vdash \Box_1 K \multimap K$ by elimination and dereliction, but $\Box_1 K \multimap K \not\vdash K \multimap K$. To see why, consider the single-point semiring $0 = 1$ and thus a function typed $\Box_1 K \multimap K$ may not use its argument at all, whereas $K \multimap K$ must use its argument exactly once.

Given the above negative result, we could then consider the opposite adjoint relationship, i.e., perhaps $\langle - \rangle \dashv \llbracket - \rrbracket$? In which case, we would require the counit operation:

$$\epsilon_A : \llbracket (A) \rrbracket \rightarrow A$$

However, there is no such mapping, even in a first-order case: for $\llbracket (K \multimap K) \rrbracket \equiv \Box_1 \llbracket (K) \rrbracket \multimap \llbracket (K) \rrbracket$ we need to map to $K \multimap K$ but as described above, no such derivation is possible.

Thus the translations are neither inverses nor adjoint. We leave it as future work to consider whether more complex invertible translations are possible.

3.3 Generalised $\Box$ -elimination

Elimination of graded modalities in both linear- and graded-base systems looks almost identical, eliminating a graded modality at grade r by a kind of ‘cut’ (substitution) against a variable with matching grade r :

$$\frac{\Gamma_1 \vdash_L t_1 : \Box_r A \quad \Gamma_2, x : [A]_r \vdash_L t_2 : B}{\Gamma_1 + \Gamma_2 \vdash_L \text{let } [x] = t_1 \text{ in } t_2 : B} \text{LET}_L \quad \frac{\Delta_1 \vdash_{G_\Box} t_1 : \Box_r A \quad \Delta_2, x :_r A \vdash_{G_\Box} t_2 : B}{\Delta_1 + \Delta_2 \vdash_{G_\Box} \text{let } [x] = t_1 \text{ in } t_2 : B} \text{LET}_G$$

Some graded-base systems in the literature however have a more general elimination rule [Abel and Bernardy 2020; Moon et al. 2021], of the form:

$$\frac{\Delta_1 \vdash_{G_\Box} t_1 : \Box_r A \quad \Delta_2, x :_{(s \cdot r)} A \vdash_{G_\Box} t_2 : B}{(s \cdot \Delta_1) + \Delta_2 \vdash_{G_\Box} \text{let } [x] = t_1 \text{ in } t_2 : B} \text{LET}'_G$$

The generalised rule allows an additional scaling of the resources for t_1 due to additional use of x in t_2 . Such a rule has not appeared in any Linear Base system, but could easily be added to both systems to provide equivalent power across both flavours of coefficient types.

This generalised typing enables the derivation of a function $\Box_r(\Box_s A) \multimap \Box_{(r \cdot s)} A$, akin to the multiplication (join) of a graded monad (the dual to comultiplication of a graded comonad structure):

$$\frac{\frac{x :_1 \Box_r(\Box_s A) \vdash_{G_\Box} x : \Box_r(\Box_s A)}{\text{VAR}_G} \quad \frac{\frac{y :_1 \Box_s A \vdash_{G_\Box} y : \Box_s A}{\text{VAR}_G} \quad \frac{z :_1 A \vdash_{G_\Box} z : A}{\text{VAR}_G} \quad \frac{z :_{r \cdot s} A \vdash_{G_\Box} [z] : \Box_{r \cdot s} A}{\text{PRG}}}{\frac{y :_r \Box_s A \vdash_{G_\Box} \text{let } [z] = y \text{ in } [z] : \Box_{r \cdot s} A}{\text{LET}'_G}} \text{LET}'_G$$

Indeed, the generalised $\Box$ -elimination can be modelled in a categorical semantics by a graded comonad along with an additional graded multiplication operation $\mu_{r,s,A} : \Box_r(\Box_s A) \multimap \Box_{(r \cdot s)} A$ (as one might find in a graded monad structure). However, not all graded comonads modelling $\Box_r A$ admit such an operation and so having the generalised $\Box$ -elimination may not be desirable in general graded comonadic calculi. We thus omitted this generalised rule from the main part of our results, considering the simpler LET_L and LET_G rules.

4 Deriving a Graded Modality with a Polymorphic Graded Base

Bernardy et al. [2018] develop an extension to GHC/Haskell that provides a notion of linear typing by using a graded type system. The core calculus is akin to that of Graded Base, but specialised to the $\{0, 1, \omega\}$ semiring (Example 2.5). Their calculus has no graded modality, but the implementation as part of GHC/Haskell can encode the *unrestricted* modality Ur via an algebraic datatype:¹

```
1 data Ur a where
2   Ur :: a → Ur a
```

where the arrow $\rightarrow$ denotes an ω -graded arrow for the two-point semiring $\{1, \omega\}$ (0 is not available in the implementation) and thus this constructor is essentially of type $A \xrightarrow{\omega} \text{Ur } A$ in the Graded Base syntax. Hughes et al. [2020] give a generalisation of this definition to a graded modality:

```
1 data Box r a where
2   Box :: a %r → Box r a
```

¹<https://hackage.haskell.org/package/linear-base-0.4.0/docs/Data-Unrestricted-Linear.html#t:Ur>

This suggests an alternate Graded Base calculus in relation to Linear Base. We previously showed (Section 2.3.2) that a type-preserving translation of Linear Base terms into Graded Base requires a CPS encoding. By changing the substrate of Graded Base to System F, we have **Graded Poly Base**, unlocking the ability for a direct-style polymorphic Church encoding of this Box data type:

$$\Box_r A = \forall \beta. (A \xrightarrow{r} \beta) \xrightarrow{1} \beta \quad (1)$$

This enables a less intrusive translation, without the need to CPS translate all terms.

4.1 Graded Poly Base

We thus extend Graded Base with polymorphic typing, the syntax and typing of which is standard:

$$t ::= \dots \mid t@A \mid \Lambda \alpha. t \quad (\text{Terms})$$

$$A ::= \dots \mid \alpha \mid \forall \alpha. B \quad (\text{Types})$$

where α, β range over type variables, with a corresponding standard substitution at the type-level written $[A/\alpha]B$. Typing judgments are of the form $\Delta \vdash_{\text{GV}} t : A$ extending the previous typing with:

$$\frac{\Delta \vdash_{\text{GV}} t : A \quad \alpha \# t}{\Delta \vdash_{\text{GV}} \Lambda \alpha. t : \forall \alpha. A} \text{TYABS}_G \quad \frac{\Delta \vdash_{\text{GV}} t : \forall \alpha. A}{\Delta \vdash_{\text{GV}} t@B : [B/\alpha]A} \text{TYAPP}_G$$

The operational semantics has the standard β rule for type abstraction/application and congruence:

$$\frac{}{(\Lambda \alpha. t)@B \rightsquigarrow_{\text{GV}} [B/\alpha]t} \beta \quad \frac{t \rightsquigarrow_{\text{GV}} t'}{t@A \rightsquigarrow_{\text{GV}} t'@A} \text{CONG}$$

Introduction and elimination for the derived $\Box_r$ (from (1)) can therefore be derived respectively as follows (with their typing derivations shown):

$$\frac{\frac{\frac{\Delta \vdash_{\text{GV}} t : A \quad f :_1 A \xrightarrow{r} \beta' \vdash_{\text{GV}} f : A \xrightarrow{r} \beta'}{\Delta \vdash_{\text{GV}} f :_1 A \xrightarrow{r} \beta' \vdash_{\text{GV}} f t : \beta'} \text{APP}_G}{\Delta \vdash_{\text{GV}} \lambda f. f t : (A \xrightarrow{r} \beta') \xrightarrow{1} \beta'} \text{ABS}_G}{\Delta \vdash_{\text{GV}} \Lambda \beta. \lambda f. f t : \forall \beta. (A \xrightarrow{r} \beta) \xrightarrow{1} \beta} \text{TYABS}_G \quad (\Box_i)$$

$$\frac{\frac{\Delta_1 \vdash_{\text{GV}} t_1 : \forall \beta. (A \xrightarrow{r} \beta) \xrightarrow{1} \beta}{\Delta_1 \vdash_{\text{GV}} t_1@B : (A \xrightarrow{r} B) \xrightarrow{1} B} \text{TYAPP}_G \quad \frac{\Delta_2, x :_r A \vdash_{\text{GV}} t_2 : B}{\Delta_2 \vdash_{\text{GV}} \lambda x. t_2 : A \xrightarrow{r} B} \text{ABS}_G}{\Delta_1 + \Delta_2 \vdash_{\text{GV}} (t_1@B) (\lambda x. t_2) : B} \text{APP}_G \quad (\Box_e)$$

4.1.1 Linear Base to Graded Poly Base Translation. We thus adapt the translation of types to:

$$\llbracket \Box_r A \rrbracket \triangleq \forall \beta. (\llbracket A \rrbracket \xrightarrow{r} \beta) \xrightarrow{1} \beta \quad (\text{Type translation})$$

We might expect that we can interpret the syntax of terms then directly as in the above derivations, i.e., $\llbracket [t] \rrbracket \triangleq \Lambda \beta. \lambda f. f (\llbracket t \rrbracket)$ and $\llbracket \text{let } [x] = t_1 \text{ in } t_2 \rrbracket \triangleq (\llbracket t_1 \rrbracket @B) (\lambda x. \llbracket t_2 \rrbracket)$. However, note the presence of B on the right-hand side of $\Box$ -elimination, which has now appeared out of nowhere. Instead the translation must ingest the entire *typing derivation*, not merely the syntax of terms and types separately. We can however translate the types separately.

Thus, we define the translation as mapping from typing derivations to terms, where we denote derivation trees by Π :

$$\left(\frac{\frac{\Pi}{\Gamma \vdash_L t : A} \quad \text{graded}(\Gamma)}{r \cdot \Gamma \vdash_L [t] : \Box_r A} \text{PR}_L \right) \triangleq \Lambda \beta. \lambda f. f \langle \Pi \rangle \quad (\text{Term translation})$$

$$\left(\frac{\frac{\Pi_1}{\Gamma_1 \vdash_L t_1 : \Box_r A} \quad \frac{\Pi_2}{\Gamma_2, x : [A]_r \vdash_L t_2 : B}}{\Gamma_1 + \Gamma_2 \vdash_L \text{let } [x] = t_1 \text{ in } t_2 : B} \text{LET}_L \right) \triangleq ((\Pi_1) @ \langle B \rangle) (\lambda x. \langle \Pi_2 \rangle)$$

The rest of the translation follows the structure of the previous term translation.

In the following, we write $\Pi : \Gamma \vdash_L t : A$ to mean that Π is the derivation of a judgment $\Gamma \vdash_L t : A$.

Theorem 6. [Soundness of the Linear Base to Graded Poly Base translation]

- Type preservation: $\Pi : \Gamma \vdash_L t : A \implies \langle \Gamma \rangle \vdash_{G_V} \langle \Pi \rangle : \langle A \rangle$
- Operational correspondence: $(\Pi : \Gamma \vdash_L t : A) \wedge (\Pi' : \Gamma \vdash_L t' : A) \wedge (t \rightsquigarrow_L t') \implies \langle \Pi \rangle \rightsquigarrow_{G_V}^* \langle \Pi' \rangle$
- Equation preservation: $(\Pi : \Gamma \vdash_L t : A) \wedge (\Pi' : \Gamma \vdash_L t' : A) \wedge (t \equiv_L^{-\eta \square} t') \implies \langle \Pi \rangle \equiv_{G_V} \langle \Pi' \rangle$

Appendix B.6 [Liepelt et al. 2026a] provides the proofs.

Note the operational correspondence needs to have typing derivations of the source term and target term. Furthermore, the translation does not preserve η -equality for the graded modality; only the β -equality rules, commuting conversions, and congruence rules are preserved; we thus denote the subset of the preserved equational theory here as $\equiv_L^{-\eta \square}$. This is because the translation of a term t of type $\Box_r A$ may not be of the form $\Lambda \beta. \lambda f. f \ t'$ for some t' (e.g., it could be just a free variable), which is required for η -equality to apply. Instead, parametricity reasoning (based on the polymorphic typing) would provide that t is *observationally equivalent* to $\Lambda \beta. \lambda f. f \ t'$ for some t' , and thus the translation preserves η -equality up to observational equivalence, though not up to syntactic equality; we leave the details for further work.

Remark 4.1. (Ur is not the $!$ -modality) Bernardy et al. [2018] suggest Ur as an encoding of Linear Logic's $!$ -modality, although Hughes et al. [2021] later show that this encoding in Linear Haskell and similarly in Idris 2, as well as the graded modality in Granule, do not exactly correspond to Linear Logic's $!$ -modality. Once (multiplicative) products are included, Linear Logic does not admit distributing $!$ over products, i.e., $!(A \otimes B) \multimap !A \otimes !B$ is not derivable. However the aforementioned systems permit its derivation due to the interaction between graded modalities and products (and similarly sums). We consider calculi with products and sums next as this takes us closer towards practical programming and it allows us to uncover other expressivity differences in the literature.

5 Extending Linear Base and Graded Modal Base with Products and Sums

Thus far the calculi studied have only covered the implicational fragments of the two approaches, i.e., just function types and variables. Adding product and sum types to the calculi brings the theory closer to a useful basis for understanding actual programming languages such as the linear-types extension to Haskell (graded-base) or Granule (linear-base). We thus now add products and sums to Linear Base and Graded Modal Base following standard approaches in the literature for both flavours of linear-base and graded-base systems. This raises interesting questions that delineate the expressive power of linear- vs. graded-base calculi. From linear-base to graded-base, a straightforward homomorphic translation is possible. However, for the opposite direction we will see a difference in expressivity that needs handling by extending the linear-base system.

$t \rightsquigarrow_L t$ Standard operational semantics (Call-By-Name) for products, units, and sums

$$\begin{array}{c}
\frac{t_1 \rightsquigarrow_L t'_1}{\text{let } \langle x, y \rangle = t_1 \text{ in } t_2 \rightsquigarrow_L \text{let } \langle x, y \rangle = t'_1 \text{ in } t_2} \text{CONGP}_{\text{ROD}_L} \quad \frac{}{\text{let } \langle x, y \rangle = \langle t_1, t_2 \rangle \text{ in } t_3 \rightsquigarrow_L [t_1/x][t_2/y]t_3} \beta_L \\
\\
\frac{t_1 \rightsquigarrow_L t'_1}{\text{let } \langle \rangle = t_1 \text{ in } t_2 \rightsquigarrow_L \text{let } \langle \rangle = t'_1 \text{ in } t_2} \text{CONGU}_{\text{NIT}_L} \quad \frac{}{\text{let } \langle \rangle = \langle \rangle \text{ in } t_2 \rightsquigarrow_L t_2} \beta_{\text{UNIT}_L} \\
\\
\frac{t \rightsquigarrow_L t'}{\text{case } t \text{ of } \{\text{inj}_1 x \rightarrow t_1; \text{inj}_2 y \rightarrow t_2\} \rightsquigarrow_L \text{case } t' \text{ of } \{\text{inj}_1 x \rightarrow t_1; \text{inj}_2 y \rightarrow t_2\}} \text{CONGC}_{\text{ASE}_L} \\
\\
\frac{}{\text{case inj}_1 t \text{ of } \{\text{inj}_1 x \rightarrow t_1; \text{inj}_2 y \rightarrow t_2\} \rightsquigarrow_L [t/x]t_1} \text{CASEINJ}_{1L} \\
\\
\frac{}{\text{case inj}_2 t \text{ of } \{\text{inj}_1 x \rightarrow t_1; \text{inj}_2 y \rightarrow t_2\} \rightsquigarrow_L [t/y]t_2} \text{CASEINJ}_{2L}
\end{array}$$

$\Gamma \vdash_L t : A$ Typing rules

$$\begin{array}{c}
\frac{\Gamma_1 \vdash_L t_1 : A \quad \Gamma_2 \vdash_L t_2 : B}{\Gamma_1 + \Gamma_2 \vdash_L \langle t_1, t_2 \rangle : A \otimes B} \otimes\text{-I}_L \quad \frac{\Gamma_1 \vdash_L t_1 : A \otimes B \quad \Gamma_2, x : A, y : B \vdash_L t_2 : C}{\Gamma_1 + \Gamma_2 \vdash_L \text{let } \langle x, y \rangle = t_1 \text{ in } t_2 : C} \otimes\text{-E}_L \\
\\
\frac{}{\emptyset \vdash_L \langle \rangle : \text{unit}} \mathbf{1}\text{-I}_L \quad \frac{\Gamma_1 \vdash_L t_1 : \text{unit} \quad \Gamma_2 \vdash_L t_2 : A}{\Gamma_1 + \Gamma_2 \vdash_L \text{let } \langle \rangle = t_1 \text{ in } t_2 : A} \mathbf{1}\text{-E}_L \quad \frac{\Gamma \vdash_L t : A}{\Gamma \vdash_L \text{inj}_1 t : A \oplus B} \oplus\text{-I}_{1L} \\
\\
\frac{\Gamma \vdash_L t : B}{\Gamma \vdash_L \text{inj}_2 t : A \oplus B} \oplus\text{-I}_{2L} \quad \frac{\Gamma_1 \vdash_L t : A \oplus B \quad \Gamma_2, x : A \vdash_L t_1 : C \quad \Gamma_2, y : B \vdash_L t_2 : C}{\Gamma_1 + \Gamma_2 \vdash_L \text{case } t \text{ of } \{\text{inj}_1 x \rightarrow t_1; \text{inj}_2 y \rightarrow t_2\} : C} \oplus\text{-E}_L
\end{array}$$

Fig. 5. Linear Core semantics and typing

Syntax. We extend Linear Base and Graded Modal Base (which already share the same syntax) to **Linear Core** and **Graded Modal Core** respectively, with the same extended syntax:

$$\begin{array}{l}
t ::= \dots \mid \langle t_1, t_2 \rangle \mid \text{let } \langle x, y \rangle = t_1 \text{ in } t_2 \mid \langle \rangle \mid \text{let } \langle \rangle = t_1 \text{ in } t_2 \\
\mid \text{inj}_1 t \mid \text{inj}_2 t \mid \text{case } t \text{ of } \{\text{inj}_1 x \rightarrow t_1; \text{inj}_2 y \rightarrow t_2\} \quad (\text{terms})
\end{array}$$

These additional term formers provide product introduction and elimination, unit introduction and elimination, and sum (coproduct) introductions and elimination respectively.

The operational semantics are the standard call-by-name operational semantics for products and sums given in Figure 5 for Linear Core, with the same added to operational semantics of Graded Modal Core. We elide the $\beta\eta$ -equational theory which is standard (see Appendices A.2 [Liepelt et al. 2026a] and A.7 [Liepelt et al. 2026a]).

To type the new syntax, we extend the type system of Linear Base to Linear Core (Section 5.1), and extend Graded Modal Base to Graded Modal Core (Section 5.2). Section 5.3 then gives the translation from Linear Core to Graded Modal Core which is straightforwardly homomorphic. Section 5.4 then shows that the opposite direction is not possible, requiring an extension of Linear Core to **Linear Push Core** in Section 5.5, at the cost of weakening the correspondence between our $\Box$ -modality and Linear Logic's $!$ -modality. We then give the translation from Graded Modal Core to Linear Push Core (Section 5.6) and the translation from Linear Push Core to Graded Modal Core (Section 5.7).

$\Delta \vdash_G t : A$

Typing rules

$$\begin{array}{c}
\frac{\Delta_1 \vdash_G t_1 : A \quad \Delta_2 \vdash_G t_2 : B}{\Delta_1 + \Delta_2 \vdash_G \langle t_1, t_2 \rangle : A \times B} \times\text{-I}_G \quad \frac{\Delta_1 \vdash_G t_1 : A \times B \quad \Delta_2, x : \textcolor{blue}{r}. A, y : \textcolor{blue}{r}. B \vdash_G t_2 : C}{\textcolor{blue}{r}. \Delta_1 + \Delta_2 \vdash_G \text{let } \langle x, y \rangle = t_1 \text{ in } t_2 : C} \times\text{-E}_G \\
\\
\frac{}{\emptyset \vdash_G \langle \rangle : \text{unit}} \text{unit-I}_G \quad \frac{\Delta_1 \vdash_G t_1 : \text{unit} \quad \Delta_2 \vdash_G t_2 : A}{\textcolor{blue}{r}. \Delta_1 + \Delta_2 \vdash_G \text{let } \langle \rangle = t_1 \text{ in } t_2 : A} \text{unit-E}_G \quad \frac{\Delta \vdash_G t : A}{\Delta \vdash_G \text{inj}_1 t : A + B} +\text{I}_G \\
\\
\frac{\Delta \vdash_G t : B}{\Delta \vdash_G \text{inj}_2 t : A + B} +\text{I}_G \quad \frac{\Delta_1 \vdash_G t : A + B \quad \Delta_2, x : \textcolor{blue}{r}. A \vdash_G t_1 : C \quad \Delta_2, y : \textcolor{blue}{r}. B \vdash_G t_2 : C \quad 1 \sqsubseteq \textcolor{blue}{r}}{\textcolor{blue}{r}. \Delta_1 + \Delta_2 \vdash_G \text{case } t \text{ of } \{\text{inj}_1 x \rightarrow t_1; \text{inj}_2 y \rightarrow t_2\} : C} +\text{E}_G
\end{array}$$

Fig. 6. Graded Modal Core additional typing rules.

5.1 Linear Core

Linear Base augmented with products and sums becomes what we call **Linear Core**. Linear Core features the following new type constructors for products, units, and sums:

$$A ::= \dots \mid A \otimes B \mid \text{unit} \mid A \oplus B \quad (\text{types})$$

Typing. Linear Core extends Linear Base typing (given in Figure 2) with the judgments for multiplicative products and their unit, along with additive disjunction, in Figure 5, which are standard from Linear Logic [Girard 1987]. This is the same typing as in linear-base systems such as Granule [Hughes et al. 2020; Orchard et al. 2019], the synthesis calculus of Hughes et al. [2021], and that of Brunel et al. [2014] (although their system has a less general natural numbers data type instead of products and sums, but the same insights apply).

As in Linear Base, any terms comprising multiple sub-terms have their contexts combined using context addition $+$ which adds together the grades of graded assumptions using semiring addition.

5.2 Graded Modal Core

Graded Modal Base augmented with products and sums becomes **Graded Modal Core**. Graded Modal Core features the following new type constructors for products, units, and sums:

$$A ::= \dots \mid A \times B \mid \text{unit} \mid A + B \quad (\text{types})$$

Typing. Graded Modal Core extends Graded Modal Base typing (given in Section 3.1) by the judgments given in Figure 6. The typing is akin to that shown by Abel and Bernardy [2020], in Linear Haskell [Bernardy et al. 2018], and by Choudhury et al. [2021] (although they restrict their products initially to $r = 1$).

In the elimination of the products we may use each variable according to grade r and thus the context Δ_1 for the eliminated product is multiplied by r in the conclusion. Thus, the usage of the components is propagated to demands on the dependencies for the product. The elimination of sums has a similar idea, propagating demands on the variables x and y through to the dependencies of the eliminated sum term t . There is an additional side condition here too ($1 \sqsubseteq r$) accounting for usage incurred by the inspection of the sum constructor, and thus some information gained. This side condition is key to applications of graded coeffect types for information-flow control (2.6) [Abel and Bernardy 2020; Liepelt et al. 2026b]: if the grade here is $r = \text{Hi}$, representing a high-security value, then the side condition fails as $1 = \text{Lo}$ and thus $\text{Lo} \not\sqsubseteq \text{Hi}$. This prevents pattern matching on a high-security value as a control-flow attack to leak information. Lastly note the elimination of the unit type similarly allows (an arbitrary) scaling of the scrutinee's context, in line with presentations in the literature (e.g., [Abel and Bernardy 2020]).

5.3 Translation from Linear Core to Graded Modal Core

We extend the translation from Section 3.2.1 via a homomorphism on the additional syntax:

$$\begin{array}{ll}
 \langle t_1, t_2 \rangle & \triangleq \langle \llbracket t_1 \rrbracket, \llbracket t_2 \rrbracket \rangle \\
 \text{let } \langle x, y \rangle = t_1 \text{ in } t_2 & \triangleq \text{let } \langle x, y \rangle = \llbracket t_1 \rrbracket \text{ in } \llbracket t_2 \rrbracket \\
 \langle \rangle & \triangleq \langle \rangle \\
 \text{let } \langle \rangle = t_1 \text{ in } t_2 & \triangleq \text{let } \langle \rangle = \llbracket t_1 \rrbracket \text{ in } \llbracket t_2 \rrbracket \\
 \text{inj}_1 t & \triangleq \text{inj}_1 \llbracket t \rrbracket \\
 \text{inj}_2 t & \triangleq \text{inj}_2 \llbracket t \rrbracket \\
 \text{case } t \text{ of } \{\text{inj}_1 x \rightarrow t_1; \text{inj}_2 y \rightarrow t_2\} & \triangleq \text{case } \llbracket t \rrbracket \text{ of } \{\text{inj}_1 x \rightarrow \llbracket t_1 \rrbracket; \text{inj}_2 y \rightarrow \llbracket t_2 \rrbracket\} \\
 A \otimes B & \triangleq \llbracket A \rrbracket \times \llbracket B \rrbracket \\
 A \oplus B & \triangleq \llbracket A \rrbracket + \llbracket B \rrbracket \\
 \text{unit} & \triangleq \text{unit}
 \end{array}
 \quad \begin{array}{l} \text{(Term translation)} \\ \text{(Type translation)} \end{array}$$

We can then re-establish the two key results about the translation (proofs in Appendix B.8 [Liepelt et al. 2026a]):

Theorem 7. [Soundness of the Linear Core to Graded Modal Core translation]

- Type preservation: $\Gamma \vdash_L t : A \implies \langle \Gamma \rangle \vdash_G \llbracket t \rrbracket : \llbracket A \rrbracket$;
- Operational correspondence: $t \rightsquigarrow_L t' \implies \llbracket t \rrbracket \rightsquigarrow_G \llbracket t' \rrbracket$;
- Equation preservation: $t \equiv_L t' \implies \llbracket t \rrbracket \equiv_G \llbracket t' \rrbracket$.

A notable point of the type preservation proof is that when translating sum elimination from Linear Core to Graded Modal Core, we pick the grade $r = 1$ which satisfies the side condition that $1 \sqsubseteq 1$ trivially by reflexivity of the pre-order.

5.4 No Type-Sound Translation from Graded Modal Core to Linear Core

We now attempt a similar homomorphic translation ($\llbracket - \rrbracket_\circ$) back from Graded Modal Core to Linear Core, focusing on just products and sums:

$$\begin{array}{ll}
 \llbracket \langle t_1, t_2 \rangle \rrbracket_\circ & \triangleq \langle \llbracket t_1 \rrbracket_\circ, \llbracket t_2 \rrbracket_\circ \rangle \\
 \llbracket \text{let } \langle x, y \rangle = t_1 \text{ in } t_2 \rrbracket_\circ & \triangleq \text{let } \langle x, y \rangle = \llbracket t_1 \rrbracket_\circ \text{ in } \llbracket t_2 \rrbracket_\circ \\
 \llbracket \text{inj}_i t \rrbracket_\circ & \triangleq \text{inj}_i \llbracket t \rrbracket_\circ \\
 \llbracket \text{inj}_2 t \rrbracket_\circ & \triangleq \text{inj}_2 \llbracket t \rrbracket_\circ \\
 \llbracket \text{case } t \text{ of } \{\text{inj}_1 x \rightarrow t_1; \text{inj}_2 y \rightarrow t_2\} \rrbracket_\circ & \triangleq \text{case } \llbracket t \rrbracket_\circ \text{ of } \{\text{inj}_1 x \rightarrow \llbracket t_1 \rrbracket_\circ; \text{inj}_2 y \rightarrow \llbracket t_2 \rrbracket_\circ\} \\
 \llbracket A \times B \rrbracket_\circ & \triangleq \llbracket A \rrbracket_\circ \otimes \llbracket B \rrbracket_\circ \\
 \llbracket A + B \rrbracket_\circ & \triangleq \llbracket A \rrbracket_\circ \oplus \llbracket B \rrbracket_\circ
 \end{array}
 \quad \begin{array}{l} \text{(Term translation)} \\ \text{(Type translation)} \end{array}$$

However, this translation is unsound: consider the following Graded Modal Core terms, f and g , which manifest a distributivity of grading over products and sums:

$$\begin{array}{ll}
 f : (A \times B) \xrightarrow{r} (\Box_r A \times \Box_r B) & g : (A + B) \xrightarrow{r} (\Box_r A + \Box_r B) \quad (\text{where } 1 \sqsubseteq r) \\
 f = \lambda p. (\text{let } \langle x, y \rangle = p \text{ in } \langle [x], [y] \rangle) & g = \lambda e. \text{case } e \text{ of } \{\text{inj}_1 x \rightarrow \text{inj}_1 [x]; \text{inj}_2 y \rightarrow \text{inj}_2 [y]\}
 \end{array}$$

Thus f is a function that distributes the resource requirements on a pair onto its components, and similarly, g distributes the resource requirements on a sum onto the components of its injections.

Applying the above translation to f and g yields the following ill-typed terms (to avoid clutter, we consider $\llbracket A \rrbracket_\circ$ interchangeable with A):

$$\begin{aligned} \llbracket f \rrbracket_\circ &: \Box_r(A \otimes B) \multimap (\Box_r A \otimes \Box_r B) && \text{(ill-typed)} \\ \llbracket f \rrbracket_\circ &= \lambda b. (\text{let } [p] = b \text{ in let } \langle x, y \rangle = p \text{ in } \langle [x], [y] \rangle) \\ \llbracket g \rrbracket_\circ &: \Box_r(A \oplus B) \multimap (\Box_r A \oplus \Box_r B) \\ \llbracket g \rrbracket_\circ &= \lambda b. (\text{let } [e] = b \text{ in case } e \text{ of } \{\text{inj}_1 x \rightarrow \text{inj}_1 [x]; \text{inj}_2 y \rightarrow \text{inj}_2 [y]\}) \end{aligned}$$

To see why these are ill-typed, the following gives a partial derivation for the body of $\llbracket f \rrbracket_\circ$:

$$\frac{\frac{p : A \otimes B \vdash_L p : A \otimes B \quad \text{VAR}_L}{p : [A \otimes B]_r \vdash_L p : A \otimes B} \quad \frac{r = 1}{\text{DER}_L^*} \quad \frac{\frac{x : A \vdash_L [x] : \Box_r A \quad ?}{x : \Box_r A + y : \Box_r B \vdash_L \langle [x], [y] \rangle : \Box_r A \otimes \Box_r B} \quad \frac{y : B \vdash_L [y] : \Box_r B \quad ?}{\otimes\text{-I}_L}}{\frac{x : \Box_r A + y : \Box_r B \vdash_L \langle [x], [y] \rangle : \Box_r A \otimes \Box_r B}{\otimes\text{-E}_L}} \quad \frac{p : [A \otimes B]_r \vdash_L \text{let } \langle x, y \rangle = p \text{ in } \langle [x], [y] \rangle : \Box_r A \otimes \Box_r B}{\text{LET}_L} \quad b : \Box_r(A \otimes B) \vdash_L \text{let } [p] = b \text{ in let } \langle x, y \rangle = p \text{ in } \langle [x], [y] \rangle : \Box_r A \otimes \Box_r B$$

At the points marked with question marks (?), we cannot apply promotion (PR_L) to x and y , because the *graded* predicate in the antecedent does not hold; they are linear variables. Further, to apply DER_L at the point marked with an asterisk (*), we would have to restrict to $r = 1$. A similar attempted derivation for $\llbracket g \rrbracket_\circ$, commuting the graded modality and sum types, encounters the same issues.

One may wonder if any sound translation is possible at all. Hughes et al. [2021] have explored precisely this point and show that linear-base and graded-base systems exhibit a fundamental difference in expressivity. Linear Core is akin to Linear Logic, in that it does not admit derivation of the distributive law $!(A \otimes B) \multimap !A \otimes !B$ as the semantics of multiplicative tensor $\otimes$ is that each component must be used equally; such a distributive law would allow A and B to be decoupled and used differently. On the other hand, Graded Base and the graded-base systems in the literature cited above admit such a distributive law as seen with f . Thus Graded Base cannot model Linear Logic's $!$ -modality exactly through *any* choice of semiring for the graded modality.

Hughes et al. [2021] propose extensions to graded systems to bring back the restrictions of Linear Logic. Here we explore a different route, namely we extend Linear Core by further constructs which admit the augmented expressivity of Graded Modal Core, giving the system **Linear Push Core**.

5.5 Linear Push Core

We extend Linear Core to **Linear Push Core** with new syntax, heaving it to a level of expressivity on par with Graded Modal Core, leaving behind the restrictions of Linear Logic discussed in 5.4.

Since the essential missing piece is the distributive laws between the graded modality and products and sums, we complete the syntax of Linear Push Core with the following two constructs:

$$t ::= \dots \mid \text{push}_{\otimes} t \mid \text{push}_{\oplus} t \mid \text{push}_{\text{unit}} t \quad \text{(terms)}$$

The syntax is non-standard but isomorphic to other approaches; Granule achieves the same expressivity implicitly via pattern matching [Orchard et al. 2019], but we prefer to introduce explicit syntax here. These two constructs provide exactly the distributive behaviour captured in product and sum elimination in Graded Modal Core. Figure 7 gives the typing and operational model.

$\Gamma \vdash_L t : A$

 Typing rules

$$\frac{\Gamma \vdash_L t : \Box_r(A \otimes B)}{\Gamma \vdash_L \text{push}_{\otimes} t : \Box_r A \otimes \Box_r B} \text{PUSH}_{\otimes} \quad \frac{1 \sqsubseteq r \quad \Gamma \vdash_L t : \Box_r(A \oplus B)}{\Gamma \vdash_L \text{push}_{\oplus} t : \Box_r A \oplus \Box_r B} \text{PUSH}_{\oplus} \quad \frac{\Gamma \vdash_L t : \Box_r \text{unit}}{\Gamma \vdash_L \text{push}_{\text{unit}} t : \text{unit}} \text{PUSH}_{\text{unit}}$$

$t \rightsquigarrow_L t'$

 Operational semantics for push constructors

$$\frac{\text{push}_{\otimes}[\langle t_1, t_2 \rangle] \rightsquigarrow_L \langle [t_1], [t_2] \rangle \quad \beta\text{PUSH}_{\otimes L} \quad \text{push}_{\text{unit}}[\langle \rangle] \rightsquigarrow_L \langle \rangle \quad \beta\text{PUSH}_{\text{unit} L} \quad \text{push}_{\oplus}[\text{inj}_i t] \rightsquigarrow_L \text{inj}_i [t] \quad \beta\text{PUSH}_{\oplus L}}{t \rightsquigarrow_L t'} \text{CONGPUSH}_{\otimes L} \quad \frac{t \rightsquigarrow_L t'}{\text{push}_{\otimes} t \rightsquigarrow_L \text{push}_{\otimes} t'} \text{CONGPUSH}_{\otimes L} \quad \frac{t \rightsquigarrow_L t'}{\text{push}_{\text{unit}} t \rightsquigarrow_L \text{push}_{\text{unit}} t'} \text{CONGPUSH}_{\text{unit} L} \quad \frac{t \rightsquigarrow_L t'}{\text{push}_{\oplus} [t] \rightsquigarrow_L \text{push}_{\oplus} [t']} \text{CONGPUSH}_{\oplus L} \quad \frac{t \rightsquigarrow_L t'}{\text{push}_{\oplus} [t] \rightsquigarrow_L \text{push}_{\oplus} [t']} \text{CONGPUSH}_{\oplus L} \quad \frac{t \rightsquigarrow_L t'}{\text{push}_{\text{unit}} [t] \rightsquigarrow_L \text{push}_{\text{unit}} [t']} \text{CONGPUSH}_{\text{unit} L}$$

Fig. 7. Linear Push Core

5.6 Translation from Graded Modal Core to Linear Push Core

We define the translation from Graded Modal Core into Linear Push Core as follows (fixing the unsound translation to Linear Core from Section 5.4 in the product and sum elimination):

$$\begin{aligned} \llbracket \langle t_1, t_2 \rangle \rrbracket &\triangleq \langle \llbracket t_1 \rrbracket, \llbracket t_2 \rrbracket \rangle \\ \llbracket \text{let } \langle x, y \rangle = t_1 \text{ in } t_2 \rrbracket &\triangleq \text{let } \langle x', y' \rangle = \text{push}_{\otimes}[\llbracket t_1 \rrbracket] \text{ in let } [x] = x' \text{ in let } [y] = y' \text{ in } \llbracket t_2 \rrbracket \\ \llbracket \text{inj}_1 t \rrbracket &\triangleq \text{inj}_1 \llbracket t \rrbracket \\ \llbracket \text{inj}_2 t \rrbracket &\triangleq \text{inj}_2 \llbracket t \rrbracket \\ \llbracket \text{case } t \text{ of } \{\text{inj}_1 x \rightarrow t_1; \text{inj}_2 y \rightarrow t_2\} \rrbracket &\triangleq \text{case } \text{push}_{\oplus}[\llbracket t \rrbracket] \text{ of } \{\text{inj}_1 x' \rightarrow \text{let } [x] = x' \text{ in } \llbracket t_1 \rrbracket; \text{inj}_2 y' \rightarrow \text{let } [y] = y' \text{ in } \llbracket t_2 \rrbracket\} \\ \llbracket \langle \rangle \rrbracket &\triangleq \langle \rangle \\ \llbracket \text{let } \langle \rangle = t_1 \text{ in } t_2 \rrbracket &\triangleq \text{let } \langle \rangle = \text{push}_{\text{unit}}[\llbracket t_1 \rrbracket] \text{ in } \llbracket t_2 \rrbracket \end{aligned} \quad \text{(Term translation)}$$

$$\begin{aligned} \llbracket A \times B \rrbracket &\triangleq \llbracket A \rrbracket \otimes \llbracket B \rrbracket \\ \llbracket A + B \rrbracket &\triangleq \llbracket A \rrbracket \oplus \llbracket B \rrbracket \\ \llbracket \text{unit} \rrbracket &\triangleq \text{unit} \end{aligned} \quad \text{(Type translation)}$$

Translation of a product elimination on t_1 first promotes the product to a graded modality which is then ‘pushed into’ each component of the product via $\text{push}_{\otimes}$. The resulting graded modal components x' and y' are then eliminated in the scope of t_2 . A similar idea is applied for sum elimination. Thus this translation gets at the heart of what is happening inside Graded Modal Core (and why it differs from Linear Core): its eliminators distribute the coefficient analysis inside products and sums, which we now capture explicitly here in Linear Push Core.

We now re-establish the core results (proofs for which are in Appendix B.7 [Liepelt et al. 2026a]):

Theorem 8. [Soundness of the Graded Modal Core to Linear Push Core translation]

- Type preservation: $\Delta \vdash_G t : A \implies \llbracket \Delta \rrbracket \vdash_L \llbracket t \rrbracket : \llbracket A \rrbracket$;
- Operational correspondence: $t \rightsquigarrow_G t' \implies \llbracket t \rrbracket \rightsquigarrow_L^* \llbracket t' \rrbracket$.
- Equation preservation: $(t \equiv_G^{-\eta \otimes \oplus \text{unit}} t') \implies \llbracket t \rrbracket \equiv_L \llbracket t' \rrbracket$.

Our translation does not, in general, preserve η -equalities for products, sums, and unit (it can only preserve η on these types for closed terms); i.e., our translation is not extensional due to additional ‘push’-structure for handling nesting of graded modalities and products, sums, and units.

Given that we have changed Linear Core to Linear Push Core by adding two new constructs, we now re-establish that we can translate Linear Push Core back into Graded Modal Core.

5.7 Translation from Linear Push Core to Graded Modal Core

We extend the Linear Core to Graded Modal Core translation with the following additional cases to handle the extra constructs of Linear Push Core:

$$\begin{aligned} \langle \text{push}_{\otimes} t \rangle &\triangleq \text{let } [x] = \langle t \rangle \text{ in let } \langle y, z \rangle = x \text{ in } \langle [y], [z] \rangle & (\text{Term translation}) \\ \langle \text{push}_{\oplus} t \rangle &\triangleq \text{let } [x] = \langle t \rangle \text{ in case } x \text{ of } \{ \text{inj}_1 y \rightarrow \text{inj}_1 [y]; \text{inj}_2 z \rightarrow \text{inj}_2 [z] \} \\ \langle \text{push}_{\text{unit}} t \rangle &\triangleq \text{let } [x] = \langle t \rangle \text{ in let } \langle \rangle = x \text{ in } \langle \rangle \end{aligned}$$

Note that (for $\text{push}_{\otimes}$ and $\text{push}_{\oplus}$) these are essentially the programs f and g shown in Section 5.4 which are the derivation of push combinators in Graded Modal Core.

We thus (re)establish the key results (proofs in Appendix B.8 [Liepelt et al. 2026a]):

Theorem 9. [Soundness of the Linear Push Core to Graded Modal Core translation]

- Type preservation: $\Gamma \vdash_L t : A \implies \langle \Gamma \rangle \vdash_G \langle t \rangle : \langle A \rangle$;
- Operational correspondence: $t \rightsquigarrow_L t' \implies \langle t \rangle \rightsquigarrow_G^* \langle t' \rangle$;
- Equation preservation: $t \equiv_L t' \implies \langle t \rangle \equiv_G \langle t' \rangle$.

Note that whilst the operational correspondence from Linear Base to Graded Modal Base mapped a single reduction $t \rightsquigarrow_L t'$ to a single graded-base reduction $\langle t \rangle \rightsquigarrow_G \langle t' \rangle$ this is now not possible for Linear Push Core. Instead, a single reduction in Linear Push Core for the push combinators corresponds to several reductions in Graded Modal Core.

Remark 5.1. Graded Modal Core extends Graded Modal Base rather than Graded Base. If Graded Modal Core did not have a graded modality it could not give a precise type to projection on products:

$$\emptyset \vdash_G (\lambda z. \text{let } \langle x, y \rangle = z \text{ in } x) : (A \times B) \xrightarrow{r} A$$

where $1 \sqsubseteq r$ due to the use of x and $0 \sqsubseteq r$ due to the absence of y . For the exact-usage natural numbers semiring this would not be typeable as there is no such r satisfying both constraints. For a semiring of intervals denoting lower-bound and upper-bound usage [Orchard et al. 2019], the constraints would be satisfied by $r = [0, 1]$.

We instead opted to include the graded modality in Graded Modal Core as in the work of Abel and Bernardy [2020], Moon et al. [2021], and others [Hughes and Orchard 2024; Liepelt et al. 2026b]. Thus we can specify for this example a more fine-grained type via the graded modality, where the B component of the product is unused:

$$\emptyset \vdash_G \lambda z. \text{let } \langle x, y \rangle = z \text{ in let } [y'] = y \text{ in } x : (A \times \Box_0 B) \xrightarrow{1} A$$

Without the inclusion of graded modalities, Graded Modal Core is less expressive than Linear Core.

Note that some dependently-typed graded-base systems such as GRAD [Choudhury et al. 2021] and QTT [Atkey 2018] omit the graded modality but include a graded dependent Σ type which can be used to encode the graded modality.

6 Discussion and Conclusion

6.1 Is One Approach Better than the Other?

A natural question at this point from the language designer's perspective is: which approach to linearity should I use in my language? We do not try to offer an authoritative answer, but instead give some observations and points to consider.

Clearly, graded-base style approaches afford a simpler system for the end-user with less syntactic overhead. The pervasive grading obviates the need for most promotions and let-unboxings. But while having explicit push syntax is an additional burden to the user of a linear-base system, in practice this is not necessary. For example, Granule handles this implicitly via pattern matching [Hughes et al. 2021; Orchard et al. 2019].

On the flip-side, it may be desirable to disallow push for reasoning about certain domains, where a coefficient should not distribute over the components of a pair. An example for this might be for reasoning about interdependent threads that need to run in parallel. Given what we have shown in this paper, linear base systems would seem better equipped to offer this more restrictive modality, however Hughes et al. [2021] have shown that it is possible to recover the non-distributive behaviour of the $\square$ -modality with respect to products in a graded-base setting. They argue that any useful language implementation would reasonably have to offer both flavours of the modality.

What about the burden for the language designer? Empirically, the authors have found that proofs for graded-base systems are significantly shorter, due, in part, to the homogeneous treatment of assumptions, i.e. the lack of split between linear and non-linear free variables. Especially when retrofitting linear or coefficient types to an existing language, only Graded Base seems to offer a backwards-compatible way and in such settings this is the only practical option (see, e.g., the addition of graded types to Haskell for linearity [Bernardy et al. 2018]).

6.2 Graded Type Systems in Practice

The LinearTypes language extension available since GHC 9.0.1 [The GHC Developers 2020] enables Linear Haskell in the context of an industrial-strength programming language. As discussed in this paper, Linear Haskell is a graded-base system, which is currently semiring-monomorphic: in the theory of Bernardy et al. [2018] the *none-one-tons* semiring is used, $\{0, 1, \omega\}$. However, as of the time of writing, the implementation in GHC offers no 0, and thus the grades range over $\{1, \omega\}$, written in Haskell as One and Many, respectively. Consider the following Linear Haskell program:

```
1 {-# LANGUAGE LinearTypes #-}
2 {-# LANGUAGE DataKinds #-}
3 import GHC.Types
4 const :: a %One → b %Many → a
5 const x y = x
```

By default arrows are Many graded, thus the above can also be written as:

```
1 const' :: a %One → b → a
2 const' x y = x
```

Notably, Linear Haskell supports *multiplicity polymorphism*, however the multiplicities are inhabited by a single semiring. Future work would be a GradedTypes extension to generalise Haskell to be polymorphic over semirings, a feature supported by Granule, where Orchard et al. [2019] use an SMT-solver to discharge constraints over the grades. It is not yet clear how this could be practically integrated with GHC, perhaps as a compiler plugin akin to Liquid Haskell [Vazou et al. 2020].

Idris 2 also implements a graded-base system, based on Quantitative Type Theory (due initially to McBride [2016] and later adapted slightly by Atkey [2018]), also providing grades in the *none-one-tons* semiring [Brady 2021a,b]. For example, const can be rendered in Idris 2:

```
1 const : {0 a : Type} → {0 b : Type} → a → (0 x : b) → a
2 const x y = x
```

Granule is a research language serving as a test bed for ideas in graded types [Orchard et al. 2019]. In our classification, Granule is a linear-base system by default.

Semiring-graded modalities written here as $\Box_r A$ in the core calculi are written as ‘ $a \llbracket r \rrbracket$ ’ in Granule. While Granule uses $\rightarrow$ for the function arrow ($\rightarrow$ in ASCII), as in Haskell or ML, it is really a linear arrow. Our running example looks as follows in Granule:

```
1  const :  $\forall \{a : \text{Type}, b : \text{Type}, s : \text{Semiring}\} . a \rightarrow b \llbracket 0 : s \rrbracket \rightarrow a$ 
2  const x y = let  $\llbracket \_ \rrbracket = y$  in x
```

Granule also includes a language extension that changes its semantics to Graded Base and whose syntax for graded arrows is borrowed from Linear Haskell, but supporting semiring polymorphism:

```
1  language GradedBase
2  const' :  $\forall \{a : \text{Type}, b : \text{Type}, s : \text{Semiring}\} . a \rightarrow b \% (0 : s) \rightarrow a$ 
3  const' x y = x
```

6.3 Comparison with Effects

Here we have considered the relationship between Linear Base with linear function spaces and linear assumptions as well as graded, and Graded Base where grades are pervasive. This situation is somewhat analogous to the relationship between *effect systems* and *monads*.

The type-and-effect systems of Gifford and Lucassen [1986; 1988] provide a technique for augmenting a type system with information about the side effects a program may perform whilst computing a result. Judgments and function arrows carry effect information:

$$\Gamma \vdash M : \tau, F \qquad \Gamma' \vdash \lambda x. M' : \sigma \xrightarrow{F'} \tau, \emptyset$$

i.e., the program M computes a result of type τ and may perform side effects described by F . Traditionally F is a set of effect ‘labels’, describing operations that have been performed. This was later generalised to semilattices and ordered monoids [Mycroft et al. 2016; Nielson et al. 1999], enabling other kinds of analysis. On the right, the function term encapsulates side effects F' and is itself pure, marked with $\emptyset$. This is analogous to Graded Base, where coeffect grades annotate function types (instead representing usage of the parameter).

Monads on the other hand provide a semantic technique for giving a denotational model of various kinds of effect [Moggi 1989, 1991]. The *monadic meta language* provides a type constructor T where $T\tau$ captures the type of computations performing some side effects in order to compute a τ value [Moggi 1991]. Thus, terms of the monadic meta language have types of the form:

$$\Gamma \vdash M : T\tau \qquad \Gamma \vdash \lambda x. M' : \sigma \rightarrow T\tau$$

On the right is an example of a function that encapsulates side effects in its result, but the function itself does not need wrapping in the monadic T constructor since abstraction is pure (side-effect free). This is analogous to Linear Base if we had only a single comonadic modality.

Wadler and Thiemann [2003] observed that the structure of effect systems and the structure of the monadic meta language are the same. They thus married the two by annotating the monadic constructor with effect information, i.e.,

$$\Gamma \vdash M : T_F \tau \qquad \Gamma' \vdash \lambda x. M' : T_\emptyset(\sigma \rightarrow T_{F'} \tau)$$

Later, this indexed structure was captured generally by the notion of *graded monads* [Katsumata 2014; Orchard et al. 2014]. Graded monads generalise monads to a family of constructors indexed by the elements of a monoid, and whose operations are stratified and accounted for by the monoidal structure. Graded monads thus unify the semantics of effects with a syntactic analysis in the types.

Linear Base with graded modality $\Box_r A$ is thus a dual notion to the graded monadic calculus, unifying the semantics of *coeffects* with a static analysis in the types, but with linearity as its base typing system. We could thus see Linear Base as a *graded comonadic meta language*. In contrast, Graded Base is analogous to effect systems. Indeed the early work on *coeffect systems* (captured here as Graded Base) has no graded modality and was motivated by dualising effect systems [Petricek et al. 2014, 2013]. As seen, without this graded modality, Graded Base cannot localise information to sub-parts of complex types as discussed in Section 5.

6.4 Adjoint Relationship

Vollmer et al. [2025] follow the Mixed Linear and Non-Linear logic approach of Benton [1994] to provide a pair of graded calculi with an adjunction mediating between them. One calculus is a linear-base system with linear functions and a mixed context of linear and graded assumptions. The other calculus is a graded-base system but with no graded function arrow. A modality Lin embeds formulae from the linear-base system into the graded-base. Conversely, a graded modality Grd_r embeds formulae from the graded-base system into the linear-base system, capturing their grade r and incurring a promotion (scaling of the graded context). From Lin and Grd_r a graded comonadic modality is derived following the adjoint resolution of graded comonads [Fujii et al. 2016].

The work seems related but on a more semantic level. Whilst we showed that our translation is not adjoint in Section 2.3, our two calculi have function types on both sides which seemed to be the main blocker to forming an adjunction. Further work is to ascertain whether our translation could be restricted to the systems considered by Vollmer et al. [2025] and then used as a model of the adjunction by using our translations as the semantics of the modalities operating on syntactic categories of terms. Another avenue would be to explore whether generalising function spaces on the graded-base side would undermine their adjunction construction in general.

6.5 Filling Out the Design Space: Cartesian Base

A central idea in this work has been to consider the ‘base’ notion of usage in graded systems: linear-base systems have linear propositions, with a linear axiom rule, and linear function type; graded-base systems have graded propositions, with a graded axiom rule, and a graded function type. This framing suggest an unexplored point in the design space: what about a graded system with a cartesian base, i.e., with a cartesian axiom rule and standard cartesian function (type), with contexts comprising both cartesian and graded assumptions?

Crucially we must then consider how the cartesian part interacts with the graded part. On the left, we have the rules for dereliction and application in Linear Base, on the right have the potential rules for a Cartesian Base. Since cartesian propositions can be used arbitrarily we now require an element $\omega \in \mathcal{R}$ of the semiring which behaves in an absorbing fashion for both addition $+$ and multiplication $\cdot$. Dereliction in Cartesian Base can then mark a cartesian assumption with ω . Application must also scale the argument context Γ_2 by ω since the function arrow is cartesian and so may use its argument arbitrarily:

$$\begin{array}{c}
 \text{(Linear Base)} \\
 \frac{\Gamma, x : A \vdash_L t : B}{\Gamma, x : [A]_1 \vdash_L t : B} (der) \\
 \frac{\Gamma_1 \vdash_L t_1 : A \multimap B \quad \Gamma_2 \vdash_L t_2 : A}{\Gamma_1 + \Gamma_2 \vdash_L t_1 t_2 : B} (app)
 \end{array}
 \qquad
 \begin{array}{c}
 \text{(Cartesian Base)} \\
 \frac{\Gamma, x : A \vdash t : B}{\Gamma, x : [A]_\omega \vdash t : B} (der) \\
 \frac{\Gamma_1 \vdash t_1 : A \rightarrow B \quad \Gamma_2 \vdash t_2 : A}{\Gamma_1 + \omega * \Gamma_2 \vdash t_1 t_2 : B} (app)
 \end{array}$$

The above seems a reasonable approach to building a Cartesian Base system. However, this system provides little benefit as the grading quickly degenerates to being always ω in derivations. Consider below the derivation of the ‘copy’ operation on an axiom to produce a pair:

$$\frac{\frac{x : A \vdash_L x : A}{x : [A]_1 \vdash_L x : A} \quad \frac{x : A \vdash_L x : A}{x : [A]_1 \vdash_L x : A}}{x : [A]_{1+1} \vdash_L (x, x) : A \otimes A} \quad \frac{\frac{x : A \vdash x : A}{x : [A]_\omega \vdash x : A} \quad \frac{x : A \vdash x : A}{x : [A]_\omega \vdash x : A}}{x : [A]_\omega \vdash (x, x) : A \otimes A}$$

In Linear Base, there is a precise account of the non-linearity incurred: the term uses x with graded $1 + 1$. However, in Cartesian Base we can only say ω due to the dereliction rule. In the end, the Cartesian Base system is ultimately no more informative as a graded system than standard intuitionistic, cartesian systems.

6.6 Resource-Aware Operational Semantics

The operational semantics presented here leverage standard λ -calculus CBN operational semantics, which are typical for the literature studied. More recent work has developed *resource-aware* operational semantics [Bianchini et al. 2023a,b; Choudhury et al. 2021; Torczon et al. 2023]. These approaches are based on an abstract machine approach with a variable environment, rather than a substitution-based model. Conceptually, a resource-aware semantics keeps track not just of variable mappings in the evaluation environment, but more broadly tracks resource supplies, which are consumed by variable use, thereby making the connection between static and dynamic resource guarantees: A *resource-sound* type system only accepts *well-resourced* programs, whose evaluation under the resource-aware operational semantics respects the static resource bounds, meaning (1) evaluation does not get stuck due to depletion (*resource progress*); and (2) every evaluation step leads to a program with comparable resource requirements under the (transitive) approximation relation ($\sqsubseteq$) of the resource algebra (*resource conservation*).

Most recently, Bianchini et al. [2025] extend these guarantees to the converse property, ensuring both upper and lower resource bounds, thereby also preventing unused resources (*wastefulness*).

Extending the various flavours of linear- and graded-base calculi and their translations, which form the substrate of this paper, to such resource-aware operational semantics goes beyond the scope of what we are able to investigate here. Instead, we leave this as future work.

6.7 Conclusion

Our goal here has been to serve as a bridge between two different approaches in the literature for constructing graded type systems that capture coeffects (context dependence). We answer the long open question about the relationship between the linear-base and graded-base styles and give a detailed analysis and metatheoretic results. Overall we see that we can translate between the two flavours: with Linear Base and Graded Base having mutual embeddings (but requiring a global CPS encoding for the Linear Base into Graded Base direction); Linear Base and Graded Modal Base having (more straightforward) mutual embeddings; and Linear Push Core and Graded Modal Core having mutual embeddings: these are the points where the two styles of calculi can be inter-translated (see Figure 1).

Now that graded types for tracking various kinds of program property are moving from the laboratory to the field, with Haskell [Bernardy et al. 2018], Idris [Brady 2021a], Granule [Orchard et al. 2019], and lately OCaml with its moding system [Georges et al. 2025; Lorenzen et al. 2024], our hope is that this study can serve as a design guide for further theoretical work and for language designers wishing to integrate graded types in some way, drawing from a wider arsenal of techniques and metatheoretic results.

Acknowledgments

We thank the anonymous reviewers of this work for their comments and feedback. During its early stages, this work was partly supported by EPSRC grant EP/T013516/1. Orchard also received support through Schmidt Sciences, LLC.

References

- Andreas Abel and Jean-Philippe Bernardy. 2020. A unified view of modalities in type systems. *Proc. ACM Program. Lang.* 4, ICFP (2020), 90:1–90:28. doi:10.1145/3408972
- Andreas Abel, Nils Anders Danielsson, and Oskar Eriksson. 2023. A Graded Modal Dependent Type Theory with a Universe and Erasure, Formalized. *Proc. ACM Program. Lang.* 7, ICFP (2023), 920–954. doi:10.1145/3607862
- Robert Atkey. 2018. Syntax and Semantics of Quantitative Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*. 56–65. doi:10.1145/3209108.3209189
- P. N. Benton. 1994. A Mixed Linear and Non-Linear Logic: Proofs, Terms and Models (Extended Abstract). In *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers (Lecture Notes in Computer Science, Vol. 933)*, Leszek Pacholski and Jerzy Tiuryn (Eds.). Springer, 121–135. doi:10.1007/BFB0022251
- Jean-Philippe Bernardy, Mathieu Boespflug, Ryan R. Newton, Simon Peyton Jones, and Arnaud Spiwack. 2018. Linear Haskell: practical linearity in a higher-order polymorphic language. *Proc. ACM Program. Lang.* 2, POPL (2018), 5:1–5:29. doi:10.1145/3158093
- Riccardo Bianchini, Francesco Dagnino, Paola Giannini, and Elena Zucca. 2023a. Multi-Graded Featherweight Java. In *37th European Conference on Object-Oriented Programming, ECOOP 2023, July 17-21, 2023, Seattle, Washington, United States (LIPIcs, Vol. 263)*, Karim Ali and Guido Salvaneschi (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 3:1–3:27. doi:10.4230/LIPICS.ECOOP.2023.3
- Riccardo Bianchini, Francesco Dagnino, Paola Giannini, and Elena Zucca. 2023b. Resource-Aware Soundness for Big-Step Semantics. *Proc. ACM Program. Lang.* 7, OOPSLA2 (2023), 1281–1309. doi:10.1145/3622843
- Riccardo Bianchini, Francesco Dagnino, Paola Giannini, and Elena Zucca. 2025. Don't exhaust, don't waste. *CoRR* abs/2507.13792 (2025). arXiv:2507.13792 doi:10.48550/ARXIV.2507.13792
- Riccardo Bianchini, Francesco Dagnino, Paola Giannini, Elena Zucca, and Marco Servetto. 2022. Coeffects for sharing and mutation. *Proc. ACM Program. Lang.* 6, OOPSLA2 (2022), 870–898. doi:10.1145/3563319
- Edwin C. Brady. 2021a. Idris 2: Quantitative Type Theory in Practice. In *35th European Conference on Object-Oriented Programming, ECOOP 2021, July 11-17, 2021, Aarhus, Denmark (Virtual Conference) (LIPIcs, Vol. 194)*, Anders Møller and Manu Sridharan (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 9:1–9:26. doi:10.4230/LIPICS.ECOOP.2021.9
- Edwin C. Brady. 2021b. Idris 2: Quantitative Type Theory in Practice (Artifact). *Dagstuhl Artifacts Ser.* 7, 2 (2021), 10:1–10:7. doi:10.4230/DARTS.7.2.10
- Aloïs Brunel, Marco Gaboardi, Damiano Mazza, and Steve Zdancewic. 2014. A core quantitative coeffect calculus. In *European Symposium on Programming Languages and Systems*. Springer, 351–370.
- Pritam Choudhury, Harley Eades III, Richard A. Eisenberg, and Stephanie Weirich. 2021. A graded dependent type system with a usage-aware semantics. *Proc. ACM Program. Lang.* 5, POPL (2021), 1–32. doi:10.1145/3434331
- Soichiro Fujii, Shin-ya Katsumata, and Paul-André Mellies. 2016. Towards a Formal Theory of Graded Monads. In *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9634)*, Bart Jacobs and Christof Löding (Eds.). Springer, 513–530. doi:10.1007/978-3-662-49630-5_30
- Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C Pierce. 2013. Linear dependent types for differential privacy. In *Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 357–370.
- Marco Gaboardi, Shin-ya Katsumata, Dominic A. Orchard, Flavien Breuvar, and Tarmo Uustalu. 2016. Combining effects and coeffects via grading. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*, Jacques Garrigue, Gabriele Keller, and Eijiro Sumii (Eds.). ACM, 476–489. doi:10.1145/2951913.2951939
- Aïna Linn Georges, Benjamin Peters, Laila Elbeheiry, Leo White, Stephen Dolan, Richard A. Eisenberg, Chris Casinghino, François Pottier, and Derek Dreyer. 2025. Data Race Freedom à la Mode. *Proc. ACM Program. Lang.* 9, POPL (2025), 656–686. doi:10.1145/3704859
- Dan R. Ghica and Alex I. Smith. 2014. Bounded linear types in a resource semiring. In *Programming Languages and Systems*. Springer, 331–350.
- David K Gifford and John M Lucassen. 1986. Integrating functional and imperative programming. In *Proceedings of the 1986 ACM Conference on LISP and Functional Programming*. 28–38.

- Jean-Yves Girard. 1987. Linear logic. *Theoretical computer science* 50, 1 (1987), 1–101.
- Jean-Yves Girard, Andre Scedrov, and Philip J Scott. 1992. Bounded linear logic: a modular approach to polynomial-time computability. *Theoretical computer science* 97, 1 (1992), 1–66.
- Jack Hughes, Danielle Marshall, James Wood, and Dominic Orchard. 2021. Linear Exponentials as Graded Modal Types. In *5th International Workshop on Trends in Linear Logic and Applications (TLLA 2021)*. Rome (virtual), Italy (Jun 2021), <https://hal-lirmm.ccsd.cnrs.fr/lirmm-03271465>.
- Jack Hughes and Dominic Orchard. 2020. Resourceful Program Synthesis from Graded Linear Types. In *Logic-Based Program Synthesis and Transformation - 30th International Symposium, LOPSTR 2020, Bologna, Italy, September 7-9, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12561)*, Maribel Fernández (Ed.). Springer, 151–170. doi:10.1007/978-3-030-68446-4_8
- Jack Hughes and Dominic Orchard. 2024. Program Synthesis from Graded Types. In *Programming Languages and Systems - 33rd European Symposium on Programming, ESOP 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 14576)*, Stephanie Weirich (Ed.). Springer, 83–112. doi:10.1007/978-3-031-57262-3_4
- Jack Hughes, Michael Vollmer, and Dominic Orchard. 2020. Deriving Distributive Laws for Graded Linear Types. In *Proceedings Second Joint International Workshop on Linearity & Trends in Linear Logic and Applications, Linearity&TLLA@IJCAR-FSCD 2020, Online, 29-30 June 2020 (EPTCS, Vol. 353)*, Ugo Dal Lago and Valeria de Paiva (Eds.). 109–131. doi:10.4204/EPTCS.353.6
- Shin-ya Katsumata. 2014. Parametric effect monads and semantics of effect systems. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, Suresh Jagannathan and Peter Sewell (Eds.). ACM, 633–646. doi:10.1145/2535838.2535846
- Vilem Liepelt, Danielle Marshall, and Dominic Orchard. 2026a. Same Coeffect, Different Base: Connecting Two Dominant Approaches to Graded Types (extended version). *arXiv* (2026). doi:10.48550/arXiv.2606.28042
- Vilem Liepelt, Danielle Marshall, Dominic Orchard, Vineet Rajani, and Michael Vollmer. 2026b. On Graded Coeffect Types for Information-Flow Control. In *Languages, Compilers, Analysis - From Beautiful Theory to Useful Practice - Essays Dedicated to Alan Mycroft on the Occasion of His Retirement (Lecture Notes in Computer Science, Vol. 15500)*, Dominic Orchard, Tomas Petricek, and Jeremy Singer (Eds.). Springer, 114–148. doi:10.1007/978-3-032-08187-2_7
- Sam Lindley. 2025. Encoding Product Types. In *Proceedings of the Workshop Dedicated to Olivier Danvy on the Occasion of His 64th Birthday*. 142–151.
- Anton Lorenzen, Leo White, Stephen Dolan, Richard A. Eisenberg, and Sam Lindley. 2024. Oxidizing OCaml with Modal Memory Management. *Proc. ACM Program. Lang.* 8, ICFP (2024), 485–514. doi:10.1145/3674642
- John M Lucassen and David K Gifford. 1988. Polymorphic effect systems. In *Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 47–57.
- Conor McBride. 2016. *I Got Plenty o' Nuttin'*. Springer International Publishing, Cham, 207–233. doi:10.1007/978-3-319-30936-1_12
- Eugenio Moggi. 1989. Computational Lambda-Calculus and Monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*. IEEE Computer Society, 14–23. doi:10.1109/LICS.1989.39155
- Eugenio Moggi. 1991. Notions of Computation and Monads. *Inf. Comput.* 93, 1 (1991), 55–92. doi:10.1016/0890-5401(91)90052-4
- B Moon, H Eades III, and D Orchard. 2021. Graded Modal Dependent Type Theory. In *Programming Languages and Systems 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27–April 1, 2021, Proceedings*, Vol. 12648. 462–490.
- Alan Mycroft, Dominic A. Orchard, and Tomas Petricek. 2016. Effect Systems Revisited - Control-Flow Algebra and Semantics. In *Semantics, Logics, and Calculi - Essays Dedicated to Hanne Riis Nielson and Flemming Nielson on the Occasion of Their 60th Birthdays (Lecture Notes in Computer Science, Vol. 9560)*, Christian W. Probst, Chris Hankin, and René Rydhof Hansen (Eds.). Springer, 1–32. doi:10.1007/978-3-319-27810-0_1
- Flemming Nielson, Hanne Riis Nielson, Chris Hankin, Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. 1999. Type and effect systems. *Principles of Program Analysis* (1999), 283–363.
- Dominic Orchard, Vilem Liepelt, and Harley Eades III. 2019. Quantitative program reasoning with graded modal types. *PACMPL* 3, ICFP (2019), 110:1–110:30. doi:10.1145/3341714
- Dominic A. Orchard, Tomas Petricek, and Alan Mycroft. 2014. The semantic marriage of monads and effects. *CoRR* abs/1401.5391 (2014). arXiv:1401.5391 <http://arxiv.org/abs/1401.5391>
- Tomas Petricek, Dominic Orchard, and Alan Mycroft. 2014. Coeffects: a calculus of context-dependent computation. In *Proceedings of the 19th ACM SIGPLAN international conference on Functional programming*. ACM, 123–135.

- Tomas Petricek, Dominic A. Orchard, and Alan Mycroft. 2013. Coeffects: Unified Static Analysis of Context-Dependence. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 7966)*, Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg (Eds.). Springer, 385–397. doi:10.1007/978-3-642-39212-2_35
- Gordon D. Plotkin. 1975. Call-by-name, call-by-value and the λ -calculus. *Theoretical computer science* 1, 2 (1975), 125–159.
- The GHC Developers. 2020. *GHC User's Guide: Linear Types*. https://web.archive.org/web/20241206233933/https://downloads.haskell.org/ghc/9.0.1/docs/html/users_guide/exts/linear_types.html
- Cassia Torczon, Emmanuel Suárez Acevedo, Shubh Agrawal, Joey Velez-Ginorio, and Stephanie Weirich. 2023. Effects and Coeffects in Call-By-Push-Value (Extended Version). *CoRR* abs/2311.11795 (2023). arXiv:2311.11795 doi:10.48550/ARXIV.2311.11795
- Niki Vazou, Eric Seidel, Richard Eisenberg, Simon Peyton Jones, and Dimitrios Vytiniotis. 2020. Liquid Haskell as a GHC Plugin. Presented at the Haskell Implementors' Workshop (HIW) 2020. <https://icfp20.sigplan.org/details/hiw-2020-papers/1/Liquid-Haskell-as-a-GHC-Plugin>
- Victoria Vollmer, Danielle Marshall, Harley Eades III, and Dominic Orchard. 2025. A Mixed Linear and Graded Logic: Proofs, Terms, and Models. 326 (2025), 32:1–32:21. doi:10.4230/LIPICS.CSL.2025.32
- Philip Wadler and Peter Thiemann. 2003. The marriage of effects and monads. *ACM Transactions on Computational Logic (TOCL)* 4, 1 (2003), 1–32.
- David Walker. 2005. Substructural type systems. *Advanced topics in types and programming languages* (2005), 3–44.
- James Wood and Robert Atkey. 2020. A linear algebra approach to linear metatheory. *arXiv preprint arXiv:2005.02247* (2020).
- James Wood and Robert Atkey. 2022. A Framework for Substructural Type Systems. In *Programming Languages and Systems - 31st European Symposium on Programming, ESOP 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings (Lecture Notes in Computer Science, Vol. 13240)*, Ilya Sergey (Ed.). Springer, 376–402. doi:10.1007/978-3-030-99336-8_14
- David A. Wright. 2026. *Counting before Grading*. doi:10.5281/zenodo.20739966
- David A Wright and Clement A Baker-Finch. 1993. Usage analysis with natural reduction types. In *International Workshop on Static Analysis*. Springer, 254–266.

Received 2026-02-19; accepted 2026-05-13