A Graded Modal Dependent Type Theory with a Universe and Erasure, Formalized

Citation for the original published paper (version of record):

Abel, A., Danielsson, N., Eriksson, O. (2023). A Graded Modal Dependent Type Theory with a Universe and Erasure, Formalized. Proceedings of the ACM on Programming Languages, 7(ICFP): 920-954. http://dx.doi.org/10.1145/3607862

A Graded Modal Dependent Type Theory with a Universe and Erasure, Formalized

ANDREAS ABEL, Chalmers University of Technology and the University of Gothenburg, Sweden
NILS ANDERS DANIELSSON, Chalmers University of Technology and the University of Gothenburg, Sweden
OSKAR ERIKSSON, Chalmers University of Technology and the University of Gothenburg, Sweden

We present a graded modal type theory, a dependent type theory with *grades* that can be used to enforce various properties of the code. The theory has Π-types, weak and strong Σ-types, natural numbers, an empty type, and a universe, and we also extend the theory with a unit type and graded Σ-types. The theory is parameterized by a modality, a kind of partially ordered semiring, whose elements (grades) are used to track the usage of variables in terms and types. Different modalities are possible. We focus mainly on quantitative properties, in particular erasure: with the erasure modality one can mark function arguments as erasable.

The theory is fully formalized in Agda. The formalization, which uses a syntactic Kripke logical relation at its core and is based on earlier work, establishes major meta-theoretic properties such as subject reduction, consistency, normalization, and decidability of definitional equality. We also prove a substitution theorem for grade assignment, and preservation of grades under reduction. Furthermore we study an extraction function that translates terms to an untyped λ-calculus and removes erasable content, in particular function arguments with the "erasable" grade. For a certain class of modalities we prove that extraction is sound, in the sense that programs of natural number type have the same value before and after extraction. Soundness of extraction holds also for *open* programs, as long as all variables in the context are erasable, the context is consistent, and *erased matches* are not allowed for weak Σ-types.

CCS Concepts: • **Theory of computation → Type theory**; **Logic and verification**; Linear logic.

Additional Key Words and Phrases: graded modal type theory, dependent types, erasure, modalities, linearity, formalization

ACM Reference Format:
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, Article 220 (August 2023), 35 pages. https://doi.org/10.1145/3607862

1 INTRODUCTION

Some strongly typed programming languages come with static analyses that determine how variables are used in an expression. The results of these analyses may contribute to the decision of whether a program is accepted, or they might inform compiler optimizations or give extra guarantees about accepted programs. Some of these analyses are:

(1) *Variance:* Coq [2023] and Agda [2023] have positivity checking for (co)inductive types, and Scala has variance checking for higher-order subtyping [Steffen 1998; Stucki and Giarrusso 2021].

Authors' addresses: Andreas Abel, andreas.abel@gu.se; Nils Anders Danielsson, nad@cse.gu.se; Oskar Eriksson, oskar.eriksson@gu.se; Department of Computer Science and Engineering, Chalmers University of Technology, 412 96 Göteborg, Sweden.

2475-1421/2023/8-ART220
https://doi.org/10.1145/3607862

(2) *Irrelevance:* Agda allows function arguments to be declared proof-irrelevant [Pfenning 2001; Abel and Scherer 2012].
(3) *Erasure:* Agda and Idris [Brady 2021] allow function arguments to be declared irrelevant for execution, hence erasable during compilation [Tejiščák 2020].
(4) *Linearity:* In Haskell [Bernardy et al. 2017] and Idris variables can be declared to be linear.

Both erasure and linearity are supported by *Quantitative Type Theory* (QTT) [Atkey 2018], in which a semiring of quantities is used to keep track of how variables are used. QTT is based on earlier work by McBride [2016], and there is related work in simply-typed settings by Brunel et al. [2014], Ghica and Smith [2014] and Petricek et al. [2014]. It has further been observed that information-flow analyses fit the same framework, by viewing the lattice of security levels as a partially ordered semiring [Orchard et al. 2019; Abel and Bernardy 2020; Choudhury et al. 2022]. We use the term *graded modal type theory* [Moon et al. 2021] for a type theory with grades taken from a semiring of some kind.

In this work we study a graded modal type theory with full-fledged dependent types, i.e., Π- and Σ-types, a universe, and a type of natural numbers that permits large elimination. We follow Abel [2018] and Moon et al. [2021] in allowing resources to be tracked in types in addition to in terms (but we do not require such tracking). The type theory comes with a meta-theory, including soundness of substitution, subject reduction, normalization, and decidability of definitional equality, proved via logical relations. The meta-theory is fully formalized in Agda: we build on an Agda formalization originally due to Abel et al. [2017].

Our framework supports quantitative instances such as erasure and linearity, and information-flow instances. However, it does not support instances that affect the definitional equality, like (compile-time) irrelevance.

Different instances of the framework lead to different guarantees for accepted programs. For example, quantitative instances can enable certain runtime optimizations, like deallocation of a linear resource after the first access [Atkey 2018; Choudhury et al. 2021] (but we do not prove that this is possible for our framework). In this article we focus on instances that allow erasure interpretations and show that actual erasure of parts marked as erasable does not alter program behaviour. As opposed to syntactic arguments based on reduction relations [Mishra-Linger and Sheard 2008; McBride 2016; Tejiščák 2020], we employ a logical relation between our type theory and an untyped target language. This soundness proof is also fully formalized in Agda, resting on the formalized meta-theory to define the logical relation by recursion on types.

The novelty in our formal treatment of erasure is that we are not only considering closed programs, but also *open* programs as long as all their free variables are marked as erasable, and, importantly, that the open programs live in consistent contexts. The empty type is inhabited in inconsistent contexts, so in that setting impossible branches in programs cannot be ruled out. Several implementations of type theory support the compilation of programs that contain axioms. For instance, Coq allows the compilation of programs that contain "logical axioms", and the manual warns that "inconsistent logical axioms may lead to incorrect or non-terminating extracted terms" [The Coq Development Team 2023]. We prove formally that it is fine to use consistent axioms, as long as the axioms only appear in erasable positions.

We support two variants of our framework, one with and one without *erased matches* for weak Σ-types: if such matches are allowed then matching on an erased pair is permitted if the obtained components are only used in erased positions. Erased matches are akin to Coq's *singleton elimination* for single-constructor propositions, and are allowed in some graded modal type theories [Tejiščák 2020; Moon et al. 2021; Abel and Bernardy 2020] but disallowed in others [Atkey 2018; Choudhury et al. 2021, 2022]. Our proof of soundness of extraction does not work for open programs if erased matches are allowed, but we suspect that soundness could be proved also in this case (see Section 7.2).

Following Abel [2018] and Moon et al. [2021] we annotate a Π-type with *two* grades: one is used for terms of that type, whereas the other is used for the codomain of the Π-type. We do not make active use of the feature in this text, but we believe that it could be useful for some type theories where types are relevant for computation, for instance due to the presence of *type-case* [Crary et al. 2002], or for some variants of cubical type theory [Cohen et al. 2015].

We make the following contributions:

(1) We extend the ordered semiring of grades with operations and laws used for the recursor for natural numbers (Section 3), refining previous work that used star-semirings to model recursion [Brunel et al. 2014].
(2) In Section 4 we present a graded modal type theory with large eliminations that allows separate usage tracking for terms and types [Abel 2018; Moon et al. 2021], with a fully formalized meta-theory. To the best of our knowledge, this is the first full formalization of the meta-theory of a graded dependent type theory with a universe and large eliminations.
(3) Section 5 contributes a grading judgement that enjoys substitution and preservation under reduction akin to the work of Wood and Atkey [2021]. The judgement classifies usage of free variables in expressions and is decidable.
(4) In Section 6 we consider the special case of erasability accounting and give a formalized proof of the correctness of erasure using a logical relation.
(5) A discussion of our design choices and alternatives follows in Section 7, in particular concerning the grading of the recursor for natural numbers, but also erased matches and unit types.
(6) In Section 8 we consider an extension by *graded* Σ*-types* along the lines of Atkey [2018] and Choudhury et al. [2021] that allows us to model record types with erased fields. This extension has been formalized as well.

Our formalization [Abel et al. 2023] is a fork of a formalization of decidability of type conversion originally due to Abel et al. [2017]. In total, it consists of around 49000 lines of Agda code (including comments), compared to 15000 lines of code in the original development. Definitions and theorems in this paper link to their formalization in an HTML rendering of the Agda code. Such links are marked in blue. (In some cases where one definition or theorem in the paper corresponds to several pieces of code we have only linked to one of those pieces.)

Note that there are differences between the presentation in the text and the formalization. For one, there is a single formalization with parameters that make it possible to control whether the theory should include things like erased matches for weak Σ-types (Section 7.2), a unit type with η-equality (Section 7.3), graded Σ-types with or without η-equality and with particular grades (Section 8), and one or two modes (Section 8). Another difference is that the formalization uses well-scoped syntax, whereas the text does not. In order to aid readability we do not include side conditions related to things like sizes of contexts in the text.[1]

In the following section, aimed at domain experts, we locate our work more precisely in the context of contemporary research. This section may be skipped at first reading, the technical content starts in Section 3.

[1] The file `README.agda` in the formalization contains more details about the differences. It also contains counterparts to all the links, so if the links no longer work, then one can use `README.agda` instead.

2 RELATION TO THE STATE OF THE ART

Our work is a contribution to an active research field about modalities and type systems. We contribute to a corner of the field where modalities are semirings, or, as a special case for information-flow analyses, lattices of security levels. There are other conceptions of modalities, such as "Fitch-style" multi-modal type theory [Gratzer et al. 2021], where modalities are morphisms between modes, or (idempotent, monadic) modalities in homotopy type theory [Rijke et al. 2020], which are type functors that satisfy certain properties.

In the following we locate our work more precisely in the design/feature space by presenting and motivating some design decisions.

2.1 Form of Judgement: Separate and One-Sided Usage Accounting

Our central judgement is $\boxed{\gamma \blacktriangleright t}$ where t is an expression (term or type) and γ is a *usage context* assigning each free variable in t a grade p. We separate this judgement from the usual typing judgement $\boxed{\Gamma \vdash t : B}$ that expresses that t has type B in typing context Γ which assigns a type to each variable that is in scope for t and B.

Alternatively, we could have considered a single judgement $\boxed{\Phi \vdash t : B}$ where $\Phi = \gamma\Gamma$ is a context mapping variables to pairs pA of their type A and their grade p. This is the dominant conception in the literature [Pfenning 2001; Reed and Pierce 2010; Petricek et al. 2014; Ghica and Smith 2014; Bernardy et al. 2017; Choudhury et al. 2021]. For simple type systems, this view is equivalent to ours, since there in the judgement $\Phi \vdash t : B$ grades only apply to the term t, not to the type B that does not mention any of the variables declared in Φ.

However in dependent type systems, where $\Gamma \vdash t : B$ usually implies $\Gamma \vdash B : \mathsf{Type}$, the separation becomes conceptually important, because $\Phi \vdash t : B$ may often not imply $\Phi \vdash B : \mathsf{Type}$. It does so[2] e.g. in Pfenning's modal type theory of intensionality, extensionality and proof irrelevance [2001], but it does not necessarily do so for quantitative modalities that denote how often a variable is used in an expression.[3]

McBride [2016] pioneered an approach in which terms t and their types B are considered in two different worlds: t in a "material" world where the resources γ for t are consumed by the construction of t, and B in a "mental" world where resources do not matter.[4] Consequently, he introduced a grade 0 for "no consumption" and a judgement we would write as $\Phi \vdash t : 0B$. The latter is essentially bare typing $\Gamma \vdash t : B$ and Φ is there of the form $\mathbf{0}\Gamma$, i.e. resource-free. His typing rules are formulated via a *two-sided* judgement $\boxed{\Phi \vdash t : qB}$, where both the hypotheses and the conclusion are annotated with grades ranging over a "rig" (a kind of semiring), in particular $\{0, 1, \omega\}$ ("none-one-tons").[5] The judgement $\gamma\Gamma \vdash t : qB$ was to mean, paraphrased, that "to produce q copies of the output t we need $\gamma(x)$ copies of each input $(x : A) \in \Gamma$". In that sense, grades may also be called *multiplicities*.

The two-sided form $\Phi \vdash t : qB$ has been considered for information flow tracking [Volpano et al. 1996; Choudhury et al. 2022], where q is the security level of the output and Φ declares the security levels of the inputs. However, as Atkey observed [2018], McBride's calculus failed the substitution property for $q = \omega$ (unrestricted use). Atkey fixed this by giving up the general "q copies" intuition and restricting the right-hand side quantity q to $\{0, 1\}$ where the unit 1 represents the "material", resource-aware world and 0 the "mental", purely logical one.

[2] Pfenning's Lemma 2 states this in a form we would write as $\Phi \vdash t : qB$ but this form is essentially an abbreviation for what we would write as $\Phi/q \vdash t : B$ with a "left-divided" context Φ/q he writes as $\Phi^{\oplus/\ominus}$.

[3] For instance, in $x{:}A \vdash \mathsf{refl}\, x : \mathsf{Eq}\, x\, x$, the term mentions x once while its type does so twice.

[4] McBride uses the words "consumption" and "contemplation" for what we paraphrased here as "material" and "mental".

[5] Quantity 0 for no use ("none"), 1 for a single use ("one"), and ω for unrestricted use ("tons").

Atkey's formulation is equivalent to splitting the judgement into a one-sided one, $\gamma\Gamma \vdash t : B$ for world 1, and a non-resourced one, $\Gamma \vdash t : B$ for world 0. From here, we proceed and separate resourcing/usage into its own judgement $\gamma \blacktriangleright t$. This works for us because we ensure that t contains enough annotations such that this judgement can be defined without reference to types. However, note that some of Atkey's term formers may only be used in the non-resourced setting, for instance the projections for his dependent tensor products. The corresponding β- and η-rules are also only present in the non-resourced setting. Our system does not support this (see also Section 2.3).

In Section 8 we consider a variant $\gamma \blacktriangleright^m t$, which resembles Atkey's two-world accounting. In that setting we support graded Σ-types, which are similar to Atkey's dependent tensors. However, instead of having a single family of types which is "strong" in the non-resourced setting and "weak" otherwise, we have one variant which is always strong and one which is always weak: the first projection for the strong graded Σ-type $\Sigma^q_{\&,0}\, F\, G$ is only available in the erased setting, but the η-rule for this type is always active. We discuss under what conditions η-expansion is resource-preserving for the strong graded Σ-types.

Fu et al. [2022] present a linear dependent type theory for quantum programming with a base type for qubits which is subject to linearity. Their solution for the coexistence of linearity and dependency is different: Expressions occurring in types must belong to the "intuitionistic fragment" of the language, in particular not contain qubits or other entities that must adhere to the linearity discipline. Whenever an expression travels from the term- to the type-side of a judgement, such as in the Π-elimination rule, its linear parts are erased by the Sh(-) operation. The intuition is that Sh(-), stripping the qubits, only leaves the *shape* of a value. Integrating an erasure operation into the typing rules is beyond the capabilities of our system.

2.2 Usage Accounting Also in Types

We take a further step by allowing (but not requiring) an independent usage declaration $\delta \blacktriangleright B$ for the type B of t, rather than ignoring variable usage in B. This has been pioneered by Abel [2018] and Moon et al. [2021]. The latter maintain a triple (Δ, γ, δ) in the typing judgement $\Gamma \vdash t : B$, where γ and δ are the resource vectors of t and B, respectively, and Δ is a triangular resource matrix for Γ, declaring how each hypothesis in Γ depends resource-wise on the previous hypotheses. The full potential of usage accounting in types has not been explored yet, and we do not contribute such applications in this work either. However, Moon et al. presented some experiments on optimizing type-checking based on usage annotations in types.

There is another reason why we account for usage also in types: ignoring usage in types—or at least in terms that stand for types—can be unsound in extensions of type theory with aspects of homotopy type theory or cubical type theory (CTT). Abel et al. [2021] discuss a variant of CTT with support for erasure. They present examples showing that if the rules for erasure are not set up properly, then one can construct two booleans that are provably distinct but equal after erasure. The examples do not rely on CTT, it suffices to have an *erased* univalence axiom [The Univalent Foundations Program 2013].

The usage rules that we present for the Π and Σ type constructors in Section 8 are similar to those presented by Abel et al., with one important difference: our type constructors are annotated with *two* grades instead of one. However, our formalization is parameterized by a relation between those two grades which in particular makes it possible to force them to be equal.

2.3 Semantics Refines Resource-Free Semantics

We maintain the standard definitional equality judgement $\Gamma \vdash t = t' : B$ of type theory. This means that modalities do not interact with equality, limiting the expressiveness of our framework. For instance, it cannot model proof irrelevance [Pfenning 2001; Abel and Scherer 2012; Choudhury et al.

2022] or variance [Steffen 1998; Abel 2006; Stucki and Giarrusso 2021]—the latter is an unsolved problem for dependent types. However, this limitation allows us to employ a standard semantics of dependent types and terms, ignoring grade annotations at first. The resource-aware semantics is then a *refinement* of that first semantics, faithful to our separation of the usage relation $\gamma \blacktriangleright t$ from the typing relation $\Gamma \vdash t : B$. Such a two-phase approach has been employed by Atkey [2018] who first gives a standard *categories with families* (CwF) semantics to the types and terms of his Quantitative Type Theory (QTT), and then refines CwFs to *quantitative CwFs* to prove resource-correctness of terms. In our case, the resource-free semantics is given by a *reducibility* logical relation following Abel et al. [2017], and we refine it for erasure instances by an *erasure* logical relation defined over reducible types.

With simple types, well-typed terms $x_1 : p_1 A_1, \ldots, x_n : p_1 A_n \vdash t : B$ in the one-sided judgement can be interpreted as morphisms $[\![t]\!] : D^{p_1}[\![A_1]\!] \times \cdots \times D^{p_n}[\![A_n]\!] \longrightarrow [\![B]\!]$ for a *graded* comonad D [Ghica and Smith 2014; Petricek et al. 2014]. Thus it is natural to refer to the semiring elements p_i as *grades*. We adopt this terminology even if we do not see how this semantics formally extends to dependent types in the general case.

2.4 Formalization

We have fully formalized our results in Agda [The Agda Team 2023]. A prime motivation for having mechanized proofs is correctness. It has already been mentioned that the substitution property failed in the work of McBride [2016]. With a mechanized formalization one can also build on and change previous work without knowing every detail of every proof: if anything breaks, then one is informed of this (barring any bugs in proof checkers). Our formalization started out as a fork of the formalization originally due to Abel et al. [2017],[6] and our formalization could in turn be the starting point for further work.

2.5 Universes and Large Eliminations

Our development includes a universe and a recursor for natural numbers that can compute values and types, the latter being known as *large elimination*. Thus we ensure that our results are general enough and scale to full-fledged dependent types. This generality eliminates common short-cuts in establishing the meta-theory. For instance, Moon et al. [2021] prove strong normalization via an embedding of reduction into a non-dependent polymorphic language, adapting the work of Geuvers [1994]. Such techniques are not available with large eliminations; we instead pursue the more stony path of dependent logical relations outlined by Abel et al. [2017]. However, this gives us a meta-theory with subject reduction, normalization, and decidability of judgemental equality, via a typed *reducibility* logical relation.

Atkey [2018] presents a more extensional semantics using quantitative CwFs, which also lends itself to large elimination. This semantics is used to show that linear execution (BCI algebras with some extra structure) forms a model of QTT. A reviewer suggested that one could also derive meta-theoretic results like decidability from a suitable CwF, however, as far as we know this has not been spelled out.

2.6 Instance: Erasure, Justified by a Logical Relation

In Section 6 we introduce the *erasure* lattice $\omega \leq 0$ (retained vs. erased) as a possible instance of our generic graded type theory. The same instance can also be understood as the 2-point security lattice $\mathsf{L} \leq \mathsf{H}$ (public vs. private).

[6] Our fork is based on commit bb69ad3f39, which includes further developments, some by Oskar Eriksson, Gaëtan Gilbert and Wojciech Nawrocki.

We demonstrate that erasure of 0-annotated function arguments is sound, i.e., preserves canonical forms at base type, via an "erasure" logical relation indexed by the reducible types.[7]

There is also the syntactic path to correctness of erasure, by showing that reduction is simulated in the target language [Letouzey 2003; Mishra-Linger and Sheard 2008; McBride 2016; Tejiščák 2020]. The syntactic approach does not rely on normalization or semantics and can thus scale to non-terminating languages. However, statements like "if a source term has a value, then the corresponding target term has a value" may not be enough: it could be the case that a source term does not have a value, and still the corresponding target term has a value. In the presence of *erased matches* that can be the case for an open term in a consistent, erasable context (see Section 7.2). We do not solve this problem, but we discuss it.

Choudhury et al. [2021] consider a quantitative dependent type theory GraD (without universes) with a heap semantics. They prove that quantities correctly predict the number of run-time accesses to heap-stored variables by giving a reference-counting reduction semantics. To show that reductions preserve typing, they extend GraD by definitions in the sense of delayed substitutions. It could be worthwhile to investigate whether a (Kripke) logical relation could be utilized for the heap soundness proof of the original GraD, without delayed substitutions.

3 MODALITIES AS ORDERED SEMIRINGS

We begin by giving the definition of modalities as semiring-like structures. The definition largely follows that of Abel and Bernardy [2020] but is also based on McBride's modality for erasure and linearity [2016], its extension by Atkey [2018], and the quantitative coeffect calculus of Brunel et al. [2014], all of which use similar algebraic structures.

Definition 3.1. A modality is a 7-tuple $(\mathcal{M}, +, \cdot, \wedge, \{\circledast_r\}_{r\in\mathcal{M}}, 0, 1)$, consisting of (from left to right) a type $\mathcal{M}$, three binary operations (addition, multiplication, and meet), a collection of binary operators (one for each $r \in \mathcal{M}$), and zero and unit elements, satisfying the following properties:

- $(\mathcal{M}, +, \cdot, 0, 1)$ forms a semiring: Addition is commutative and associative with 0 as identity. Multiplication is associative with 1 as identity and 0 as absorbing element and is distributive over addition. Like McBride [2016] we do *not* require $0 \neq 1$, thus, the unit type is a valid (but trivial) modality.
  As usual, we will typically abbreviate multiplication of p and q as pq.
- $(\mathcal{M}, \wedge)$ forms a semilattice: Meet is commutative, associative, and idempotent. The semilattice induces the usual partial ordering relation: $p \leq q$ iff $p = p \wedge q$.
- For all p, q, and r, $x = p \circledast_r q$ is a solution to the following inequalities:
$$x \leq q + rx \tag{1}$$
$$x \leq p \tag{2}$$
- Both multiplication and addition distribute over meet. Additionally, $\circledast_r$ is sub-distributive over meet, multiplication is right sub-distributive over $\circledast_r$, and addition is sub-interchangeable with $\circledast_r$:
$$\begin{array}{rcl}(p \wedge p') \circledast_r q & \leq & (p \circledast_r q) \wedge (p' \circledast_r q)\\ p \circledast_r (q \wedge q') & \leq & (p \circledast_r q) \wedge (p \circledast_r q')\\ (p \circledast_r p')q & \leq & pq \circledast_r p'q\\ (p \circledast_r q) + (p' \circledast_r q') & \leq & (p + p') \circledast_r (q + q')\end{array}$$

[7] A reviewer pointed out that a similar result could perhaps be obtained for QTT by instantiating its realizability semantics [Atkey 2018, Section 4] so that $!_p x$ is a dummy for $p = 0$.

- Equality is decidable. This property is used in decision procedures for typing and usage as well as to differentiate between 0 and other grades in the erasure case study.

We will typically use $\mathcal{M}$ to refer both to the modality and the underlying type. As we will see, the elements of $\mathcal{M}$ are used to give modal interpretations to programs and in particular to free variables. Which interpretations are available is determined by the specific modality instance that is used.

The operators represent different ways of combining interpretations. Addition combines grades of subterms when both are used, for instance for components of a weak pair, and meet when only one is used, for instance for components of a strong pair or for branches in a case distinction. Multiplication allows us to scale by the number of uses. The operator family $\circledast_r$ has a more specific use and will be used to combine grades for use with the natural number eliminator. For this reason, we will verbalize these operators as "natrec-star".

The modality properties are used for our proofs of the grade analogues of subject reduction and the substitution lemma. For instance, the (sub-) distributivity properties for meet ensure that addition, multiplication, and natrec-star are all monotone operations. For the most part, this definition thus matches the one of Abel and Bernardy [2020] with the main exception being the natrec-star operator, which is used for natrec.

Following Petricek et al. [2014] and Abel and Bernardy [2020] we introduce grade or usage contexts, analogous to typing contexts, that map free variables to grades. For such contexts the operators are lifted to act pointwise in the following way (the ordering relation is also lifted):

$$
\begin{aligned}
(\gamma + \delta)(x) &= \gamma(x) + \delta(x) & \qquad (p \cdot \gamma)(x) &= p \cdot \gamma(x) \\
(\gamma \wedge \delta)(x) &= \gamma(x) \wedge \delta(x) & \qquad (\gamma \circledast_r \delta)(x) &= \gamma(x) \circledast_r \delta(x)
\end{aligned}
$$

Note that the lifting of $\circledast_r$ is still indexed by a single grade, not a context, and the left argument of multiplication is also a single grade. Since the operators are defined pointwise, all properties can similarly be lifted to also hold for contexts. Usage contexts of a given size form a *left semimodule* over the semiring, with the zero context, $\mathbf{0}$, as zero [McBride 2016].

The already mentioned trivial (one element) modality which reduces the system to the underlying non-graded system is a valid instance. A more interesting example is the two-element set $\{0, \omega\}$ with 0 as the additive unit, ω as the multiplicative unit, $\omega + \omega = \omega$, and $\omega \leq 0$. For this instance the only lawful definition of $\circledast_r$ is $\wedge$, independent of r. A possible interpretation of this instance is erasure which is the focus of our case study in Section 6, but the same instance can also be used for information flow in the lattice $\mathsf{L} \leq \mathsf{H}$ (see Section 7.4). For the three-element set $\{0, 1, \omega\}$ with $1 + 1 = 1 + \omega = \omega + \omega = \omega$, 0 as the additive unit, 1 as the multiplicative unit, and $\omega\omega = \omega$ the choice of partial order gives different interpretations. Notably, $\omega \leq 1 \leq 0$ represents affine types whereas $0 \geq \omega \leq 1$ gives linear types. In either case one can show that setting $p \circledast_0 q = p \wedge q$ and $p \circledast_1 q = p + \omega q$ and $p \circledast_\omega q = \omega(p \wedge q)$ gives the largest solution to Eqs. (1) and (2) and the distributivity laws. The affine and linear types interpretations can also be combined using the four-element set $\{0, 1, 01, \omega\}$, where 1 and 01 represent linear and affine uses, respectively. For this instance $p + q = \omega$ unless either p or q is 0, and multiplication is given by $01 \cdot 01 = 01$ and $p \cdot q = \omega$ otherwise, with the obvious exceptions for 0 and 1. The partial order is the reflexive, transitive closure of $\omega \leq 01$ and $1 \geq 01 \leq 0$, and the largest solution to Eqs. (1) and (2) and the distributivity laws are given by $p \circledast_0 q = p \wedge q$ and $p \circledast_1 q = p + \omega q$ and $p \circledast_{01} q = p \wedge \omega q$ and $p \circledast_\omega q = \omega(p \wedge q)$.

Bounded, distributive lattices with decidable equality can be turned into modalities where addition is meet and multiplication is join, the top element is the unit of addition (0), and the bottom element is the unit of multiplication (1). As for the two-point lattice we have that $\circledast_r$ is $\wedge$. Lattices are used in information flow applications, which we discuss further in Section 7.4.

4 TYPE THEORY WITH GRADES

The language we consider, $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}_{\mathcal{M}}$, is an extension of the language originally used in the formalization by Abel et al. [2017]. Its syntax is shown below. The language includes the single (Russell) universe U, natural number type $\mathbb{N}$, and Π-type of the original formalization by Abel et al. [2017], as well as the later additions of an empty type (⊥) and strong Σ-types.[8] This work additionally adds weak Σ-types and graded modalities. Since different instances of the modality $\mathcal{M}$ give different interpretations, $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}_{\mathcal{M}}$ is more accurately seen as a family of languages ranging over the possible instances.

Expressions (terms and types) are given by the following grammar:

$$\begin{aligned} A, B, t, u, v ::= {} & \mathsf{U} \mid \mathbb{N} \mid \bot \mid \Pi^q_p\, A\, B \mid \Sigma^q_{\&}\, A\, B \mid \Sigma^q_{\otimes}\, A\, B \\ & \mid x_i \mid \lambda^p t \mid t\,^p u \mid (t, u)_{\&} \mid \mathsf{fst}\, t \mid \mathsf{snd}\, t \mid (t, u)_{\otimes} \mid \mathsf{prodrec}^q_r\, A\, t\, u \\ & \mid \mathsf{zero} \mid \mathsf{suc}\, t \mid \mathsf{natrec}^q_{p,r}\, A\, t\, u\, v \mid \mathsf{emptyrec}_p\, A\, t \end{aligned}$$

Note that there are two kinds of Σ-types and pair constructors, annotated with & and ⊗ for strong and weak Σ-types, respectively. Often we will not need to differentiate between the two and simply write $\Sigma^q\, A\, B$, or $\Sigma^q_k\, A\, B$, and similarly for pairs. We include two kinds of Σ-types because the types have different characteristics: the strong kind comes with projections and η-equality while the weak one comes with pattern matching (prodrec) and optionally erased matches (and Agda supports all of these variants).

We use de Bruijn-style nameless syntax and write x_i for the variable with index i. The subterms which bind new variables are the second argument B to Π and Σ-types, the body t of lambda abstractions, and the first argument A to natrec and prodrec which each bind one variable, as well as the third arguments u to natrec and prodrec which bind two variables. Since the original formalization [Abel et al. 2017] the formalized language has been updated to be well-scoped by construction, so subterms are guaranteed to not introduce more variables than specified.

Some terms are annotated with grades, indicated by p, q, and r. These are elements from the modality and are used to give programs their desired interpretations based on the chosen instance. The meaning of an annotation varies somewhat between terms and will be detailed in Section 5.

There are three weakening constructors: identity (id), with no effect, shifting (↑), for increasing variable indices by one, and lifting (⇑), which is used when a weakening is pushed under a binder. The application of a weakening ρ to a term t is denoted by $t[\rho]$. For variables we have $x_i[\rho] = x_{i[\rho]}$, with the following definition of $i[\rho]$:

$$i[\mathsf{id}] = i \qquad i[\uparrow\rho] = i[\rho] + 1 \qquad 0[\Uparrow\rho] = 0 \qquad (i+1)[\Uparrow\rho] = i[\rho] + 1$$

For other terms the weakening is applied to each subterm, lifted n times for subterms in which n new variables are bound.

Substitutions are functions from variable indices to terms. The weakening constructors above can be implemented for substitutions, and we overload the notation. We also use the notation $t[\sigma]$ for the application of a substitution σ to a term t. This operation can be implemented similarly to the application of weakenings, with the variable case $x_i[\sigma] = \sigma(i)$. Given a substitution σ and a term t one can form the *extended* substitution σ, t: $(\sigma, t)(0) = t$ and $(\sigma, t)(i + 1) = \sigma(i)$. Note that the substitution (id, u) replaces x_0 with u and reduces the indices of all other free variables by one. To avoid clutter we will often drop the identity substitution. For instance, we can write $t[u]$ instead of $t[\mathsf{id}, u]$. Given a substitution σ we let $\mathsf{head}\,\sigma = \sigma(0)$ and $(\mathsf{tail}\,\sigma)(i) = \sigma(i + 1)$. Note that $\mathsf{head}\,(\sigma, t) = t$ and $\mathsf{tail}\,(\sigma, t) = \sigma$.

[8] The empty type was added to the formalization by Gaëtan Gilbert in 2018. Wojciech Nawrocki added strong Σ-types and a unit type in 2021. The unit type is discussed in Section 7.3.

$$\frac{}{\vdash \epsilon} \qquad \frac{\vdash \Gamma \quad \Gamma \vdash A}{\vdash \Gamma, A} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \mathsf{U}} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \mathbb{N}} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \bot} \qquad \frac{\Gamma \vdash F \quad \Gamma, F \vdash G}{\Gamma \vdash \Pi^q_p \, F \, G}$$

$$\frac{\Gamma \vdash F \quad \Gamma, F \vdash G}{\Gamma \vdash \Sigma^q \, F \, G} \qquad \frac{\Gamma \vdash A : \mathsf{U}}{\Gamma \vdash A} \qquad \frac{}{x_0 : A[\uparrow \mathsf{id}] \in \Gamma, A} \qquad \frac{x_i : A \in \Gamma}{x_{i+1} : A[\uparrow \mathsf{id}] \in \Gamma, B}$$

$$\frac{\Gamma \vdash F : \mathsf{U} \quad \Gamma, F \vdash G : \mathsf{U}}{\Gamma \vdash \Pi^q_p \, F \, G : \mathsf{U}} \qquad \frac{\Gamma \vdash F : \mathsf{U} \quad \Gamma, F \vdash G : \mathsf{U}}{\Gamma \vdash \Sigma^q \, F \, G : \mathsf{U}} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \mathbb{N} : \mathsf{U}} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \bot : \mathsf{U}}$$

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash A = B}{\Gamma \vdash t : B} \qquad \frac{\vdash \Gamma \quad x_i : A \in \Gamma}{\Gamma \vdash x_i : A} \qquad \frac{\Gamma \vdash F \quad \Gamma, F \vdash t : G}{\Gamma \vdash \lambda^p t : \Pi^q_p \, F \, G}$$

$$\frac{\Gamma \vdash t : \Pi^q_p \, F \, G \quad \Gamma \vdash u : F}{\Gamma \vdash t \,^p u : G[u]} \qquad \frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma, F \vdash G \\ \Gamma \vdash t : F \quad \Gamma \vdash u : G[t]\end{array}}{\Gamma \vdash (t, u)_k : \Sigma^q_k \, F \, G} \qquad \frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma, F \vdash G \\ \Gamma \vdash t : \Sigma^q_{\&} \, F \, G\end{array}}{\Gamma \vdash \mathsf{fst}\, t : F}$$

$$\frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma, F \vdash G \\ \Gamma \vdash t : \Sigma^q_{\&} \, F \, G\end{array}}{\Gamma \vdash \mathsf{snd}\, t : G[\mathsf{fst}\, t]} \qquad \frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma, \Sigma^{q'}_{\otimes} \, F \, G \vdash A \\ \Gamma \vdash t : \Sigma^{q'}_{\otimes} \, F \, G \quad \Gamma, F, G \vdash u : A[\uparrow^2 \mathsf{id}, (x_1, x_0)_{\otimes}]\end{array}}{\Gamma \vdash \mathsf{prodrec}^q_r \, A \, t \, u : A[t]} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \mathsf{zero} : \mathbb{N}}$$

$$\frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{suc}\, n : \mathbb{N}} \qquad \frac{\begin{array}{c}\Gamma, \mathbb{N} \vdash A \quad \Gamma \vdash z : A[\mathsf{zero}] \\ \Gamma, \mathbb{N}, A \vdash s : A[\uparrow^2 \mathsf{id}, \mathsf{suc}\, x_1] \quad \Gamma \vdash n : \mathbb{N}\end{array}}{\Gamma \vdash \mathsf{natrec}^q_{p,r} \, A \, z \, s \, n : A[n]} \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash t : \bot}{\Gamma \vdash \mathsf{emptyrec}_p \, A \, t : A}$$

Fig. 1. Well-formed contexts, $\vdash \Gamma$, types, $\Gamma \vdash A$, variables, $x_i : A \in \Gamma$, and terms, $\Gamma \vdash t : A$.

$\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}_{\mathcal{M}}$ is dependently typed with the typical judgements: $\vdash \Gamma$ for well-formed contexts, $\Gamma \vdash A$ for well-formed types, $\Gamma \vdash t : A$ for well-formed terms of type A and $\Gamma \vdash A = B$ and $\Gamma \vdash t = u : A$ for type and term equality. Their inductive definitions are given in Figures 1 and 2. Contexts are lists of types, each giving a type to a free variable. We write ϵ for the empty context and Γ, A for the context Γ extended with A. In that context, x_0 has type $A[\uparrow \mathsf{id}]$ and Γ contains type information for the remaining free variables as expressed by the relation $x_i : B \in \Gamma$ defined in Fig. 1.

For the most part these judgements are unchanged by the addition of grades and are defined as expected. In some rules certain grades are required to match, for instance in the typing rule for lambda abstractions, where the annotation on the lambda is required to match one of the Π-type's annotations. The remaining work of ensuring correctness of grade annotations is instead handled by the usage relation, which is defined in Section 5.

Some rules contain more assumptions than one might expect. For instance, the type equality rule for Π-types contains the assumption $\Gamma \vdash F$, which one might believe could be inferred from the second assumption, $\Gamma \vdash F = H$. In the formalization the extra assumption is used in the proof of a weakening lemma which proceeds by induction on the structure of the derivation. However, one can prove a derived rule without this extra assumption (our proof of this fact indirectly uses the weakening lemma).

$$\frac{\Gamma \vdash A = B : \mathsf{U}}{\Gamma \vdash A = B} \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A = A} \qquad \frac{\Gamma \vdash A = B}{\Gamma \vdash B = A} \qquad \frac{\Gamma \vdash A = B \quad \Gamma \vdash B = C}{\Gamma \vdash A = C}$$

$$\frac{\Gamma \vdash F \quad \Gamma \vdash F = H \quad \Gamma, F \vdash G = E}{\Gamma \vdash \Pi^q_p\, F\, G = \Pi^q_p\, H\, E} \qquad \frac{\Gamma \vdash F \quad \Gamma \vdash F = H \quad \Gamma, F \vdash G = E}{\Gamma \vdash \Sigma^q_k\, F\, G = \Sigma^q_k\, H\, E}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t = t : A} \qquad \frac{\Gamma \vdash t = u : A}{\Gamma \vdash u = t : A} \qquad \frac{\Gamma \vdash t = u : A \quad \Gamma \vdash u = v : A}{\Gamma \vdash t = v : A} \qquad \frac{\Gamma \vdash t = u : A \quad \Gamma \vdash A = B}{\Gamma \vdash t = u : B}$$

$$\frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma \vdash F = H : \mathsf{U}\\ \Gamma, F \vdash G = E : \mathsf{U}\end{array}}{\Gamma \vdash \Pi^q_p\, F\, G = \Pi^q_p\, H\, E : \mathsf{U}} \qquad \frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma \vdash F = H : \mathsf{U}\\ \Gamma, F \vdash G = E : \mathsf{U}\end{array}}{\Gamma \vdash \Sigma^q_k\, F\, G = \Sigma^q_k\, H\, E : \mathsf{U}} \qquad \frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma, F \vdash G\\ \Gamma, F \vdash t : G \quad \Gamma \vdash a : F\end{array}}{\Gamma \vdash (\lambda^p t)\ {}^p a = t[a] : G[a]}$$

$$\frac{\Gamma \vdash f = g : \Pi^q_p\, F\, G \quad \Gamma \vdash a = b : F}{\Gamma \vdash f\ {}^p a = g\ {}^p b : G[a]} \qquad \frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma \vdash f : \Pi^q_p\, F\, G \quad \Gamma \vdash g : \Pi^q_p\, F\, G\\ \Gamma, F \vdash (f[\Uparrow \mathsf{id}])\ {}^p x_0 = (g[\Uparrow \mathsf{id}])\ {}^p x_0 : G\end{array}}{\Gamma \vdash f = g : \Pi^q_p\, F\, G}$$

$$\frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma \vdash t : F\\ \Gamma \vdash u : G[t]\end{array}}{\Gamma \vdash \mathsf{fst}\, (t, u)_{\&} = t : F} \qquad \frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma, F \vdash G\\ \Gamma \vdash t = u : \Sigma^q_{\&}\, F\, G\end{array}}{\Gamma \vdash \mathsf{fst}\, t = \mathsf{fst}\, u : F} \qquad \frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma \vdash t : F\\ \Gamma \vdash u : G[t]\end{array}}{\Gamma \vdash \mathsf{snd}\, (t, u)_{\&} = u : G[\mathsf{fst}\, (t, u)_{\&}]}$$

$$\frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma, F \vdash G\\ \Gamma \vdash t = u : \Sigma^q_{\&}\, F\, G\end{array}}{\Gamma \vdash \mathsf{snd}\, t = \mathsf{snd}\, u : G[\mathsf{fst}\, t]} \qquad \frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma \vdash t : \Sigma^q_{\&}\, F\, G \quad \Gamma \vdash u : \Sigma^q_{\&}\, F\, G\\ \Gamma \vdash \mathsf{fst}\, t = \mathsf{fst}\, u : F \quad \Gamma \vdash \mathsf{snd}\, t = \mathsf{snd}\, u : G[\mathsf{fst}\, t]\end{array}}{\Gamma \vdash t = u : \Sigma^q_{\&}\, F\, G}$$

$$\frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma, F \vdash G\\ \Gamma \vdash t = t' : F \quad \Gamma \vdash u = u' : G[t]\end{array}}{\Gamma \vdash (t, u)_k = (t', u')_k : \Sigma^q_k\, F\, G} \qquad \frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma, \Sigma^{q'}_{\otimes}\, F\, G \vdash A\\ \Gamma \vdash t : F \quad \Gamma \vdash t' : G[t] \quad \Gamma, F, G \vdash u : A[\Uparrow^2 \mathsf{id}, (x_1, x_0)_{\otimes}]\end{array}}{\Gamma \vdash \mathsf{prodrec}^q_r\, A\, (t, t')_{\otimes}\, u = u[t, t'] : A[(t, t')_{\otimes}]}$$

$$\frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma, \Sigma^{q'}_{\otimes}\, F\, G \vdash A = A'\\ \Gamma \vdash t = t' : \Sigma^{q'}_{\otimes}\, F\, G \quad \Gamma, F, G \vdash u = u' : A[\Uparrow^2 \mathsf{id}, (x_1, x_0)_{\otimes}]\end{array}}{\Gamma \vdash \mathsf{prodrec}^q_r\, A\, t\, u = \mathsf{prodrec}^q_r\, A'\, t'\, u' : A[t]} \qquad \frac{\Gamma \vdash m = n : \mathbb{N}}{\Gamma \vdash \mathsf{suc}\, m = \mathsf{suc}\, n : \mathbb{N}}$$

$$\frac{\begin{array}{c}\Gamma, \mathbb{N} \vdash A \quad \Gamma \vdash z : A[\mathsf{zero}]\\ \Gamma, \mathbb{N}, A \vdash s : A[\Uparrow^2 \mathsf{id}, \mathsf{suc}\, x_1]\end{array}}{\Gamma \vdash \mathsf{natrec}^q_{p,r}\, A\, z\, s\, \mathsf{zero} = z : A[\mathsf{zero}]} \qquad \frac{\begin{array}{c}\Gamma, \mathbb{N} \vdash A \quad \Gamma \vdash z : A[\mathsf{zero}]\\ \Gamma, \mathbb{N}, A \vdash s : A[\Uparrow^2 \mathsf{id}, \mathsf{suc}\, x_1] \quad \Gamma \vdash n : \mathbb{N}\end{array}}{\Gamma \vdash \mathsf{natrec}^q_{p,r}\, A\, z\, s\, (\mathsf{suc}\, n) = s[n, \mathsf{natrec}^q_{p,r}\, A\, z\, s\, n] : A[\mathsf{suc}\, n]}$$

$$\frac{\begin{array}{c}\Gamma, \mathbb{N} \vdash A \quad \Gamma, \mathbb{N} \vdash A = A' \quad \Gamma \vdash z = z' : A[\mathsf{zero}]\\ \Gamma, \mathbb{N}, A \vdash s = s' : A[\Uparrow^2 \mathsf{id}, \mathsf{suc}\, x_1] \quad \Gamma \vdash n = n' : \mathbb{N}\end{array}}{\Gamma \vdash \mathsf{natrec}^q_{p,r}\, A\, z\, s\, n = \mathsf{natrec}^q_{p,r}\, A'\, z'\, s'\, n' : A[n]} \qquad \frac{\Gamma \vdash A = B \quad \Gamma \vdash t = u : \bot}{\Gamma \vdash \mathsf{emptyrec}_p\, A\, t = \mathsf{emptyrec}_p\, B\, u : A}$$

Fig. 2. Equality of types, $\Gamma \vdash A = B$, and terms, $\Gamma \vdash t = u : A$.

One can also note the differences between weak and strong Σ-types. Some rules are shared between the two variants, while some only apply to one of them. In particular, the rules involving projections—including the rule for η-equality—are only valid for strong Σ-types. Rules involving prodrec are only given for weak Σ-types (see Remark A.5 for some discussion of this).

We have typeset certain grade and Σ-type annotations in colour to highlight some "non-standard" aspects of the rules.

Remark 4.1 (Atkey's Dependent Tensor Product Types)**.** Our Σ-types differ from the *dependent tensor product types* $(x \overset{p}{:} S) \otimes T$ of Atkey [2018]. In Atkey's case, the grade p is the multiplicity of the first component of the *pair*, whereas in our case this multiplicity is always 1, and the q in $\Sigma_k^q\, F\, G$ is the grade of the first component in the *type* G. Note that Atkey, while including syntax and rules for the dependent tensor product, does not spell out its semantics in the given quantitative CwF model. We describe a variant of our theory with support for something similar to the tensor products of Atkey—inspired by his work—in Section 8.

Remark 4.2 (Eliminator for $\Sigma_\&$, Projections for $\Sigma_\otimes$)**.** One can define a variant of prodrec for strong Σ-types by $\mathsf{prodrec}\; t\; u = u[\mathsf{fst}\; t, \mathsf{snd}\; t]$, and one can derive a typing rule for this construction which is similar to the one for weak Σ-types. However, this construction does not in general satisfy the usage rule that we give for weak Σ-types in Section 5. One can also define projections for weak Σ-types using prodrec, but η-equality does not hold in general for those projections (when they are defined as in our formalization).

The operational semantics of $\lambda_{\mathcal{M}}^{\Sigma\Pi\cup\mathbb{N}}$ is given by call-by-name weak head reduction relations. The reduction relations for types and terms, $\Gamma \vdash A \longrightarrow B$ and $\Gamma \vdash t \longrightarrow u : A$ (see Figure 3), as well as their reflexive transitive closures, $\Gamma \vdash A \longrightarrow^* B$ and $\Gamma \vdash t \longrightarrow^* u : A$ (omitted from the paper), are typed, following Abel et al. [2016, 2017]. Terms that are in weak head normal form (WHNF) do not reduce, and reduction to WHNF is deterministic.

Theorem 4.3. *If $\Gamma \vdash t \longrightarrow^* u : A$ with t in WHNF, then $t = u$. Similarly, if $\Gamma \vdash A \longrightarrow^* B$ with A in WHNF, then $A = B$.*

Theorem 4.4. *If $\Gamma \vdash t \longrightarrow^* u : A$ and $\Gamma \vdash t \longrightarrow^* u' : A'$ with u and u' in WHNF, then $u = u'$. Likewise, if $\Gamma \vdash A \longrightarrow^* B$ and $\Gamma \vdash A \longrightarrow^* B'$ with B and B' in WHNF, then $B = B'$.*

Admissibility of substitution, subject reduction, normalization, and decidability of equality are proved via a *reducibility logical relation* adapted from the work of Abel et al. [2017], see Appendix A. Furthermore type-checking is decidable for a class of expressions which excludes some redexes.

5 ASSIGNING GRADES

We are now ready to explain the purpose of the different annotations and we will do so in parallel with introducing the usage relation which is used to ensure that terms are correctly annotated. Note that because the usage relation is separate from the typing relation—it is a grade assignment system—the logical relation in the previous section can be defined without reference to usage contexts. Also note that, unlike for typing, there is only one relation for both terms and types. Depending on the application, it might thus make sense to require $\delta \blacktriangleright A$, and $\eta_i \blacktriangleright B_i$ for all $x_i : B_i \in \Gamma$ in addition to $\Gamma \vdash t : A$ and $\gamma \blacktriangleright t$ (checking that the type of t and all types in Γ are well-resourced).

Definition 5.1. Given a usage context γ and term t, we say that t is *well-resourced* with respect to γ if $\gamma \blacktriangleright t$. That judgement is defined inductively in Fig. 4.

Similarly to the typing judgements of the previous section, the context is used to keep track of the grades associated with the free variables of a term. Following the study of erasure in Section 6,

$$\frac{\Gamma \vdash A \longrightarrow B : U}{\Gamma \vdash A \longrightarrow B} \qquad \frac{\Gamma \vdash t \longrightarrow u : A \quad \Gamma \vdash A = B}{\Gamma \vdash t \longrightarrow u : B} \qquad \frac{\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma, F \vdash t : G \quad \Gamma \vdash a : F}{\Gamma \vdash (\lambda^p t)\,{}^p a \longrightarrow t[a] : G[a]}$$

$$\frac{\Gamma \vdash t \longrightarrow u : \Pi^q_p\, F\, G \quad \Gamma \vdash a : F}{\Gamma \vdash t\,{}^p a \longrightarrow u\,{}^p a : G[a]} \qquad \frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma \vdash t : F \\ \Gamma \vdash u : G[t] \quad \Gamma \vdash (t, u) : \Sigma^q_{\&}\, F\, G\end{array}}{\Gamma \vdash \mathsf{fst}\,(t, u)_{\&} \longrightarrow t : F}$$

$$\frac{\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma \vdash t \longrightarrow u : \Sigma^q_{\&}\, F\, G}{\Gamma \vdash \mathsf{fst}\, t \longrightarrow \mathsf{fst}\, u : F} \qquad \frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma \vdash t : F \\ \Gamma \vdash u : G[t] \quad \Gamma \vdash (t, u) : \Sigma^q_{\&}\, F\, G\end{array}}{\Gamma \vdash \mathsf{snd}\,(t, u)_{\&} \longrightarrow u : G[\mathsf{fst}\,(t, u)_{\&}]}$$

$$\frac{\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma \vdash t \longrightarrow u : \Sigma^q_{\&}\, F\, G}{\Gamma \vdash \mathsf{snd}\, t \longrightarrow \mathsf{snd}\, u : G[\mathsf{fst}\, t]} \qquad \frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma, \Sigma^{q'}_{\otimes}\, F\, G \vdash A \quad \Gamma \vdash t : F \\ \Gamma \vdash t' : G[t] \quad \Gamma, F, G \vdash u : A[\Uparrow^2 \mathsf{id}, (x_1, x_0)_{\otimes}]\end{array}}{\Gamma \vdash \mathsf{prodrec}^q_r\, A\, (t, t')_{\otimes}\, u \longrightarrow u[t, t'] : A[(t, t')_{\otimes}]}$$

$$\frac{\begin{array}{c}\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma, \Sigma^{q'}_{\otimes}\, F\, G \vdash A \\ \Gamma, F, G \vdash u : A[\Uparrow^2 \mathsf{id}, (x_1, x_0)_{\otimes}] \quad \Gamma \vdash t \longrightarrow t' : \Sigma^{q'}_{\otimes}\, F\, G\end{array}}{\Gamma \vdash \mathsf{prodrec}^q_r\, A\, t\, u \longrightarrow \mathsf{prodrec}^q_r\, A\, t'\, u : A[t]} \qquad \frac{\begin{array}{c}\Gamma, \mathbb{N} \vdash A \quad \Gamma \vdash z : A[\mathsf{zero}] \\ \Gamma, \mathbb{N}, A \vdash s : A[\Uparrow^2 \mathsf{id}, \mathsf{suc}\, x_1]\end{array}}{\Gamma \vdash \mathsf{natrec}^q_{p,r}\, A\, z\, s\, \mathsf{zero} \longrightarrow z : A[\mathsf{zero}]}$$

$$\frac{\Gamma, \mathbb{N} \vdash A \quad \Gamma \vdash z : A[\mathsf{zero}] \quad \Gamma, \mathbb{N}, A \vdash s : A[\Uparrow^2 \mathsf{id}, \mathsf{suc}\, x_1] \quad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{natrec}^q_{p,r}\, A\, z\, s\, (\mathsf{suc}\, n) \longrightarrow s[n, \mathsf{natrec}^q_{p,r}\, A\, z\, s\, n] : A[\mathsf{suc}\, n]}$$

$$\frac{\begin{array}{c}\Gamma, \mathbb{N} \vdash A \quad \Gamma \vdash z : A[\mathsf{zero}] \\ \Gamma, \mathbb{N}, A \vdash s : A[\Uparrow^2 \mathsf{id}, \mathsf{suc}\, x_1] \quad \Gamma \vdash n \longrightarrow n' : \mathbb{N}\end{array}}{\Gamma \vdash \mathsf{natrec}^q_{p,r}\, A\, z\, s\, n \longrightarrow \mathsf{natrec}^q_{p,r}\, A\, z\, s\, n' : A[n]} \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash t \longrightarrow u : \bot}{\Gamma \vdash \mathsf{emptyrec}_p\, A\, t \longrightarrow \mathsf{emptyrec}_p\, A\, u : A}$$

Fig. 3. Weak head reduction of types, $\Gamma \vdash A \longrightarrow B$, and terms, $\Gamma \vdash t \longrightarrow u : A$.

$$\frac{}{\mathbf{0} \blacktriangleright \mathsf{U}} \qquad \frac{}{\mathbf{0} \blacktriangleright \mathbb{N}} \qquad \frac{}{\mathbf{0} \blacktriangleright \bot} \qquad \frac{\gamma \blacktriangleright F \qquad \delta, q \blacktriangleright G}{\gamma + \delta \blacktriangleright \Pi^q_p\, F\, G} \qquad \frac{\gamma \blacktriangleright F \qquad \delta, q \blacktriangleright G}{\gamma + \delta \blacktriangleright \Sigma^q\, F\, G} \qquad \frac{}{\mathbf{e}_i \blacktriangleright x_i}$$

$$\frac{\gamma, p \blacktriangleright t}{\gamma \blacktriangleright \lambda^p t} \qquad \frac{\gamma \blacktriangleright t \qquad \delta \blacktriangleright u}{\gamma + p\delta \blacktriangleright t\,{}^p u} \qquad \frac{\gamma \blacktriangleright t \qquad \delta \blacktriangleright u}{\gamma + \delta \blacktriangleright (t, u)_{\otimes}} \qquad \frac{\gamma \blacktriangleright t \qquad \delta \blacktriangleright u}{\gamma \wedge \delta \blacktriangleright (t, u)_{\&}} \qquad \frac{\gamma \blacktriangleright t}{\gamma \blacktriangleright \mathsf{fst}\, t} \qquad \frac{\gamma \blacktriangleright t}{\gamma \blacktriangleright \mathsf{snd}\, t}$$

$$\frac{\gamma \blacktriangleright t \qquad \delta, r, r \blacktriangleright u \qquad \eta, q \blacktriangleright A}{r\gamma + \delta \blacktriangleright \mathsf{prodrec}^q_r\, A\, t\, u}\ \text{Prodrec}\ r \qquad \frac{}{\mathbf{0} \blacktriangleright \mathsf{zero}} \qquad \frac{\gamma \blacktriangleright t}{\gamma \blacktriangleright \mathsf{suc}\, t}$$

$$\frac{\gamma \blacktriangleright z \qquad \delta, p, r \blacktriangleright s \qquad \eta \blacktriangleright n \qquad \theta, q \blacktriangleright A}{(\gamma \wedge \eta) \circledast_r (\delta + p\eta) \blacktriangleright \mathsf{natrec}^q_{p,r}\, A\, z\, s\, n} \qquad \frac{\gamma \blacktriangleright t \qquad \delta \blacktriangleright A}{p\gamma \blacktriangleright \mathsf{emptyrec}_p\, A\, t} \qquad \frac{\gamma \blacktriangleright t}{\delta \blacktriangleright t}\ \delta \leq \gamma$$

Fig. 4. The grade assignment relation $\gamma \blacktriangleright t$.

we will stick to a quantitative view where the grades are used to keep track of how many times some resource (term) is referenced or *used*. It should be noted, however, that the system allows for non-quantitative interpretations as well (see Section 7.4).

Starting with the simplest cases, the constants, U, $\mathbb{N}$, $\bot$, and zero, are well-resourced with respect to the zero context $\mathbf{0}$. It associates 0 with every variable in scope, none of which appears in a constant. A variable x_i is well-resourced with respect to the unit-vector context $\mathbf{e}_i$ that maps index i to 1 and everything else to 0, representing a single variable occurrence. These rules are not as restrictive as they may first appear since the subsumption rule (on the bottom right) allows us to pass to any pointwise smaller context. Note that the direction of the order relation is different from what one might first expect. For quantitative modalities a smaller grade represents a resource that is allowed to be used *more* times.

For lambda abstractions $\lambda^{p} t$ the annotation p represents how many times the newly bound variable (the function argument) is allowed to be used in the body t. Through the typing rules discussed in the previous section, this is also the meaning of p in the function type $\Pi^{q}_{p}\, F\, G$. The annotation q represents how many times the function argument is allowed to be used by the co-domain G of the function. Analogously, the annotation q on pair types $\Sigma^{q}\, F\, G$ specifies how the first component is used in the type G of the second component. For Σ and Π, as well as for weak pairing $(t, u)_{\otimes}$, the resulting number of uses is given by adding the uses of the subterms. One can imagine a modality interpretation where the two annotations p and q on the Π-type should not be independent of each other, see Section 2.2 for one example. Such cases could be handled by parameterizing typing by a suitable relation between p and q (and the formalization has such a parameter).

Applications $t\,^{p} u$ also add the contexts of the two subterms together but with one crucial difference: the context of the function argument u is first scaled by the annotation p put on the application, which represents the number of times the function argument is allowed to be used. This corresponds to the function argument possibly occurring multiple times once β-reduction has been performed.

In the pairing rule for weak pairs $(t, u)_{\otimes}$ the contexts of the two components are simply added together as described above. When such pairs are deconstructed via prodrec both components are used, and the usage rule for the pair constructor accounts for both components.

The grade annotation r on $\mathrm{prodrec}^{q}_{r}\, t\, u$ stands for the number of times the components of the pair can be used, and the context corresponding to the argument t is scaled accordingly. The annotation q represents how many times the pair can be used in the type argument. Note that while we check that the type argument uses resources correctly, its resources are not included in the conclusion of the rule for prodrec, because type annotations do not contribute to computation. The configurable side condition Prodrec r defines which grades r may be used. If $r = 0$ is admitted (when $0 \neq 1$), then we say that *erased matches* are allowed for (weak) Σ-types. Erased matches are discussed further in Sections 6 and 7.2.

In contrast to weak pairs, for strong pairs $(t, u)_{\&}$ we instead take the *meet* of the two contexts. Strong pairs can only be used through projections, and when a projection is used one component is discarded. The meet of γ and δ is an approximation to both contexts. If, for instance, a variable occurs linearly in one of the components it will only be treated linearly in the pair if it also occurs linearly in the other component (when the linear types modality from Section 3 is used).

The eliminator for the empty type, $\mathrm{emptyrec}_{p}\, A\, t$, is similar to prodrec. However, because uses of this eliminator correspond to dead or impossible branches, we do not restrict the annotation p, allowing the context to be scaled by any amount. In particular, $p = 0$ means that we do not count any resources. A use of this can be found in a safe *head* function for (sized) lists, which takes an erasable proof which shows that the list is non-empty, and uses this proof in an application of $\mathsf{emptyrec}_0$. As for prodrec, we also check the usage in the type argument A.

This leaves only natrec, which is the only term in our language with recursive and branching semantics, both of which give rise to complications. The usage relation has primarily been defined with subject reduction in mind. For terms with (at most) one rule for β-reduction, doing so is straightforward, but for natrec there are two reduction alternatives and we must make sure to include enough resources for either. In addition, the resources consumed by the natural number must also be accounted for.

There are thus three constraints that the context in the conclusion of the rule, call it χ, must satisfy. For the resources of the natural number argument we get $\chi \leq \eta$, and because $\mathsf{natrec}^{q}_{p,r}\,A\,z\,s\,\mathsf{zero}$ reduces to z, we get $\chi \leq \gamma$. In the successor case, $\mathsf{natrec}^{q}_{p,r}\,A\,z\,s\,(\mathsf{suc}\,n)$ reduces to $s[n, \mathsf{natrec}^{q}_{p,r}\,A\,z\,s\,n]$, which uses the resources $\delta + p\eta + r\chi$ (if we assume that a suitable substitution lemma holds), and this gives us the inequality $\chi \leq \delta + p\eta + r\chi$ with χ on both sides.

If the number of iterations, i.e., the value of n, is known, it would be possible to unfold the above definition and arrive at a non-implicit expression for χ, but since this is not always the case we employ a different strategy. A similar problem has been solved by Brunel et al. who include a fixed-point operator and case branching on natural numbers in their calculus [2014]. Translated into our setting, their approach roughly corresponds to setting $\chi = r^*(\eta \wedge \gamma \wedge (\delta + p\eta))$, where the unary star-operator satisfies $r^* = 1 + rr^*$. Using this rule, however, it seems to be difficult to prove subject reduction in general (but see Section 7.1.3).[9]

We have been unable to find a general solution to these inequalities that can be expressed using only addition, multiplication and meet. Instead we include $\circledast_r$, a family of operators, which provides such a solution. The characteristic inequalities, $p \circledast_r q \leq p$ and $p \circledast_r q \leq q + r(p \circledast_r q)$, are motivated by the above reasoning, and allow us to prove subject reduction. In particular, with $\chi = (\gamma \wedge \eta) \circledast_r (\delta + p\eta)$ we have $\chi \leq \gamma$, $\chi \leq \eta$ and $\chi \leq \delta + p\eta + r\chi$ as desired (note that $\gamma \wedge \eta \leq \gamma$). The distributivity and interchange properties specify how $\circledast_r$ relates to the other operators. These are primarily justified by being used in our proof of the substitution lemma and, consequently, subject reduction.

For some modalities there may be some amount of freedom in how to define $\circledast_r$. Since subsumption can always be used, a greatest definition which produces valid solutions to the inequalities is in some sense best, unless smaller solutions are required to achieve some other desired property.

The usage judgement is algorithmic in the sense that it induces an algorithm for computing a usage context recursively from the term. This is possible since all variable-binding subterms have a corresponding annotation giving the newly bound variable its grade. If one can decide whether $\mathsf{Prodrec}\,r$ holds for any r, then it is decidable whether a given term is well-resourced. For details, see Appendix B.

The definition of the usage relation was mainly motivated by a desire to ensure that subject reduction holds, but before we prove that we consider substitutions and a substitution lemma. The usage relation is designed with the substitution lemma in mind as well, in particular the application, prodrec and natrec cases where substitutions correspond to scaling a context. When substitutions act on terms they can be considered as maps between terms that (typically) change the number of free variables, or, alternatively, as maps between typing contexts (under which the terms are well-formed). In the same way, substitutions correspond to mappings between usage contexts and these maps turn out to be linear [Wood and Atkey 2021; Abel and Bernardy 2020], in the

[9] Even if subject reduction would not hold in our system with this rule, this does not mean that the system used by Brunel et al. does not have subject reduction. Our form of natrec is not definable in their system since their rules for case branching correspond to p always being 1 in our system.

sense that they can be represented by matrices that contain grades. By treating usage contexts as vectors for which the i-th component is the grade associated with x_i, substitutions become matrix multiplications.

The key to this is, of course, that the matrix must be chosen in such a way that it accurately represents the substitution. We generalize the usage relation to substitutions, with $\Psi \blacktriangleright \sigma$ meaning that substitution matrix Ψ corresponds to substitution σ.

Definition 5.2. $\Psi \blacktriangleright \sigma$ iff $\forall x_i.\, \mathbf{e}_i\Psi \,{\scriptstyle\blacktriangleright}\, x_i[\sigma]$.

Here $\mathbf{e}_i\Psi$ is the i-th row of Ψ. In other words, a valid substitution matrix should have rows corresponding to valid contexts for each term in the substitution.

The identity substitution corresponds to the identity matrix and a matrix for the single term substitution (id, u) of the kind that occurs during β-reduction can be constructed by appending a context matching u to the bottom of the identity matrix. The substitution lemma for grade usage can now be formulated.

Theorem 5.3. *If $\gamma \,{\scriptstyle\blacktriangleright}\, t$ and $\Psi \blacktriangleright \sigma$ then $\gamma\Psi \,{\scriptstyle\blacktriangleright}\, t[\sigma]$.*

Proof. By induction on $\gamma \,{\scriptstyle\blacktriangleright}\, t$. □

Note that for β-reductions, with $\gamma, p \,{\scriptstyle\blacktriangleright}\, t$ and $\delta \,{\scriptstyle\blacktriangleright}\, u$ and the matrix constructed as above, we get $\gamma + p\delta \,{\scriptstyle\blacktriangleright}\, t[u]$, and $\gamma + p\delta$ is the context that is used for applications in the usage relation. Observations of this kind are key for subject reduction.

Theorem 5.4. *If $\gamma \,{\scriptstyle\blacktriangleright}\, t$ and $\Gamma \vdash t \longrightarrow u : A$ then $\gamma \,{\scriptstyle\blacktriangleright}\, u$.*

Proof. By induction on $\Gamma \vdash t \longrightarrow u : A$ using Theorem 5.3. □

Since valid substitution matrices consist of usage contexts corresponding to substituted terms, context inference can be used to infer a substitution matrix from a substitution, see Appendix B.

6 ERASURE CASE STUDY

In order to show our system in practice, we now turn our attention to a specific use case of $\lambda^{\Sigma\Pi\cup\mathbb{N}}_{\mathcal{M}}$, using a modality for erasure. The goal is to use annotations to keep track of parts of programs that are not used during evaluation and thus can be safely removed. We will perform this erasure while compiling programs to an untyped lambda calculus and finally prove the process to be sound in the sense that it does not affect the outcome of evaluating the program.

We could perform this case study using the erasure modality introduced above. However, the zero can be interpreted as erasure for other modalities as well, if the zero satisfies certain properties:

Definition 6.1. A modality is said to have a well-behaved zero if the following properties hold:

- Addition is positive: if $p + q = 0$, then $p = 0$ and $q = 0$.
- Meet is positive: if $p \wedge q = 0$ then $p = 0$ and $q = 0$.
- The zero-product property holds: if $pq = 0$ then $p = 0$ or $q = 0$.
- The multiplicative zero and unit are distinct, that is, $0 \neq 1$.

The first property is discussed by McBride [2016] and the third one by Atkey [2018]. McBride also discusses the requirement that 0 should be maximal (minimal in his setting): this follows from the second property.

For the remainder of the section, we will work with an arbitrary modality with a well-behaved zero. The modalities for erasure, linear types, affine types and linear or affine types introduced above all have a well-behaved zero.

Though the modality may contain any number of elements, for erasure we only need to keep track of whether a variable is used or not during evaluation. The annotation 0 will be used to represent the latter and any other annotation will be interpreted as the former. For convenience, we will borrow the notation of the erasure modality and denote any non-zero grade as ω. Through subsumption, variables that are not used may be labelled as either used or not (note that $0 \wedge 1 < 0$), giving the programmer a choice of whether they want them to be erased or not.

We first show that our intended interpretation seems reasonable by the following lemma.

Theorem 6.2. *If $\gamma \blacktriangleright x_i$ then $\gamma(x_i) \neq 0$.*

Proof. From $\gamma \blacktriangleright x_i$ we can conclude that $\gamma \leq \mathbf{e}_i$, and in particular $\gamma(x_i) \leq 1$, but for a well-behaved zero $0 \nleq 1$. □

In short, the lemma states that when the usage relation checks a variable its grade must not be 0. This does not necessarily mean, however, that all occurring variables are computationally relevant. As an example of this, consider the polymorphic identity function $\mathsf{id} \coloneqq \lambda^0 \lambda^\omega x_0$ with an erased type argument (using the two-element erasure instance for concreteness). This term has type $\Pi^p_0\ \mathsf{U}\ (\Pi^q_\omega\ x_0\ x_1)$ and is well-resourced with respect to the zero context. In the context $\epsilon, \mathsf{U}, x_0$ the term $(\lambda^0\lambda^\omega x_0)\ {}^0x_1\ {}^\omega x_0$ has type x_1. Since the type argument x_1 is not used during evaluation it should be erasable and indeed, this term is well-resourced with respect to the context $\epsilon, 0, \omega$ (where x_0 is marked as relevant and x_1 as erasable):

$$\dfrac{\dfrac{\epsilon, 0, 0 \blacktriangleright \lambda^0\lambda^\omega x_0 \qquad \dfrac{}{\epsilon, \omega, 0 \blacktriangleright x_1}}{\epsilon, 0, 0 \blacktriangleright (\lambda^0\lambda^\omega x_0)\ {}^0x_1} \qquad \dfrac{}{\epsilon, 0, \omega \blacktriangleright x_0}}{\epsilon, 0, \omega \blacktriangleright (\lambda^0\lambda^\omega x_0)\ {}^0x_1\ {}^\omega x_0}$$

The key is that for erasable applications, any context can be used to check the function argument since it will anyway be scaled by 0. Erasable variables are thus allowed to occur in places such as erasable function arguments.

Since we are now able to track whether a term is computationally relevant or not, we can move on to the task of erasing unneeded parts. As already mentioned, we do this by compiling $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}_{\mathcal{M}}$ to an untyped lambda calculus. The syntax of this target language is the following:

$$v, w ::= x_i \mid \lambda\, t \mid t\ u \mid (t, u) \mid \mathsf{fst}\ t \mid \mathsf{snd}\ t \mid \mathsf{prodrec}\ t\ u \mid \mathsf{zero} \mid \mathsf{suc}\ t \mid \mathsf{natrec}\ t\ u\ v \mid \lightning$$

This language is, except for the lack of types and grades, very similar to $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}_{\mathcal{M}}$, but it also includes $\lightning$, which represents an undefined value. Like $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}_{\mathcal{M}}$, the operational semantics is call-by-name, reducing terms to WHNF.

The compiler is implemented as a function $\_^\bullet$, taking $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}_{\mathcal{M}}$ terms to terms of the target language.

Definition 6.3. The extraction function $\_^\bullet$ is defined by recursion over terms in Figure 5.

Because the structure of the languages is similar, the definition is mostly straightforward but there are two things to note. Firstly, all types are extracted to $\lightning$ since there is no proper representation of types. Secondly, for two terms we make a case distinction on the annotation, namely applications and prodrec. For erasable applications, the argument is simply replaced by $\lightning$. Likewise, for prodrec with an erasable pair, the pair argument is replaced by $(\lightning, \lightning)$.

As an example, consider $\mathsf{id}\ {}^0\mathbb{N}\ {}^\omega\mathsf{zero}$, the identity function applied to $\mathbb{N}$ and zero. This (closed) term has type $\mathbb{N}$ and is well-resourced. As we saw above, the type argument is erasable and indeed, we have $((\lambda^0\lambda^\omega x_0)\ {}^0\mathbb{N}\ {}^\omega\mathsf{zero})^\bullet = (\lambda\,\lambda\, x_0)\ \lightning\ \mathsf{zero}$.

$$\begin{array}{llllll}
\mathsf{U}^\bullet & := \lightning & (\lambda^p\, t)^\bullet & := \lambda\, t^\bullet & \mathsf{zero}^\bullet & := \mathsf{zero} \\
\mathbb{N}^\bullet & := \lightning & (t\ {}^{0}u)^\bullet & := t^\bullet\ \lightning & (\mathsf{suc}\ t)^\bullet & := \mathsf{suc}\ t^\bullet \\
\bot^\bullet & := \lightning & (t\ {}^{\omega}u)^\bullet & := t^\bullet\ u^\bullet & (\mathsf{natrec}^q_{p,r}\ A\ z\ s\ n)^\bullet & := \mathsf{natrec}\ z^\bullet\ s^\bullet\ n^\bullet \\
(\Pi^q_p\ F\ G)^\bullet & := \lightning & (t, u)^\bullet & := (t^\bullet, u^\bullet) & (\mathsf{emptyrec}_p\ A\ t)^\bullet & := \lightning \\
(\Sigma^q\ F\ G)^\bullet & := \lightning & (\mathsf{fst}\ t)^\bullet & := \mathsf{fst}\ t^\bullet & (\mathsf{prodrec}^q_0\ A\ t\ u)^\bullet & := \mathsf{prodrec}\ (\lightning, \lightning)\ u^\bullet \\
x_i^\bullet & := x_i & (\mathsf{snd}\ t)^\bullet & := \mathsf{snd}\ t^\bullet & (\mathsf{prodrec}^q_\omega\ A\ t\ u)^\bullet & := \mathsf{prodrec}\ t^\bullet\ u^\bullet
\end{array}$$

Fig. 5. Extraction/erasure function. Here ω stands for any grade distinct from 0.

If correctly designed, the extraction function should satisfy two main properties: that it erases everything erasable and that it is sound in the sense that it does not affect the outcome of evaluating the program. The first property is straightforward to prove.

Theorem 6.4. *If* $\gamma \blacktriangleright t$ *and* $\gamma(x_i) = 0$ *then* x_i *does not occur in* $t^\bullet$.

Proof. By induction on $\gamma \blacktriangleright t$ using Theorem 6.2. □

For the example we can see that extraction is sound: the term reduces to zero both before and after extraction. In the remainder of this section we shall demonstrate that the extraction function is sound in general, meaning that programs retain their value under erasure.

We could restrict the term *programs* to closed expressions of type $\mathbb{N}$. However, in practical dependently typed programming it can be convenient to work under extra assumptions that are not computationally relevant but can be used to establish correctness properties. For example, one might want to use the law of excluded middle, a choice principle, some other extension of type theory, or simply a correctness conjecture that one intends to prove later. The erasure grades allow us to track which assumptions are not computationally relevant as they only appear in erased parts. Thus, we aim at correct erasure for programs $\Delta_0 \vdash t : \mathbb{N}$ with $\mathbf{0} \blacktriangleright t$, meaning that assumptions from Δ_0 appear in t only in erased positions.

However, we have to be careful so that t does not get stuck through a match, in a non-erased position, on an erased term. In our case, there are two sources of such *erased matches*:

(1) Eliminating the empty type with $\mathsf{emptyrec}_0$. We exclude this by requiring our assumptions to be consistent, i.e., there should be no u such that $\Delta_0 \vdash u : \bot$.
(2) Eliminating an erased weak pair with $\mathsf{prodrec}^q_0$: prodrec does not reduce if applied to a neutral pair. We disallow matching on erased weak pairs in non-erased positions by requiring Prodrec 0 to be false unless Δ_0 is empty (in which case there are no neutral terms to get stuck on).

Some further discussion can be found in Section 7.2.

Our soundness proof uses a logical relation for erasure which we will simply call *the logical relation*. The logical relation relates a term t in the source language with a term v in the target language. It might seem natural to define the relation by induction on the type of t, but dependent types do not directly give us a suitable induction principle. Instead we use a logical relation for reducibility based on the one presented by Abel et al. [2017].

We highlight only the key ideas of the reducibility relation that are relevant here. Further details can be found in Appendix A. An expression A is a reducible type of level ℓ in context Γ, written $\Gamma \Vdash_\ell A$, if, roughly speaking, A reduces to a canonical type with reducible subexpressions or a neutral type. Type families G are reducible if all of their instantiations $G[a]$ with reducible terms a are reducible; this gives us the required induction principle for Π- and Σ-types.

The relation is defined inductively with one case for each type former. We will use labels such as $\langle \mathbb{N} \rangle$ and $\langle \Pi^q_p F G / \mathscr{F} / \mathscr{G} \rangle$ to indicate the different cases. In the latter case, A reduces to $\Pi^q_p\, F\, G$, $\mathscr{F}$ is a proof that F is reducible and $\mathscr{G}(a)$ is a proof that $G[a]$ is reducible for any reducible term a of type F (when we write things like $\mathscr{G}(a)$, a is implicitly required to be a reducible term). In the full definition there are additional requirements for being a reducible Π-type. Additionally, the proofs that F and $G[a]$ are reducible hold for arbitrary weakenings. These differences are omitted here and in the rest of this section because we do not need this extra information and we refer to Appendix A or the formalization for details.

The relation is indexed by the type level ℓ which can be either 0 or 1. In the spirit of Russell universes, reducibility at a lower level can be embedded at a higher level. This is written $\langle \mathrm{emb} / \mathscr{A} \rangle$, where $\mathscr{A}$ is the proof of reducibility at the lower level. Reducibility of terms t of type A, written $\Gamma \Vdash_\ell t : A \,/\, \mathscr{A}$, is defined through induction-recursion by recursion over $\mathscr{A}$, a proof that A is reducible. Fundamental lemmas establish that well-formed types and well-typed terms are reducible. For details, see Appendix A.

The reducibility relation contributes an induction principle which makes it possible to define the erasure relation recursively by cases on the type. In the following, for any judgement J, we will write $\mathscr{A} :: J$ to indicate that $\mathscr{A}$ is a proof of J. For the remainder of this section, we fix a consistent context Δ_0 with $\vdash \Delta_0$.

Definition 6.5. The logical relation for erasure, $t \circledR_\ell v : A \,/\, \mathscr{A}$, where t and A are $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}_{\mathcal{M}}$ terms and v is a target language term, is defined by recursion on $\mathscr{A} :: \Delta_0 \Vdash_\ell A$ as follows:

- If $\mathscr{A} = \langle \mathsf{U} \rangle$ then $t \circledR_\ell v : A \,/\, \mathscr{A}$ iff $\Delta_0 \vdash t : \mathsf{U}$.
- If $\mathscr{A} = \langle \mathbb{N} \rangle$ then $t \circledR_\ell v : A \,/\, \mathscr{A}$ iff $t \circledR_{\mathbb{N}} v$, which is inductively defined to hold iff
  - $\Delta_0 \vdash t \longrightarrow^* \mathsf{zero} : \mathbb{N}$ and $v \longrightarrow^* \mathsf{zero}$, or
  - $\Delta_0 \vdash t \longrightarrow^* \mathsf{suc}\, u : \mathbb{N}$ and $v \longrightarrow^* \mathsf{suc}\, w$ and $u \circledR_{\mathbb{N}} w$.
- If $\mathscr{A} = \langle \Pi^q_\omega F G / \mathscr{F} / \mathscr{G} \rangle$ then $t \circledR_\ell v : A \,/\, \mathscr{A}$ iff $t\,{}^{\omega}a \circledR_\ell v\, w : G[a] \,/\, \mathscr{G}(a)$ for all a and w with $\Delta_0 \Vdash_\ell a : F \,/\, \mathscr{F}$ and $a \circledR_\ell w : F \,/\, \mathscr{F}$.
- If $\mathscr{A} = \langle \Pi^q_0 F G / \mathscr{F} / \mathscr{G} \rangle$ then $t \circledR_\ell v : A \,/\, \mathscr{A}$ iff $t\,{}^{0}a \circledR_\ell v\, \lightning : G[a] \,/\, \mathscr{G}(a)$ for all a with $\Delta_0 \Vdash_\ell a : F \,/\, \mathscr{F}$.
- If $\mathscr{A} = \langle \Sigma^q_k F G / \mathscr{F} / \mathscr{G} \rangle$ then $t \circledR_\ell v : A \,/\, \mathscr{A}$ iff there are terms t_1, t_2, v_1, v_2 such that
  - $\Delta_0 \vdash t \longrightarrow^* (t_1, t_2)_k : \Sigma^q_k\, F\, G$,
  - $v \longrightarrow^* (v_1, v_2)$,
  - $\Delta_0 \Vdash_\ell t_1 : F \,/\, \mathscr{F}$,
  - $t_1 \circledR_\ell v_1 : F \,/\, \mathscr{F}$, and
  - $t_2 \circledR_\ell v_2 : G[t_1] \,/\, \mathscr{G}(t_1)$.
- If $\mathscr{A} = \langle \mathrm{emb} / \mathscr{A}' \rangle$ then $t \circledR_\ell v : A \,/\, \mathscr{A}$ iff $t \circledR_0 v : A \,/\, \mathscr{A}'$.
- If $\mathscr{A} = \langle \bot \rangle$ or $\mathscr{A} = \langle \mathsf{Ne}\, K \rangle$ then $t \circledR_\ell v : A \,/\, \mathscr{A}$ does not hold.

For base types, the idea is that both terms should represent the same value of the right type. In the universe case, t is only required to have type U. Since t is then a type, and there are no types in the target language, there is no requirement on v. The natural number case is defined recursively. Either both terms reduce to zero, or to the successor of some terms which in turn need to be related.

Π^q_p-types are treated differently depending on whether the function argument is erasable or not. Non-erasable applications ($p = \omega$) need only be related when the arguments are related. For erasable functions ($p = 0$) the function argument on the target language side is replaced with $\lightning$ and the relatedness requirement is dropped. This matches the application cases of the extraction function. Σ-types are fairly similar to the base types and no difference is made between weak and strong variants: related terms should reduce to pairs whose components are pairwise related. The embedding case is defined recursively. Finally, for the empty type and neutral types, no terms are in the logical relation. The reason for the former is that the consistency of Δ_0 ensures that there are no terms of type $\bot$. Neutral types are excluded as part of a simplification of only considering non-neutral terms. A consequence of this simplification is that we can only prove the fundamental lemma for empty contexts Δ_0 or when erased matches are disallowed for weak Σ-types (see also Section 7.2).

In order to prove the fundamental lemma we generalize the relation above so that it holds under substitutions to Δ_0. Let us first define a logical relation for substitutions. For this we use another reducibility relation: validity of substitutions. A substitution σ from Γ to Δ is valid, written $\Delta \Vdash^s \sigma : \Gamma \mathbin{/} \mathscr{T}$, if every term in σ is reducible. Here $\mathscr{T}$ is a proof that Γ is valid, and Δ is implicitly required to be well-formed. Validity of contexts, written $\Vdash^v \Gamma$, means that every type in Γ is valid, written $\Gamma \Vdash^v_\ell A \mathbin{/} \mathscr{T}$. This in turn means that A is reducible under any valid substitution. Again we refer to Appendix A for details.

Definition 6.6. The logical relation for substitutions, $\sigma \circledR_\ell \sigma' : \Gamma \blacktriangleleft \gamma \mathbin{/} \mathscr{T} \mathbin{/} \mathscr{S}$, relates substitutions σ and σ' from $\lambda^{\Sigma\Pi\cup\mathbb{N}}_{\mathcal{M}}$ and the target language, respectively, under contexts Γ and γ. It is defined by recursion on $\mathscr{T} :: \Vdash^v \Gamma$, using $\mathscr{S} :: \Delta_0 \Vdash^s \sigma : \Gamma \mathbin{/} \mathscr{T}$, as follows:

- If $\mathscr{T} = \langle \epsilon \rangle$ then $\sigma \circledR_\ell \sigma' : \epsilon \blacktriangleleft \epsilon \mathbin{/} \mathscr{T} \mathbin{/} \mathscr{S}$ holds unconditionally.
- If $\mathscr{T} = \langle \mathscr{T}', \mathscr{A}_{\ell'} \rangle$ then $\sigma \circledR_\ell \sigma' : (\Gamma, A) \blacktriangleleft (\gamma, p) \mathbin{/} \mathscr{T} \mathbin{/} \mathscr{S}$ iff $\mathsf{tail}\,\sigma \circledR_\ell \mathsf{tail}\,\sigma' : \Gamma \blacktriangleleft \gamma \mathbin{/} \mathscr{T}' \mathbin{/} \mathsf{tail}\,\mathscr{S}$ and either $p = 0$, or $p = \omega$ and $\mathsf{head}\,\sigma \circledR_{\ell'} \mathsf{head}\,\sigma' : A[\mathsf{tail}\,\sigma] \mathbin{/} \mathscr{A}(\mathsf{tail}\,\sigma)$.

Similarly to the definition of valid substitutions the idea is to think of substitutions as lists of terms, one for each free variable. These are required to be pairwise related, except for terms corresponding to erasable variables. The motivation for this is related to Theorems 6.2 and 6.4: since erasable variables will not occur in non-erasable settings, and not at all in the target language term, one should be able to substitute anything (of the correct type) for them without consequence.

The final piece of the logical relation is erasure validity, which both expresses the closure of the logical relation under substitutions and our goal that terms should be related to their extracted counterparts.

Definition 6.7. Given $\mathscr{T} :: \Vdash^v \Gamma$ and $\mathscr{A} :: \Gamma \Vdash^v_\ell A \mathbin{/} \mathscr{T}$ the validity relation for erasure, $\gamma \blacktriangleright \Gamma \Vdash_\ell t : A \mathbin{/} \mathscr{T} \mathbin{/} \mathscr{A}$, is defined to hold iff for all substitutions σ, σ' such that $\mathscr{S} :: \Delta_0 \Vdash^s \sigma : \Gamma \mathbin{/} \mathscr{T}$ and $\sigma \circledR_\ell \sigma' : \Gamma \blacktriangleleft \gamma \mathbin{/} \mathscr{T} \mathbin{/} \mathscr{S}$ we have $t[\sigma] \circledR_\ell t^\bullet[\sigma'] : A[\sigma] \mathbin{/} \mathscr{A}(\sigma)$.

Before proving the fundamental lemma we prove a couple of crucial properties. The first expresses that related terms should correspond to the same value in the sense that they reduce to the same weak head normal form. In other words, we may replace a term with any other term in the same reduction chain without leaving the relation.

Theorem 6.8. *If $\mathscr{A} :: \Gamma \Vdash_\ell A$ and $\Delta_0 \vdash t \longrightarrow^* t' : A$ and $v \longrightarrow^* v'$ and $t' \circledR_\ell v' : A \mathbin{/} \mathscr{A}$ then $t \circledR_\ell v : A \mathbin{/} \mathscr{A}$.*

Proof. By induction on $t' \circledR_\ell v' : A \mathbin{/} \mathscr{A}$. □

The second can be seen as a different form of the subsumption rule in the usage relation, allowing us to change the quantity or grade context in the substitution and validity relations.

Theorem 6.9. *If $\mathscr{T} :: \Vdash^v \Gamma$ and $\mathscr{S} :: \Delta_0 \Vdash^s \sigma : \Gamma \mathbin{/} \mathscr{T}$ and $\mathscr{A} :: \Gamma \Vdash^v_\ell A \mathbin{/} \mathscr{T}$, and $\gamma(x_i) = 0$ implies $\delta(x_i) = 0$ for all x_i, then*

- *if $\sigma \circledR_\ell \sigma' : \Gamma \blacktriangleleft \gamma \mathbin{/} \mathscr{T} \mathbin{/} \mathscr{S}$, then $\sigma \circledR_\ell \sigma' : \Gamma \blacktriangleleft \delta \mathbin{/} \mathscr{T} \mathbin{/} \mathscr{S}$, and*
- *if $\delta \blacktriangleright \Gamma \Vdash_\ell t : A \mathbin{/} \mathscr{T} \mathbin{/} \mathscr{A}$ then $\gamma \blacktriangleright \Gamma \Vdash_\ell t : A \mathbin{/} \mathscr{T} \mathbin{/} \mathscr{A}$.*

Proof. For substitutions, by induction on $\mathscr{T}$. The validity case then follows. □

This property is important for our proof of the fundamental lemma since it allows us to re-fit a proof of related substitutions to the correct usage context for each subterm.

Theorem 6.10. *(The fundamental lemma.) If* $\Gamma \vdash t : A$ *and* $\gamma \blacktriangleright t$*, then there are* $\mathscr{T} :: \Vdash^{v} \Gamma$ *and* $\mathscr{A} :: \Gamma \Vdash^{v}_{1} A \mathbin{/} \mathscr{T}$ *such that* $\gamma \blacktriangleright \Gamma \Vdash_{1} t : A \mathbin{/} \mathscr{T} \mathbin{/} \mathscr{A}$*.*

Proof. By induction on $\Gamma \vdash t : A$ using Theorems 6.8, 6.9 and 6.2. □

The fundamental lemma implies that for any related substitutions σ and σ', $t[\sigma]$ is related to $t^{\bullet}[\sigma']$. We would like to apply this result to identity substitutions, but naturally first need to show that these are related. More generally, *all* valid substitutions from erasable contexts are related.

Theorem 6.11. *If* $\mathscr{T} :: \Vdash^{v} \Gamma$ *and* $\mathscr{S} :: \Delta_0 \Vdash^{s} \sigma : \Gamma \mathbin{/} \mathscr{T}$ *then* $\sigma \circledR_{\ell} \sigma' : \Gamma \blacktriangleleft \mathbf{0} \mathbin{/} \mathscr{T} \mathbin{/} \mathscr{S}$ *for all* σ'*.*

Proof. By induction on $\mathscr{T}$. □

Theorem 6.12. *If* $\Delta_0 \vdash t : A$ *and* $\mathbf{0} \blacktriangleright t$ *then there is* $\mathscr{A} :: \Delta_0 \Vdash_1 A$ *such that* $t \circledR_1 t^{\bullet} : A \mathbin{/} \mathscr{A}$*.*

Proof. By the fundamental lemma applied to identity substitutions and Theorem 6.11. □

With the fundamental lemma proven, we can finally move on to showing that the extraction function is sound for programs computing natural numbers. Since terms in both the source and target language only reduce to weak head normal form, the final result of evaluating such a program will either be zero or suc t for some t which might not be in WHNF. In other words, reducing a term to WHNF is not enough to determine whether two terms in WHNF represent the same number or not. To circumvent this we extend the reduction relations of both the source and target languages to allow reduction under suc, see Fig. 6, with reflexive-transitive closures $\longrightarrow^{s*}$ as expected.

$$\frac{\Delta \vdash t \longrightarrow t' : \mathbb{N}}{\Delta \vdash t \longrightarrow^{s} t' : \mathbb{N}} \qquad \frac{\Delta \vdash t \longrightarrow^{s} t' : \mathbb{N}}{\Delta \vdash \mathsf{suc}\, t \longrightarrow^{s} \mathsf{suc}\, t' : \mathbb{N}} \qquad \frac{v \longrightarrow v'}{v \longrightarrow^{s} v'} \qquad \frac{v \longrightarrow^{s} v'}{\mathsf{suc}\, v \longrightarrow^{s} \mathsf{suc}\, v'}$$

Fig. 6. Extended reduction relations, $\Delta \vdash t \longrightarrow^{s} u : \mathbb{N}$ and $t \longrightarrow^{s} u$.

Writing $\underline{n}$ for the numeral suc^{n} zero, where $\mathsf{suc}^{0}\ t$ is t and $\mathsf{suc}^{k+1}\ t$ is suc $(\mathsf{suc}^{k}\ t)$, we give the soundness theorem.

Theorem 6.13. *Let* Δ *be a consistent context. If* Prodrec 0 *is false or* Δ *is empty, and furthermore* $\Delta \vdash t : \mathbb{N}$ *and* $\mathbf{0} \blacktriangleright t$*, then there is a natural number n such that* $\Delta \vdash t \longrightarrow^{s*} \underline{n} : \mathbb{N}$ *and* $t^{\bullet} \longrightarrow^{s*} \underline{n}$*.*

Proof. By Theorem 6.12 we have $t \circledR_1 t^{\bullet} : \mathbb{N} \mathbin{/} \langle \mathbb{N} \rangle$. The proof proceeds by induction on this derivation. □

7 DISCUSSION

In total the formalization, including the erasure case study, the extensions discussed in Sections 7.3 and 8, and various other additions, consists of roughly 39000 lines (2000 kB) of Agda code (comments and empty lines not included). This can be compared to the roughly 13000 lines of code (700 kB) forked from the development of Abel et al. [2017] (including some later extensions).

While some time was spent modifying the Agda development, we believe that extending a pre-existing formalization overall provided great benefit, as it simplified the development process in several ways. For instance, we did not have to define the language from scratch and the code already included proofs of many non-trivial properties. We also found the reducibility relation useful as the starting point for defining the logical relation in the erasure case study.

7.1 Natrec-Star

Formalizing the modality structure and usage relation required relatively little effort, with one main exception. Finding a satisfying and non-restricting way to formulate the usage rule for natrec (including the properties of natrec-star) was non-trivial. Several different options were tested, and ultimately discarded, before we settled on the present solution. However, we review some of our experiments here because they provide methods for defining natrec-star.

7.1.1 Lower-Bounded Structures. In a structure with a global least element, call it ω, the characteristic inequation $x \leq p \wedge (q + rx)$ has the solution $x = \omega$. However, one cannot simply set $p \circledast_r q = \omega$ because this violates the distribution property $(p \circledast_r p')q \leq pq \circledast_r p'q$ for $q = 0$ (unless $0 = \omega$). The refinement $p \circledast_r q := \omega(p \wedge q)$ does satisfy all necessary properties, but it is not in general the *greatest* solution (for instance for the affine and linear types modalities).

7.1.2 Recursive Definition. Since the characteristic inequation has the form of a post-fixed point $x \leq f_{p,q,r}(x)$ with $f_{p,q,r}(x) = p \wedge (q + rx)$, we might try to find a solution for x by iteratively applying $f_{p,q,r}$ to a start value. Concretely, we define the sequence x_n by $x_0 = 0$ and $x_{n+1} = f_{p,q,r}(x_n)$. If for every r there is some n such that, for all p and q, this sequence becomes stationary for n ($x_n = x_{n+1}$), then we can define $p \circledast_r q := x_n$. This satisfies the inequality by definition, and we can prove the distribution and interchange properties by induction on the numbers n. However, in infinite modality structures such a fixed point may not exist, for instance in $\mathcal{M} = \mathbb{N} \cup \{\infty\}$ with standard addition and multiplication and the flipped natural ordering. Yet, when natrec-star is definable this way, it can be a reasonable option. In particular, when 0 is the top element, then we get the greatest solution to $x \leq f_{p,q,r}(x)$.

7.1.3 Bounded Star-Semiring. The recursive inequality invites the introduction of a recursive operation and the previous definition was one attempt at this. It is perhaps not a very natural definition, as it is very tailored to the specific inequalities. As an alternative we can extend the semiring to a star semiring with a unary operator $\_^*$ satisfying $p^* = 1 + pp^* = 1 + p^*p$. (Note that the proofs mentioned below only use $p^* = 1 + pp^*$.)

This alone does not appear to be enough to find a solution in general, but with some further assumptions about the star operator, $p \circledast_r q := r^*(p \wedge q)$ is a candidate. With this definition Equation (1) can easily be shown to hold. If in addition $p^* \leq 0$ or $p^* \leq 1$ hold for all p, then Equation (2) holds.

Unlike the recursive variant, this allows us to find a definition for $\mathbb{N} \cup \{\infty\}$ (with $0^* = 1$ and $p^* = \infty$ for $p \neq 0$). In this case ∞ is a lower bound, so $\infty(p \wedge q)$ also provides a possible definition. These definitions turn out to be equal unless $r = 0$, in which case the star-based operator gives a larger solution. It is also the case in general that the star-based solution is greater than or equal to the one for bounded structures.

7.1.4 Excluding Natrec-Star. Another approach is to have modalities without natrec-star operators and instead use a usage rule for natrec with an inequality as a side condition:

$$\frac{\gamma \blacktriangleright z \qquad \delta, p, r \blacktriangleright s \qquad \eta \blacktriangleright n \qquad \theta, q \blacktriangleright A}{\chi \blacktriangleright \mathsf{natrec}^{q}_{p,r}\ A\ z\ s\ n}\ \chi \leq \gamma \wedge \eta \wedge (\delta + p\eta + r\chi)$$

Our formalization supports this variant of the usage rule, and key results like Theorems 5.4 and 6.13 still hold in this setting.[10] However, our procedure for usage inference uses the natrec-star operators (see Definition B.1).

[10] For the extension with two modes discussed in Section 8 we use an extra assumption, $p + q \leq p$, to prove a key substitution lemma in the setting with the alternative usage rule for natrec. We do not know if this assumption is essential.

7.2 Erased Matches

In our case study on erasure (Section 6) we identified two possible sources of erased matches and imposed restrictions to prevent problematic uses of these. Letouzey [2003] discusses similar issues in the context of extraction for Coq.

The first source of erased matches is $\mathsf{emptyrec}_0\, A\, t$, which allows a term t to be considered unused or erasable in impossible branches. Requiring consistent contexts allows such programs to be written but ensures that any impossible branch will never be executed when the program is run.

Few other works appear to have investigated (elimination of) the empty type in a graded setting. Wood and Atkey [2021] do this for a system with simple types but do not allow this style of treating impossible branches as erased. The usage rule corresponding to their system adds an arbitrary context to that of t instead of scaling with some p as we do.

The other source of erased matches is prodrec, which was restricted to disallow matching on erasable pairs in non-erased settings. Previous work has studied systems both where matching on eliminated pairs [Moon et al. 2021] or forced patterns [Tejiščák 2020] is allowed and systems where it is not [Choudhury et al. 2021], but the consequences of making either choice appear to be less studied.

Our formalization includes the ability to restrict what values the r annotation on prodrec is allowed to take. For erasure, we can thus instantiate our system in two ways, either by allowing $r = 0$ or not, giving us the ability to explore the consequences of the choice.

As an example of the consequences of allowing or disallowing erased matches for prodrec, consider the following, weaker version of our soundness theorem, Theorem 6.13, where only the source language is considered.

Theorem 7.1. *If Γ is consistent, $\Gamma \vdash t : \mathbb{N}$ and $\mathbf{0} \blacktriangleright t$ hold, and* $\mathsf{Prodrec}\, 0$ *is false, then there is a natural number n such that $\Gamma \vdash t \longrightarrow^{s} \underline{n} : \mathbb{N}$.*

If the requirement about $\mathsf{Prodrec}\, 0$ is dropped, then the theorem does not hold. A counterexample is $t := \mathsf{prodrec}^{0}_{0}\, \mathbb{N}\, x_0\, \mathsf{zero}$, which does not reduce to a numeral, but for which we have $\epsilon, \Sigma^{0}_{\otimes}\, \mathbb{N}\, \mathbb{N} \vdash t : \mathbb{N}$ and $\mathbf{0} \blacktriangleright t$, and the context $\epsilon, \Sigma^{0}_{\otimes}\, \mathbb{N}\, \mathbb{N}$ is provably consistent. While this shows that canonicity for natural numbers is lost when erased matches are allowed, it is notable that this is not a counterexample to canonicity for programs compiled to the target language. In this example x_0 is erased and thus compiled to $(\lightning, \lightning)$, which allows the compiled program to evaluate to zero.

Our choice to disallow erased matches for weak Σ-types in the case study was driven by provability of the fundamental lemma. The general proof strategy consists of letting t and $t^{\bullet}$ reduce as much as possible and use Theorem 6.8, but this strategy fails since $\mathsf{prodrec}^{q}_{0}\, A\, t\, u$ might not reduce even though $(\mathsf{prodrec}^{q}_{0}\, A\, t\, u)^{\bullet}$ does. In fact, the fundamental lemma cannot hold if erased matches are allowed for weak Σ-types since there are no neutral elements of type $\mathbb{N}$ in the logical relation.

Although the fundamental lemma does not hold, we conjecture that canonicity for natural numbers does hold for the target language even in the presence of erased matches.

7.3 Unit Type

The difference between the two kinds of pairs might be familiar from linear logic, where the weak and strong pairs correspond to multiplicative and additive conjunction, respectively. Could one also have two distinct unit types, to match the two Σ-types?

A weak or multiplicative unit type might come with the following rules (plus a few more):

$$\frac{}{\mathbf{0} \blacktriangleright \star_{\otimes}} \qquad \frac{\gamma \blacktriangleright t \qquad \delta \blacktriangleright u \qquad \eta, q \blacktriangleright A}{\gamma + \delta \blacktriangleright \mathsf{unitrec}^{q}\, A\, t\, u} \qquad \frac{\Gamma, \top_{\otimes} \vdash A \qquad \Gamma \vdash t : \top_{\otimes} \qquad \Gamma \vdash u : A[\star_{\otimes}]}{\Gamma \vdash \mathsf{unitrec}^{q}\, A\, t\, u : A[t]}$$

The eliminator unitrec extends the pattern of prodrec to the nullary tuple $\star_{\otimes}$.

Conversely, a strong or additive unit type might not include an eliminator but instead allow η-equality and the following usage rule:

$$\frac{}{\gamma \blacktriangleright \star_{\&}} \qquad \frac{\Gamma \vdash t : \top_{\&} \qquad \Gamma \vdash u : \top_{\&}}{\Gamma \vdash t = u : \top_{\&}}.$$

The value $\star_{\&}$ acts as a "sink", allowing a program to "consume" any variable an arbitrary number of times.

Our formalization currently contains $\top_{\&}$, but with the usage rule $\mathbf{0} \blacktriangleright \star_{\&}$, from which the rule above can be derived if 0 is the largest element in $\mathcal{M}$ (because then $\gamma \leq \mathbf{0}$). Under this restriction, which holds for the affine types modality but not the linear one, typed η-expansion $\Gamma \vdash t \longrightarrow \star_{\&} : \top_{\&}$ is usage preserving, and we have proved that there is a type- and resource-preserving procedure that takes a well-typed, well-resourced term to its η-long normal form.

To get the usage rule $\gamma \blacktriangleright \star_{\&}$ and still support usage inference, one could perhaps annotate $\star_{\&}$ with the vector of resources to be consumed. We did not pursue this approach further, as it would require a large change to our formalization.

7.4 Information Flow Interpretation

The erasure lattice $\omega \leq 0$ also serves as a two-point information flow lattice $\mathsf{L} \leq \mathsf{H}$ where publicly available resources are labelled with L and private (confidential) resources are labelled with H. A program t is viewed from the public perspective, and a context γ with $\gamma \blacktriangleright t$ describes how it accesses its resources (its free variables). The law $\mathsf{L}+\mathsf{H} = \mathsf{L}$ expresses that a resource that is accessed both publicly and privately must be publicly available. Our fundamental lemma for the erasure logical relation (Theorem 6.10) implies a form of non-interference: the result of computing a natural number does not depend on the values of H-labelled resources.

In a two-sided formulation $\gamma\Gamma \vdash t : pA$ one can view programs from perspectives p other than $\mathsf{L} = 1$. Depending on what rules are used one may have that a variable x can be accessed if and only if $\gamma(x) \leq p$. For $p = \mathsf{H}$ this means that any variable can be accessed.

We can simulate the two-sided approach by shifting the perspective from 1 to p by *dividing* γ by p. This happens pointwise, so $(\gamma/p)(x) = \gamma(x)/p$. Let us say that a modality *supports division by* q if there is a function $-/q$ such that the Galois connection $\forall p, r.\ p/q \leq r \iff p \leq q \cdot r$ holds. Note that p/q is the least r such that $p \leq q \cdot r$, and that $-/q$ is monotone. The erasure modality supports division by L ($p/\mathsf{L} = p$) and H ($p/\mathsf{H} = \mathsf{L}$): the latter equality entails that all resources are accessible from a private perspective.

This shift of perspective can be carried out in finer information flow lattices. Recall that bounded, distributive lattices with decidable equality can be seen as modalities where 1 is the least element, 0 is the greatest element, addition is meet, and multiplication is join. One example of such a lattice is the total order $1 = \mathsf{L} \leq \mathsf{M} \leq \mathsf{H} = 0$, where we have added a *medium* clearance to the low and high ones. It is straightforward to derive the following division laws in such lattices (the last three are given under the assumption that the modality supports division by p, and for the last one we additionally assume that $p \neq 0$ and that the zero-product property holds for the modality): $p/1 = p$ and $p/0 = 1$ and $p/p = 1$ and $1/p = 1$ and $0/p = 0$. For the lattice $\mathsf{L} \leq \mathsf{M} \leq \mathsf{H}$ we get that $\mathsf{L}/\mathsf{M} = \mathsf{L}$ and $\mathsf{M}/\mathsf{M} = \mathsf{L}$ and $\mathsf{H}/\mathsf{M} = \mathsf{H}$, so shifting the perspective to M makes M-labelled resources available.

8 EXTENSION: MODES AND GRADED Σ-TYPES

Let us now present an extension of the theory with support for *graded Σ-types*, Σ-types where the first component has a grade. The type $\Sigma^{q}_{k,p}\, F\, G$ is a variant of $\Sigma^{q}_{k}\, F\, G$ where the first component, of type F, has grade p. If the erasure modality is used then $\Sigma^q_{k,0}\,F\,G$ is a Σ-type with an erased first component. This kind of type can be used to have an erased component (perhaps a proof of an invariant) in a data structure.

Note that it might not make sense to apply fst to a value of type $\Sigma^0_{\&,0}\,\mathbb{N}\,\mathbb{N}$ (say): if the first component is erased then there is nothing to return. So, in order to support graded Σ-types we follow Atkey [2018] and switch to a usage relation with one of two *modes*: 0_M (erased/compile-time) or 1_M (present/run-time). The idea is that fst is allowed for the graded Σ-type $\Sigma^q_{\&,0}\,F\,G$ only if the mode is 0_M, whereas there are no extra restrictions for $\Sigma^q_{\&,p}\,F\,G$ when $p \neq 0$.

In this section we work with modalities with well-behaved zeros (Definition 6.1), i.e., modalities for which $0 \neq 1$, addition and meet are positive, and the zero-product property holds. We require that $0 \neq 1$ to ensure that the modes 0_M and 1_M correspond to different grades.

As mentioned above, the modalities for erasure, affine types, linear types, and linear or affine types discussed in Section 3 all have well-behaved zeros, and we have proved a subject reduction lemma for arbitrary such modalities (Theorem 8.2). However, note that our main soundness theorem (Theorem 8.3) is focused on erasure. We do not claim that the definitions below make sense in all contexts for all modalities with well-behaved zeros, and we use erasure to motivate the definitions.

Modes can be turned into values of type $\mathcal{M}$: $\overline{0_\mathsf{M}} = 0$ and $\overline{1_\mathsf{M}} = 1$. Values of type $\mathcal{M}$ can also be translated to modes ($\underline{0} = 0_\mathsf{M}$, and $\underline{p} = 1_\mathsf{M}$ if $p \neq 0$), and modes can be scaled by values of type $\mathcal{M}$ ($0_\mathsf{M} \odot p = 0_\mathsf{M}$ and $1_\mathsf{M} \odot p = \underline{p}$).

The syntax is modified a little to support $\Sigma^q_{k,p}\,F\,G$: the projections fst and snd, the pair constructor, and prodrec are all annotated with a grade corresponding to p (in addition to any prior annotations). The type system and reduction relations are mostly unchanged, except to ensure that the p annotations match when appropriate—see the formalization for details. However, the usage relation is modified more drastically. Its definition is inspired by the type system presented by Atkey [2018].

Definition 8.1. Given a usage context γ, a mode m and a term t, we say that t is *well-resourced* with respect to γ and m if $\gamma \blacktriangleright^m t$, where the relation $\blacktriangleright$ is defined inductively in Figure 7.

The variable rule uses the usage context $\overline{m}\mathbf{e}_i$. If m is 1_M, then we get the same context as before, but if m is 0_M, then we get $\overline{0_\mathsf{M}}\mathbf{e}_i = \mathbf{0}$, so the variable is not counted. In the antecedent of the rule for λ-abstractions the grade of variable 0 is $\overline{m}p$: if m is 0_M, then $\overline{m}p = 0$, which matches the variable rule's grade context for 0_M. Multiplication by $\overline{m}$ is used in this way also in other rules.

Note that $m \odot p$ is 0_M exactly when $m = 0_\mathsf{M}$ or $p = 0$. This construction is used when we want to ensure that things are checked in the erased mode 0_M if either the mode is already 0_M, or some argument is erased, for instance for $\Pi^q_0\,F\,G$, $\Sigma^q_{k,0}\,F\,G$ and $t\;{}^0u$.

The usage rules given above are partly based on Agda's rules for erasure [Abel et al. 2021]. Agda supports cubical type theory, which imposes restrictions on what rules one can use for erasure (see Section 2.2). We have tried to be compatible with the rules of Agda. In particular, the rules for Π and Σ are based on those of Agda. In Agda p and q are required to be equal, and the formalization has a parameter which makes it possible to enforce this requirement.

The term $\mathsf{fst}_p\;t$ can only be used if the mode is $m \odot p$, so if $p = 0$, then the mode must be 0_M. There is also a side condition: if $m \odot p$ is 1, then $p \leq 1$ must hold. This condition is used in the proof of Theorem 8.2. Note that for the erasure, affine types, linear types, and linear or affine types modalities, the only grade that is not bounded by 1 is 0.

The first component of a pair of type $\Sigma^q_{k,p}\,F\,G$ is associated with a grade p, and correspondingly the rules for the pair constructors include scaling by p in their conclusions. Scaling by p is also

$$\frac{}{\mathbf{0} \blacktriangleright^{m} \mathsf{U}} \qquad \frac{}{\mathbf{0} \blacktriangleright^{m} \mathbb{N}} \qquad \frac{}{\mathbf{0} \blacktriangleright^{m} \bot} \qquad \frac{\gamma \blacktriangleright^{m \odot p} F \qquad \delta, \overline{m}q \blacktriangleright^{m} G}{\gamma + \delta \blacktriangleright^{m} \Pi^{q}_{p}\, F\, G} \qquad \frac{\gamma \blacktriangleright^{m \odot p} F \qquad \delta, \overline{m}q \blacktriangleright^{m} G}{\gamma + \delta \blacktriangleright^{m} \Sigma^{q}_{k,p}\, F\, G}$$

$$\frac{}{\overline{m}\mathbf{e}_i \blacktriangleright^{m} x_i} \qquad \frac{\gamma, \overline{m}p \blacktriangleright^{m} t}{\gamma \blacktriangleright^{m} \lambda^{p} t} \qquad \frac{\gamma \blacktriangleright^{m} t \qquad \delta \blacktriangleright^{m \odot p} u}{\gamma + p\delta \blacktriangleright^{m} t\,{}^{p} u} \qquad \frac{\gamma \blacktriangleright^{m \odot p} t \qquad \delta \blacktriangleright^{m} u}{p\gamma + \delta \blacktriangleright^{m} (t, u)_{\otimes, p}}$$

$$\frac{\gamma \blacktriangleright^{m \odot p} t \qquad \delta \blacktriangleright^{m} u}{p\gamma \wedge \delta \blacktriangleright^{m} (t, u)_{\&, p}} \qquad \frac{\gamma \blacktriangleright^{m \odot p} t}{\gamma \blacktriangleright^{m \odot p} \mathsf{fst}_{p}\, t}\ m \odot p = 1 \Rightarrow p \leq 1 \qquad \frac{\gamma \blacktriangleright^{m} t}{\gamma \blacktriangleright^{m} \mathsf{snd}_{p}\, t}$$

$$\frac{\gamma \blacktriangleright^{m \odot r} t \qquad \delta, \overline{m}rp, \overline{m}r \blacktriangleright^{m} u \qquad \eta, 0 \blacktriangleright^{0_{\mathsf{M}}} A}{r\gamma + \delta \blacktriangleright^{m} \mathsf{prodrec}^{q}_{r,p}\, A\, t\, u}\ \mathsf{Prodrec}\, r \qquad \frac{}{\mathbf{0} \blacktriangleright^{m} \mathsf{zero}} \qquad \frac{\gamma \blacktriangleright^{m} t}{\gamma \blacktriangleright^{m} \mathsf{suc}\, t}$$

$$\frac{\gamma \blacktriangleright^{m} z \qquad \delta, \overline{m}p, \overline{m}r \blacktriangleright^{m} s \qquad \eta \blacktriangleright^{m} n \qquad \theta, 0 \blacktriangleright^{0_{\mathsf{M}}} A}{(\gamma \wedge \eta) \circledast_{r} (\delta + p\eta) \blacktriangleright^{m} \mathsf{natrec}^{q}_{p,r}\, A\, z\, s\, n} \qquad \frac{\gamma \blacktriangleright^{m \odot p} t \qquad \delta \blacktriangleright^{0_{\mathsf{M}}} A}{p\gamma \blacktriangleright^{m} \mathsf{emptyrec}_{p}\, A\, t} \qquad \frac{\gamma \blacktriangleright^{m} t}{\delta \blacktriangleright^{m} t}\ \delta \leq \gamma$$

Fig. 7. Moded grading $\gamma \blacktriangleright^{m} t$. New variable occurrences are highlighted in colour.

used in the rule for prodrec: note that if p is 0, then there must be no (counted) uses of the variable corresponding to the first component in u.

The rules for prodrec, natrec and emptyrec still include assumptions about the motives ("A"), but now these assumptions use the mode 0_{M}, and in the case of prodrec and natrec the last grade in the usage context is 0: the grade q is no longer used. Note that $\mathbf{0} \blacktriangleright^{0_{\mathsf{M}}} A$ holds exactly when $\mathsf{Prodrec}\, r$ holds for all subterms in A of the form $\mathsf{prodrec}^{q}_{r,p}\, B\, t\, u$.

We can prove a substitution lemma for the theory with graded Σ-types, and using that we obtain the following variant of subject reduction for usage:

Theorem 8.2. *If* $\gamma \blacktriangleright^{m} t$ *and* $\Gamma \vdash t \longrightarrow u : A$ *then* $\gamma \blacktriangleright^{m} u$.

The extraction function from Section 6 is modified for some term formers corresponding to graded Σ-types. Erased first components are erased entirely, and $\mathsf{fst}_0\, t$ is replaced by $\lightning$ (here ω stands for any grade distinct from 0):

$$\begin{array}{llll}
((t, u)_0)^{\bullet} := u^{\bullet} & (\mathsf{fst}_0\, t)^{\bullet} := \lightning & (\mathsf{snd}_0\, t)^{\bullet} & := t^{\bullet} \\
((t, u)_{\omega})^{\bullet} := (t^{\bullet}, u^{\bullet}) & (\mathsf{fst}_{\omega}\, t)^{\bullet} := \mathsf{fst}\, t^{\bullet} & (\mathsf{snd}_{\omega}\, t)^{\bullet} & := \mathsf{snd}\, t^{\bullet} \\
\multicolumn{2}{l}{(\mathsf{prodrec}^{q}_{0,p}\, A\, t\, u)^{\bullet} := \mathsf{prodrec}\, (\lightning, \lightning)\, u^{\bullet}} & (\mathsf{prodrec}^{q}_{\omega,\omega}\, A\, t\, u)^{\bullet} & := \mathsf{prodrec}\, t^{\bullet}\, u^{\bullet} \\
\multicolumn{2}{l}{(\mathsf{prodrec}^{q}_{\omega,0}\, A\, t\, u)^{\bullet} := \mathsf{prodrec}\, (\lightning, t^{\bullet})\, u^{\bullet}} & &
\end{array}$$

With these definitions in place we can state the main soundness theorem for the theory with graded Σ-types (the statement is identical to that of Theorem 6.13). We do not include a proof here but refer to the formalization for details.

Theorem 8.3. *Let* Δ *be a consistent context. If* $\mathsf{Prodrec}\, 0$ *is false or* Δ *is empty, and furthermore* $\Delta \vdash t : \mathbb{N}$ *and* $\mathbf{0} \blacktriangleright t$*, then there is a natural number n such that* $\Delta \vdash t \longrightarrow^{s*} \underline{n} : \mathbb{N}$ *and* $t^{\bullet} \longrightarrow^{s*} \underline{n}$.

Note that the reduction relation does not include η-expansion. Just like in Section 7.3 we can prove that there is a type- and resource-preserving procedure that takes a well-typed, well-resourced term to one of its η-long normal forms. However, there are some restrictions. The linear identity function $\lambda^{1} x_0$ of type $\Pi^{r}_{1}\, (\Sigma^{q}_{\&,p}\, \mathbb{N}\, \mathbb{N})\, (\Sigma^{q}_{\&,p}\, \mathbb{N}\, \mathbb{N})$ is definitionally equal to the η-long normal form

$\lambda^1(\mathsf{fst}_p\ x_0, \mathsf{snd}_p\ x_0)_{\&,p}$, which is well-resourced in the empty context if and only if $p = 1$ or $1 \leq 0 = p$. We have proved that well-typed, well-resourced terms have η-long normal forms if the theory is restricted so that, if $\Sigma^q_{\&,p}$ is allowed, then $p = 1$ or $1 \leq 0 = p$. For the erasure modality these conditions always hold, without any restrictions. For the affine types modality the conditions hold if $\Sigma^q_{\&,\omega}$ is disallowed. And finally the conditions hold for the linear types modality, and the linear or affine types modality, if $\Sigma^q_{\&,p}$ is only allowed when $p = 1$. Note that there are no restrictions for $\Sigma^q_{\otimes,p}$.

The graded Σ-types discussed here are similar to the tensor products presented by Atkey [2018], but there are some differences. Atkey has a single tensor product/Σ-type with a single usage annotation. In his theory the projections fst and snd are allowed only in the erased fragment (when the mode is 0_{M}), whereas a construction similar to prodrec is always allowed (but there is nothing like the "third grade r", and erased matches are not supported). The usual η-equality rule for Σ-types is available for tensor products only in the erased fragment: Atkey does not separate usage from typing, so it is straightforward to restrict an equality rule in this way.

9 CONCLUSIONS

We have presented a dependently typed core language with grades for variable usage, generic over a semiring-like modality structure. The language features Π, weak and strong Σ, a universe, an empty type, and an infinite base type with induction. We have formalized the meta-theory up to normalization and decidability of definitional equality in Agda, including standard results such as substitution lemmata, subject reduction, and consistency, building on the work of Abel et al. [2017]. We further added a usage calculus and showed its correctness via a subject reduction result. As an instance of our generic usage calculus, we studied modalities able to track erasure and proved their soundness with respect to program extraction by a logical relations argument.

Some loose ends that could be investigated in future work:

- Prove that extraction is sound also in the presence of erased matches for weak Σ-types.
- Add an equality (identity) type to the language.
- Add a weak unit type. One may think that such a type could be encoded using $\mathsf{Id}\ \mathbb{N}\ \mathsf{zero}\ \mathsf{zero}$, but experiments in Agda suggest that this might not be possible, depending on what usage rules the identity type comes with, and what usage rules are desired for the unit type.
- Formalize other modality instances, for instance linearity or strictness. A serious soundness argument for linearity might involve a more low-level semantics for the target language, with an explicit heap [Choudhury et al. 2021].
- Add grade inference, following Tejiščák [2020]. A full formalization might require a lot of work, but one could work against an abstract constraint solver whose correctness is stipulated.
- Investigate the interaction of modalities with meta-variables used for type reconstruction in real-world dependently typed languages such as Agda and Idris. Implementations already exist, but theoretical work is still lacking.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers of this and a previous submission for their engaging and valuable feedback. Andreas Abel and Oskar Eriksson acknowledge support by *Vetenskapsrådet* (the Swedish Research Council) via project 2019-04216 *Modal typteori med beroende typer* (Modal Dependent Type Theory).

A A LOGICAL RELATION FOR REDUCIBILITY

In our erasure case study we used the reducibility relation of Abel et al. [2017] (with later additions) to define the logical relation. In this work some, mostly minor, modifications have been made to the relation to account for the addition of grades and weak Σ-types to the language. For completeness we present some details of the reducibility relation here, but we refer to Abel et al. for deeper discussion and motivations. Note that the presentation in this section (like in the rest of the paper) has not been automatically generated from the formalization, and there could be small mistakes in it: the Agda code is the authoritative source for the definitions and theorems.

The definitions we give here are somewhat simplified, with some parts omitted which are used to prove the fundamental lemma of the logical relation, but that we did not need to define the logical relation for erasure. Our presentation also uses the standard equality relations for terms and types. In the formalization the logical relation is parameterized by a collection of equality relations that satisfies a number of properties. We refer to the formalization for full details.

Borrowing the notation of Abel et al., $\Gamma \vdash t :\longrightarrow^*: u : A$ denotes the conjunction of $\Gamma \vdash t : A$, $\Gamma \vdash u : A$ and $\Gamma \vdash t \longrightarrow^* u : A$. Similarly, $\Gamma \vdash A :\longrightarrow^*: B$ denotes the conjunction of $\Gamma \vdash A$, $\Gamma \vdash B$ and $\Gamma \vdash A \longrightarrow^* B$. We also use $\rho : \Delta \supseteq \Gamma$ to denote that ρ is a weakening from context Γ to context Δ.

What we call the reducibility relation actually consists of four relations, $\Gamma \Vdash_\ell A$, $\Gamma \Vdash_\ell t : A \mathbin{/} \mathscr{A}$, $\Gamma \Vdash_\ell A = B \mathbin{/} \mathscr{A}$, and $\Gamma \Vdash_\ell t = u : A \mathbin{/} \mathscr{A}$, for reducible types, terms, equality of reducible types, and equality of reducible terms, respectively. All relations are parameterized by a type level ℓ which can take the values 0 (the level of the single universe) or 1. As with the typing relations above these are defined simultaneously, but we present them one by one, starting with the inductive definition of reducibility of types. The other relations are defined by recursion on $\mathscr{A}$, a proof that the type A is reducible.

In these definitions we make use of some implicit quantifications. For instance, $\forall \rho : \Delta \supseteq \Gamma$ in the Π case of the definition of $\Gamma \Vdash_\ell A$ should be interpreted as quantifying over all contexts Δ, all weakenings ρ from Γ to Δ, as well as a proof of $\rho : \Delta \supseteq \Gamma$.

The notation $\mathscr{A} :: \Gamma \Vdash_\ell A$ is used to indicate that $\mathscr{A}$ is a proof that A is a reducible type, and $\mathrm{Ne}\, t$ means that t is a neutral term (a term that has a variable in a weak head position). If $\mathscr{G}$ is a proof that a type family G is reducible (as, for instance, in the Π case of Definition A.1) we write $\mathscr{G}(\rho, a)$ for the proof that $G[\Uparrow\rho][a]$ is reducible. Note that we leave some requirements, such as a being reducible and the target context of ρ being well-formed, implicit.

Definition A.1. Given a context Γ, reducibility of type A in Γ at type level ℓ is expressed by the relation $\Gamma \Vdash_\ell A$, defined inductively in Figure 8.

As the name suggests the reducibility relation essentially represents terms that reduce to canonical form (or to neutrals). For Π- and Σ-types we also require that the types of the domain and codomain are reducible under any weakening (and for the codomain for all reducible instantiations of the domain). U is only reducible at level 1, and there is one rule for lifting reducibility at level 0 to level 1, but otherwise the rules are applicable at both levels. Because we will define other relations in terms of this one, we will use the labels on the inference rules to refer to the individual cases. Note that some labels also include extra information, like terms or proofs. In the formalization these labels correspond to constructors.

Without going into further details, we now define the remaining three reducibility relations: reducibility of terms, $\Gamma \Vdash_\ell t : A \mathbin{/} \mathscr{A}$, equality of reducible types, $\Gamma \Vdash_\ell A = B \mathbin{/} \mathscr{A}$, and equality of reducible terms, $\Gamma \Vdash_\ell t = u : A \mathbin{/} \mathscr{A}$, all of which are defined by recursion on $\mathscr{A}$, a proof of the reducibility of A.

$$\langle \mathsf{U} \rangle\ \frac{\vdash \Gamma}{\Gamma \Vdash_1 \mathsf{U}} \qquad \langle \mathbb{N} \rangle\ \frac{\Gamma \vdash A :\longrightarrow^*: \mathbb{N}}{\Gamma \Vdash_\ell A} \qquad \langle \bot \rangle\ \frac{\Gamma \vdash A :\longrightarrow^*: \bot}{\Gamma \Vdash_\ell A}$$

$$\langle \mathsf{Ne}\, K \rangle\ \frac{\Gamma \vdash A :\longrightarrow^*: K \qquad \mathsf{Ne}\, K}{\Gamma \Vdash_\ell A} \qquad \langle \mathrm{emb}/\mathscr{A} \rangle\ \frac{\mathscr{A} :: \Gamma \Vdash_0 A}{\Gamma \Vdash_1 A}$$

$$\langle \Pi^q_p FG/\mathscr{F}/\mathscr{G} \rangle\ \frac{\begin{array}{c} \Gamma \vdash A :\longrightarrow^*: \Pi^q_p\, F\, G \\ \Gamma \vdash F \qquad \Gamma, F \vdash G \qquad \mathscr{F} :: (\forall \rho : \Delta \supseteq \Gamma.\ \vdash \Delta \Rightarrow \Delta \Vdash_\ell F[\rho]) \\ \mathscr{G} :: (\forall \rho : \Delta \supseteq \Gamma.\ \vdash \Delta \Rightarrow \Delta \Vdash_\ell a : F[\rho] \ / \ \mathscr{F}(\rho) \Rightarrow \Delta \Vdash_\ell (G[\Uparrow\rho])[a]) \\ \forall \rho : \Delta \supseteq \Gamma. \vdash \Delta \Rightarrow \Delta \Vdash_\ell a : F[\rho] \ / \ \mathscr{F}(\rho) \Rightarrow \Delta \Vdash_\ell b : F[\rho] \ / \ \mathscr{F}(\rho) \Rightarrow \\ \Delta \Vdash_\ell a = b : F[\rho] \ / \ \mathscr{F}(\rho) \Rightarrow \\ \Delta \Vdash_\ell (G[\Uparrow\rho])[a] = (G[\Uparrow\rho])[b] \ / \ \mathscr{G}(\rho, a) \end{array}}{\Gamma \Vdash_\ell A}$$

$$\langle \Sigma^q_k FG/\mathscr{F}/\mathscr{G} \rangle\ \frac{\Gamma \vdash A :\longrightarrow^*: \Sigma^q_k\, F\, G \qquad \textit{The remaining antecedents are equal to those for } \Pi.}{\Gamma \Vdash_\ell A}$$

Fig. 8. Reducibility of types, $\Gamma \Vdash_\ell A$.

Definition A.2. Given a context Γ and $\mathscr{A} :: \Gamma \Vdash_\ell A$, reducibility of term t of type A in Γ at type level ℓ is expressed by the relation $\Gamma \Vdash_\ell t : A \ / \ \mathscr{A}$, which is defined by recursion on $\mathscr{A}$ as follows:

- If $\mathscr{A} = \langle \mathsf{U} \rangle$ then $\Gamma \Vdash_\ell t : A \ / \ \mathscr{A}$ iff $\Gamma \vdash t :\longrightarrow^*: t' : \mathsf{U}$ for some t' that is either a type constructor or a neutral and $\Gamma \Vdash_0 t$.
- If $\mathscr{A} = \langle \mathbb{N} \rangle$ then $\Gamma \Vdash_\ell t : A \ / \ \mathscr{A}$ iff $\Gamma \Vdash_{\mathbb{N}} t$, which is inductively defined to hold if and only if $\Gamma \vdash t :\longrightarrow^*: u : \mathbb{N}$ and
  - $u = \mathsf{zero}$, or
  - $u = \mathsf{suc}\, u'$ and $\Gamma \Vdash_{\mathbb{N}} u'$, or
  - $\mathsf{Ne}\, u$.
- If $\mathscr{A} = \langle \bot \rangle$ then $\Gamma \Vdash_\ell t : A \ / \ \mathscr{A}$ iff $\Gamma \vdash t :\longrightarrow^*: u : \bot$ and $\mathsf{Ne}\, u$.
- If $\mathscr{A} = \langle \mathsf{Ne}\, K \rangle$ then $\Gamma \Vdash_\ell t : A \ / \ \mathscr{A}$ iff $\Gamma \vdash t :\longrightarrow^*: u : K$ and $\mathsf{Ne}\, u$.
- If $\mathscr{A} = \langle \Pi^q_p FG/\mathscr{F}/\mathscr{G} \rangle$ then $\Gamma \Vdash_\ell t : A \ / \ \mathscr{A}$ iff
  - $\Gamma \vdash t :\longrightarrow^*: u : \Pi^q_p\, F\, G$ for $u = \lambda^{p'} u'$ or $\mathsf{Ne}\, u$,
  - for all $\rho : \Delta \supseteq \Gamma$ and $\vdash \Delta$ and $\Delta \Vdash_\ell a : F[\rho] \ / \ \mathscr{F}(\rho)$ and $\Delta \Vdash_\ell b : F[\rho] \ / \ \mathscr{F}(\rho)$ and $\Delta \Vdash_\ell a = b : F[\rho] \ / \ \mathscr{F}(\rho)$ we have $\Delta \Vdash_\ell u[\rho]\ {}^{p}a = u[\rho]\ {}^{p}b : (G[\Uparrow\rho])[a] \ / \ \mathscr{G}(\rho, a)$, and
  - for all $\rho : \Delta \supseteq \Gamma$ and $\vdash \Delta$ and $\Delta \Vdash_\ell a : F[\rho] \ / \ \mathscr{F}(\rho)$ we have $\Delta \Vdash_\ell u[\rho]\ {}^{p}a : (G[\Uparrow\rho])[a] \ / \ \mathscr{G}(\rho, a)$.
- If $\mathscr{A} = \langle \Sigma^q_{\&} FG/\mathscr{F}/\mathscr{G} \rangle$ then $\Gamma \Vdash_\ell t : A \ / \ \mathscr{A}$ iff $\Gamma \vdash t :\longrightarrow^*: u : \Sigma^q_{\otimes}\, F\, G$ such that
  - either $\mathsf{Ne}\, u$ or $u = (u_1, u_2)$,
  - $\Gamma \Vdash_\ell \mathsf{fst}\, u : F[\mathrm{id}] \ / \ \mathscr{F}(\mathrm{id})$, and
  - $\Gamma \Vdash_\ell \mathsf{snd}\, u : (G[\Uparrow\mathrm{id}])[\mathsf{fst}\, u] \ / \ \mathscr{G}(\Uparrow\mathrm{id}, \mathsf{fst}\, u)$.
- If $\mathscr{A} = \langle \Sigma^q_{\otimes} FG/\mathscr{F}/\mathscr{G} \rangle$ then $\Gamma \Vdash_\ell t : A \ / \ \mathscr{A}$ iff $\Gamma \vdash t :\longrightarrow^*: u : \Sigma^q_{\otimes}\, F\, G$ such that
  - either $\mathsf{Ne}\, u$ or $u = (u_1, u_2)$,
  - $\Gamma \Vdash_\ell u_1 : F[\mathrm{id}] \ / \ \mathscr{F}(\mathrm{id})$ and
  - $\Gamma \Vdash_\ell u_2 : (G[\Uparrow\mathrm{id}])[u_1] \ / \ \mathscr{G}(\Uparrow\mathrm{id}, u_1)$.
- If $\mathscr{A} = \langle \mathrm{emb}/\mathscr{A}' \rangle$ then $\Gamma \Vdash_\ell t : A \ / \ \mathscr{A}$ iff $\Gamma \Vdash_0 t : A \ / \ \mathscr{A}'$.

Definition A.3. Given a context Γ and $\mathscr{A} :: \Gamma \Vdash_\ell A$, equality of reducible types A and B in Γ at type level ℓ is expressed by the relation $\Gamma \Vdash_\ell A = B \mathrel{/} \mathscr{A}$, which is defined by recursion on $\mathscr{A}$ as follows:

- If $\mathscr{A} = \langle \mathsf{U} \rangle$ then $\Gamma \Vdash_\ell A = B \mathrel{/} \mathscr{A}$ iff $B = \mathsf{U}$.
- If $\mathscr{A} = \langle \mathbb{N} \rangle$ then $\Gamma \Vdash_\ell A = B \mathrel{/} \mathscr{A}$ iff $\Gamma \vdash B \longrightarrow^* \mathbb{N}$.
- If $\mathscr{A} = \langle \bot \rangle$ then $\Gamma \Vdash_\ell A = B \mathrel{/} \mathscr{A}$ iff $\Gamma \vdash B \longrightarrow^* \bot$.
- If $\mathscr{A} = \langle \mathsf{Ne}\, K \rangle$ then $\Gamma \Vdash_\ell A = B \mathrel{/} \mathscr{A}$ iff $\Gamma \vdash B :\longrightarrow^*: M$ for some $\mathsf{Ne}\, M$ with $\Gamma \vdash K = M : \mathsf{U}$.
- If $\mathscr{A} = \langle \Pi^q_p FG / \mathscr{F} / \mathscr{G} \rangle$ then $\Gamma \Vdash_\ell A = B \mathrel{/} \mathscr{A}$ iff there are F', G' such that
  - $\Gamma \vdash B \longrightarrow^* \Pi^q_p\, F'\, G'$,
  - $\Gamma \vdash \Pi^q_p\, F\, G = \Pi^q_p\, F'\, G'$,
  - for all $\rho : \Delta \supseteq \Gamma$ and $\vdash \Delta$ we have $\Delta \Vdash_\ell F[\rho] = F'[\rho] \mathrel{/} \mathscr{F}(\rho)$, and
  - for all $\rho : \Delta \supseteq \Gamma$ and $\vdash \Delta$ and $\Delta \Vdash_\ell a : F[\rho] \mathrel{/} \mathscr{F}(\rho)$ we have $\Delta \Vdash_\ell (G[\Uparrow\rho])[a] = (G'[\Uparrow\rho])[a] \mathrel{/} \mathscr{G}(\rho, a)$.
- If $\mathscr{A} = \langle \Sigma^q_k FG / \mathscr{F} / \mathscr{G} \rangle$ then $\Gamma \Vdash_\ell A = B \mathrel{/} \mathscr{A}$ iff there are F', G' such that
  - $\Gamma \vdash B \longrightarrow^* \Sigma^q_k\, F'\, G'$,
  - $\Gamma \vdash \Sigma^q_k\, F\, G = \Sigma^q_k\, F'\, G'$,
  - for all $\rho : \Delta \supseteq \Gamma$ and $\vdash \Delta$ we have $\Delta \Vdash_\ell F[\rho] = F'[\rho] \mathrel{/} \mathscr{F}(\rho)$, and
  - for all $\rho : \Delta \supseteq \Gamma$ and $\vdash \Delta$ and $\Delta \Vdash_\ell a : F[\rho] \mathrel{/} \mathscr{F}(\rho)$ we have $\Delta \Vdash_\ell (G[\Uparrow\rho])[a] = (G'[\Uparrow\rho])[a] \mathrel{/} \mathscr{G}(\rho, a)$.
- If $\mathscr{A} = \langle \mathrm{emb} / \mathscr{A}' \rangle$ then $\Gamma \Vdash_\ell A = B \mathrel{/} \mathscr{A}$ iff $\Gamma \Vdash_0 A = B \mathrel{/} \mathscr{A}'$.

Definition A.4. Given a context Γ and $\mathscr{A} :: \Gamma \Vdash_\ell A$, equality of reducible terms t and u of type A in Γ at type level ℓ is expressed by the relation $\Gamma \Vdash_\ell t = u : A \mathrel{/} \mathscr{A}$, which is defined by recursion on $\mathscr{A}$ as follows:

- If $\mathscr{A} = \langle \mathsf{U} \rangle$ then $\Gamma \Vdash_\ell t = u : A \mathrel{/} \mathscr{A}$ iff there are terms t', u' such that
  - $\Gamma \vdash t :\longrightarrow^*: t' : \mathsf{U}$,
  - $\Gamma \vdash u :\longrightarrow^*: u' : \mathsf{U}$,
  - t' and u' are either type constructors or neutrals,
  - $\Gamma \vdash t' = u' : \mathsf{U}$,
  - $\mathscr{T} :: \Gamma \Vdash_\ell t$,
  - $\Gamma \Vdash_\ell u$, and
  - $\Gamma \Vdash_\ell t = u \mathrel{/} \mathscr{T}$.
- If $\mathscr{A} = \langle \mathbb{N} \rangle$ then $\Gamma \Vdash_\ell t = u : A \mathrel{/} \mathscr{A}$ iff $\Gamma \Vdash_{\mathbb{N}} t = u$, which is inductively defined to hold iff there are terms t_1 and u_1 such that
  - $\Gamma \vdash t :\longrightarrow^*: t_1 : \mathbb{N}$,
  - $\Gamma \vdash u :\longrightarrow^*: u_1 : \mathbb{N}$,
  - $\Gamma \vdash t_1 = u_1 : \mathbb{N}$, and either
    - $t_1 = u_1 = \mathsf{zero}$,
    - $t_1 = \mathsf{suc}\, t_2$ and $u_1 = \mathsf{suc}\, u_2$ and $\Gamma \Vdash_{\mathbb{N}} t_2 = u_2$, or
    - $\mathsf{Ne}\, t_1$ and $\mathsf{Ne}\, u_1$.
- If $\mathscr{A} = \langle \bot \rangle$ then $\Gamma \Vdash_\ell t = u : A \mathrel{/} \mathscr{A}$ iff there are terms t', u' such that
  - $\Gamma \vdash t :\longrightarrow^*: t' : \bot$,
  - $\Gamma \vdash u :\longrightarrow^*: u' : \bot$,
  - $\Gamma \vdash t' = u' : \bot$, and
  - $\mathsf{Ne}\, t'$ and $\mathsf{Ne}\, u'$.
- If $\mathscr{A} = \langle \mathsf{Ne}\, K \rangle$ then $\Gamma \Vdash_\ell t = u : A \mathrel{/} \mathscr{A}$ iff there are terms t', u' such that
  - $\Gamma \vdash t :\longrightarrow^*: t' : K$,

  - $\Gamma \vdash u :\longrightarrow^*: u' : K$,
  - $\Gamma \vdash t' = u' : K$, and
  - $\mathsf{Ne}\, t'$ and $\mathsf{Ne}\, u'$.
- If $\mathscr{A} = \langle \Pi^q_p F G / \mathscr{F} / \mathscr{G} \rangle$ then $\Gamma \Vdash_\ell t = u : A \,/\, \mathscr{A}$ iff there are terms t', u' such that
  - $\Gamma \vdash t :\longrightarrow^*: t' : \Pi^q_p\, F\, G$,
  - $\Gamma \vdash u :\longrightarrow^*: u' : \Pi^q_p\, F\, G$,
  - t' and u' are either lambda abstractions or neutrals,
  - $\Gamma \vdash t' = u' : \Pi^q_p\, F\, G$,
  - $\Gamma \Vdash_\ell t : A \,/\, \mathscr{A}$,
  - $\Gamma \Vdash_\ell u : A \,/\, \mathscr{A}$, and
  - for all $\rho : \Delta \supseteq \Gamma$ and $\vdash \Delta$ and $\Delta \Vdash_\ell a : F[\rho] \,/\, \mathscr{F}(\rho)$ we have $\Delta \Vdash_\ell t'[\rho]\ {}^{p}a = u'[\rho]\ {}^{p}a : (G[\Uparrow\rho])[a] \,/\, \mathscr{G}(\rho, a)$.
- If $\mathscr{A} = \langle \Sigma^q_{\&} F G / \mathscr{F} / \mathscr{G} \rangle$ then $\Gamma \Vdash_\ell t = u : A \,/\, \mathscr{A}$ iff there are terms t', u' such that
  - $\Gamma \vdash t :\longrightarrow^*: t' : \Sigma^q_{\&}\, F\, G$,
  - $\Gamma \vdash u :\longrightarrow^*: u' : \Sigma^q_{\&}\, F\, G$,
  - t' and u' are either pairs or neutrals,
  - $\Gamma \vdash t' = u' : \Sigma^q_{\&}\, F\, G$,
  - $\Gamma \Vdash_\ell t : A \,/\, \mathscr{A}$,
  - $\Gamma \Vdash_\ell u : A \,/\, \mathscr{A}$,
  - $\Gamma \Vdash_\ell \mathsf{fst}\, t' : F[\mathsf{id}] \,/\, \mathscr{F}(\mathsf{id})$,
  - $\Gamma \Vdash_\ell \mathsf{fst}\, u' : F[\mathsf{id}] \,/\, \mathscr{F}(\mathsf{id})$,
  - $\Gamma \Vdash_\ell \mathsf{fst}\, t' = \mathsf{fst}\, u' : F[\mathsf{id}] \,/\, \mathscr{F}(\mathsf{id})$, and
  - $\Gamma \Vdash_\ell \mathsf{snd}\, t' = \mathsf{snd}\, u' : (G[\Uparrow\mathsf{id}])[\mathsf{fst}\, t'] \,/\, \mathscr{G}(\Uparrow\mathsf{id}, \mathsf{fst}\, t')$.
- If $\mathscr{A} = \langle \Sigma^q_{\otimes} F G / \mathscr{F} / \mathscr{G} \rangle$ then $\Gamma \Vdash_\ell t = u : A \,/\, \mathscr{A}$ iff there are terms t', u' such that
  - $\Gamma \vdash t :\longrightarrow^*: t' : \Sigma^q_{\otimes}\, F\, G$,
  - $\Gamma \vdash u :\longrightarrow^*: u' : \Sigma^q_{\otimes}\, F\, G$,
  - $\Gamma \vdash t' = u' : \Sigma^q_{\otimes}\, F\, G$,
  - $\Gamma \Vdash_\ell t : A \,/\, \mathscr{A}$,
  - $\Gamma \Vdash_\ell u : A \,/\, \mathscr{A}$, and
  - either $\mathsf{Ne}\, t'$ and $\mathsf{Ne}\, u'$, or
    * $t' = (t_1, t_2)$, $u' = (u_1, u_2)$,
    * $\Gamma \Vdash_\ell t_1 : F[\mathsf{id}] \,/\, \mathscr{F}(\mathsf{id})$,
    * $\Gamma \Vdash_\ell u_1 : F[\mathsf{id}] \,/\, \mathscr{F}(\mathsf{id})$,
    * $\Gamma \Vdash_\ell t_2 : (G[\Uparrow\mathsf{id}])[t_1] \,/\, \mathscr{G}(\Uparrow\mathsf{id}, t_1)$,
    * $\Gamma \Vdash_\ell u_2 : (G[\Uparrow\mathsf{id}])[u_1] \,/\, \mathscr{G}(\Uparrow\mathsf{id}, u_1)$,
    * $\Gamma \Vdash_\ell t_1 = u_1 : F[\mathsf{id}] \,/\, \mathscr{F}(\mathsf{id})$, and
    * $\Gamma \Vdash_\ell t_2 = u_2 : (G[\Uparrow\mathsf{id}])[t_1] \,/\, \mathscr{G}(\Uparrow\mathsf{id}, t_1)$.
- If $\mathscr{A} = \langle \mathsf{emb} / \mathscr{A}' \rangle$ then $\Gamma \Vdash_\ell t = u : A \,/\, \mathscr{A}$ iff $\Gamma \Vdash_0 t = u : A \,/\, \mathscr{A}'$.

With regards to modalities there are not any major changes to the relations. The only requirements we have added are that some grade annotations must match (similarly to the changes to the typing relations).

For Σ-types themselves no distinction is made between the weak and strong variants, but the definitions differ for terms of Σ-type. In both cases the terms should reduce to WHNFs. For strong Σ-types we require that their components as given by the projections are reducible (and, for equality, equal). In the weak case the WHNFs must either be pairs, in which case the components are required to be reducible (and possibly equal), or neutral, in which case we require nothing else. For equality there is no case for one neutral term and one term that reduces to a pair, because such terms would never be equal.

Remark A.5 (Weak and Strong Σ-types in the Logical Relation)**.** The eliminator for Σ-types is restricted to weak products. One reason for this is that it does not appear to be possible to prove the prodrec case of the fundamental lemma for the logical relation for terms of type $\Sigma_\&$. The proof strategy used for $\Sigma_\otimes$ is to use the fact that for non-neutral, reducible t we have $\Gamma \vdash t \longrightarrow^* (t_1, t_2)_\otimes : \Sigma^q_\otimes F\, G$ and thus $\Gamma \vdash \mathsf{prodrec}^q_r\, A\, t\, u \longrightarrow^* u[t_1, t_2] : A[t]$. It can then be proved that $u[t_1, t_2]$ is reducible, and reducibility of $\mathsf{prodrec}^q_r\, A\, t\, u$ then follows. For strong Σ-types, this strategy does not work since we only get that $\mathsf{fst}\, t$ and $\mathsf{snd}\, t$ are reducible. One could allow prodrec to be used for both weak and strong Σ-types by letting terms of the two variants have the same definition of reducibility (the one we currently have for $\Sigma_\otimes$). This would, in turn, lead to problems with the η-equality case of the fundamental lemma. One could choose to discard η-equality for strong Σ-types but doing so would obviously be a rather large change of the system.

It is not possible to directly prove the fundamental lemma for this logical relation, because the inductive hypothesis is too weak to handle binding subterms. Instead Abel et al. introduce another set of relations, for *valid* terms, types, contexts and substitutions, and equality of those things [2017]. We have not needed to make any modifications to these relations but made use of them in the erasure case study, so we will briefly introduce parts of them. The basic idea is to make the reducibility relation closed under valid substitutions. The fundamental lemma for reducibility then becomes a special case of the fundamental lemma for validity, applied to the identity substitution.

We begin with validity of contexts. Similarly to reducibility of types, this will serve as the basis of the relation, with the other relations defined by recursion on this one. Again, some of these definitions are made simultaneously, see the formalization for details.

Definition A.6. Validity of contexts, $\Vdash^v \Gamma$, is defined inductively as follows:

$$\langle \epsilon \rangle \ \frac{}{\Vdash^v \epsilon} \qquad\qquad \langle \mathscr{T}, \mathscr{A}_\ell \rangle \ \frac{\mathscr{T} :: \Vdash^v \Gamma \qquad \mathscr{A} :: \Gamma \Vdash^v_\ell A \,/\, \mathscr{T}}{\Vdash^v \Gamma, A}$$

We can then define validity of substitutions and of equal substitutions.

Definition A.7. Validity of substitutions, $\Delta \Vdash^s \sigma : \Gamma \,/\, \mathscr{T}$, is defined by recursion on $\mathscr{T} :: \Vdash^v \Gamma$ as follows:

- If $\mathscr{T} = \langle \epsilon \rangle$ then $\Delta \Vdash^s \sigma : \epsilon \,/\, \mathscr{T}$ holds unconditionally.
- If $\mathscr{T} = \langle \mathscr{T}', \mathscr{A}_\ell \rangle$ then $\Delta \Vdash^s \sigma : \Gamma, A \,/\, \mathscr{T}$ holds iff $\Delta \Vdash^s \mathsf{tail}\, \sigma : \Gamma \,/\, \mathscr{T}'$ and $\Delta \Vdash_\ell \mathsf{head}\, \sigma : A[\mathsf{tail}\, \sigma] \,/\, \mathscr{A}(\mathsf{tail}\, \sigma)$.

Given $\mathscr{S} :: \Delta \Vdash^s \sigma : \Gamma \,/\, \mathscr{T}$, validity of equal substitutions $\Delta \Vdash^s \sigma = \sigma' : \Gamma \,/\, \mathscr{T} \,/\, \mathscr{S}$ is defined by recursion on $\mathscr{T} :: \Vdash^v \Gamma$ as follows:

- If $\mathscr{T} = \langle \epsilon \rangle$ then $\Delta \Vdash^s \sigma = \sigma' : \epsilon \,/\, \mathscr{T} \,/\, \mathscr{S}$ holds unconditionally.
- If $\mathscr{T} = \langle \mathscr{T}', \mathscr{A}_\ell \rangle$ then $\Delta \Vdash^s \sigma = \sigma' : \Gamma, A \,/\, \mathscr{T} \,/\, \mathscr{S}$ holds iff $\Delta \Vdash^s \mathsf{tail}\, \sigma = \mathsf{tail}\, \sigma' : \Gamma \,/\, \mathscr{T}' \,/\, \mathsf{tail}\, \mathscr{S}$ and $\Delta \Vdash_\ell \mathsf{head}\, \sigma = \mathsf{head}\, \sigma' : A[\mathsf{tail}\, \sigma] \,/\, \mathscr{A}(\mathsf{tail}\, \sigma)$.

Here $\mathsf{tail}\, \mathscr{S}$, for $\mathscr{S} :: \Delta \Vdash^s \sigma : \Gamma, A \,/\, \mathscr{T}$, is a proof showing that $\mathsf{tail}\, \sigma$ is a valid substitution. The idea behind these two relations is to treat substitutions as lists of terms, all of which need to be reducible.

Finally we can define validity of types, terms, and their equalities.

Definition A.8. Let $\mathscr{T} :: \Vdash^v \Gamma$. Then $\Gamma \Vdash^v_\ell A \,/\, \mathscr{T}$ iff for all $\mathscr{S} :: \Delta \Vdash^s \sigma : \Gamma \,/\, \mathscr{T}$ we have $\mathscr{A}(\sigma) :: \Delta \Vdash_\ell A[\sigma]$, and additionally for all $\Delta \Vdash^s \sigma' : \Gamma \,/\, \mathscr{T}$ and $\Delta \Vdash^s \sigma = \sigma' : \Gamma \,/\, \mathscr{T} \,/\, \mathscr{S}$ we have $\Delta \Vdash_\ell A[\sigma] = A[\sigma'] \,/\, \mathscr{A}(\sigma)$. Furthermore, for $\mathscr{A} :: \Gamma \Vdash^v_\ell A \,/\, \mathscr{T}$:

- $\Gamma \Vdash^{v}_{\ell} t : A \;/\; \mathscr{T} \;/\; \mathscr{A}$ holds iff for all $\mathscr{S} :: \Delta \Vdash^{s} \sigma : \Gamma \;/\; \mathscr{T}$ we have $\Delta \Vdash_{\ell} t[\sigma] : A[\sigma] \;/\; \mathscr{A}(\sigma)$, and additionally for all $\Delta \Vdash^{s} \sigma' : \Gamma \;/\; \mathscr{T}$ and $\Delta \Vdash^{s} \sigma = \sigma' : \Gamma \;/\; \mathscr{T} \;/\; \mathscr{S}$ we have $\Delta \Vdash_{\ell} t[\sigma] = t[\sigma'] : A[\sigma] \;/\; \mathscr{A}(\sigma)$,
- $\Gamma \Vdash^{v}_{\ell} A = B \;/\; \mathscr{T} \;/\; \mathscr{A}$ holds iff for all $\mathscr{S} :: \Delta \Vdash^{s} \sigma : \Gamma \;/\; \mathscr{T}$ we have $\Delta \Vdash_{\ell} A[\sigma] = B[\sigma] \;/\; \mathscr{A}(\sigma)$, and
- $\Gamma \Vdash^{v}_{\ell} t = u : A \;/\; \mathscr{T} \;/\; \mathscr{A}$ holds iff for all $\mathscr{S} :: \Delta \Vdash^{s} \sigma : \Gamma \;/\; \mathscr{T}$ we have $\Delta \Vdash_{\ell} t[\sigma] = u[\sigma] : A[\sigma] \;/\; \mathscr{A}(\sigma)$.

Again we refer to Abel et al. [2017] or the formalization for the details, but the validity relation is essentially reducibility under a (valid) substitution.

For this relation, the fundamental lemma can be proven, and as mentioned before, the fundamental lemma for reducibility follows as a special case.

Theorem A.9.

- *If* $\Gamma \vdash A$ *then* $\Gamma \Vdash_{1} A$.
- *If* $\Gamma \vdash t : A$ *then* $\mathscr{A} :: \Gamma \Vdash_{1} A$ *and* $\Gamma \Vdash_{1} t : A \;/\; \mathscr{A}$.
- *If* $\Gamma \vdash A = B$ *then* $\mathscr{A} :: \Gamma \Vdash_{1} A$ *and* $\Gamma \Vdash_{1} B$ *and* $\Gamma \Vdash_{1} A = B \;/\; \mathscr{A}$.
- *If* $\Gamma \vdash t = u : A$ *then* $\mathscr{A} :: \Gamma \Vdash_{1} A$ *and* $\Gamma \Vdash_{1} t = u : A \;/\; \mathscr{A}$.

Using the logical relation and the fundamental lemma, Abel et al. are able to show several properties of the language that they study, culminating with decidability of type conversion. These properties still hold, with minor changes, after the addition of grades.

B USAGE INFERENCE

The observation that the usage relation is algorithmic can be formalized via the following function, which infers a usage context from a term.

Definition B.1. The usage inference function $|\_|$, which computes a usage context from a term, is defined recursively as follows:

$$\begin{array}{rcl@{\qquad}rcl}
|\mathsf{U}| & := & \mathbf{0} & |\mathsf{zero}| & := & \mathbf{0} \\
|\mathbb{N}| & := & \mathbf{0} & |\mathsf{suc}\, t| & := & |t| \\
|\bot| & := & \mathbf{0} & |\mathsf{natrec}^{q}_{p,r}\, A\, z\, s\, n| & := & (|z| \wedge |n|) \circledast_{r} \big(\mathsf{tail}\,(\mathsf{tail}\,|s|) + p|n|\big) \\
|\Pi^{q}_{p}\, F\, G| & := & |F| + \mathsf{tail}\,|G| & |(t,u)_{\otimes}| & := & |t| + |u| \\
|\Sigma^{q}\, F\, G| & := & |F| + \mathsf{tail}\,|G| & |(t,u)_{\&}| & := & |t| \wedge |u| \\
|x_i| & := & \mathbf{e}_i & |\mathsf{fst}\, t| & := & |t| \\
|\lambda^{p} t| & := & \mathsf{tail}\,|t| & |\mathsf{snd}\, t| & := & |t| \\
|t\,^{p}u| & := & |t| + p|u| & |\mathsf{prodrec}^{q}_{r}\, A\, t\, u| & := & r|t| + \mathsf{tail}\,(\mathsf{tail}\,|u|) \\
 & & & |\mathsf{emptyrec}_{p}\, A\, t| & := & p|t|
\end{array}$$

Here $(\mathsf{tail}\,\gamma)(x_i) = \gamma(x_{i+1})$. Note that $|\_|$ makes no checks on the annotations and will produce a context whether a term is well-resourced or not. However, if the annotations are correct, then the produced context will not only be valid but will also be an upper bound on valid contexts. In other words, for any well-resourced term, we can always find the most specific context.

Theorem B.2. *If* $\gamma \blacktriangleright t$ *for some* γ*, then* $|t| \blacktriangleright t$ *and* $\gamma \leq |t|$.

Proof. By induction on $\gamma \blacktriangleright t$ using the monotonicity of the operators. □

Additionally, we get decidability of the usage relation (if one can decide whether Prodrec r holds for any r).

Theorem B.3. *For any usage context γ and term t, $|t| \blacktriangleright t$ and $\gamma \blacktriangleright t$ are decidable.*

Proof. A lemma showing that either $|t| \blacktriangleright t$, or that $\gamma \blacktriangleright t$ does not hold for any γ, can be shown by induction on t. Decidability of $\gamma \blacktriangleright t$ can then be proved using decidability of the partial order and Theorem B.2. □

By viewing substitution matrices as lists of usage contexts, a similar approach can be taken for inferring a substitution matrix from a substitution.

Definition B.4. The substitution inference function $\|\_\|$, which infers a substitution matrix from a substitution, is defined such that for a substitution taking terms of m free variables to terms of n free variables, $\|\sigma\|$ is an $m \times n$-matrix such that the i-th row is $\left|x_i[\sigma]\right|$.

Similarly to the situation for contexts there is no guarantee that $\|\sigma\|$ is a valid substitution matrix for σ, but it is if a valid substitution matrix exists.

Theorem B.5. *If $\Psi \blacktriangleright \sigma$ for some Ψ, then $\|\sigma\| \blacktriangleright \sigma$.*

Proof. By applying Theorem B.2 to every term in σ and its corresponding row in Ψ. □

REFERENCES

Andreas Abel. 2006. Polarized Subtyping for Sized Types. In *Computer Science – Theory and Applications, First International Computer Science Symposium in Russia, CSR 2006 (LNCS, Vol. 3967)*. 381–392. https://doi.org/10.1007/11753728_39

Andreas Abel. 2018. Resourceful Dependent Types. In *24th International Conference on Types for Proofs and Programs, TYPES 2018, Abstracts*. 7–8. https://types2018.projj.eu/book-of-abstracts/

Andreas Abel and Jean-Philippe Bernardy. 2020. A Unified View of Modalities in Type Systems. *Proc. ACM Program. Lang.* 4, ICFP, Article 90 (Aug. 2020), 28 pages. https://doi.org/10.1145/3408972

Andreas Abel, Thierry Coquand, and Bassel Mannaa. 2016. On the Decidability of Conversion in Type Theory. In *TYPES 2016, Types for Proofs and Programs, 22nd Meeting, Book of Abstracts*. 5–6. http://www.types2016.uns.ac.rs/images/abstracts/abel1.pdf

Andreas Abel, Nils Anders Danielsson, and Oskar Eriksson. 2023. *An Agda Formalization of a Graded Modal Type Theory with a Universe and Erasure*. https://doi.org/10.5281/zenodo.8119348

Andreas Abel, Nils Anders Danielsson, and Andrea Vezzosi. 2021. Compiling Programs with Erased Univalence. Draft. https://www.cse.chalmers.se/~nad/publications/abel-danielsson-vezzosi-erased-univalence.html

Andreas Abel, Joakim Öhman, and Andrea Vezzosi. 2017. Decidability of Conversion for Type Theory in Type Theory. *Proc. ACM Program. Lang.* 2, POPL, Article 23 (Dec. 2017), 29 pages. https://doi.org/10.1145/3158111

Andreas Abel and Gabriel Scherer. 2012. On Irrelevance and Algorithmic Equality in Predicative Type Theory. *Logical Methods in Computer Science* 8, 1 (March 2012), 36 pages. https://doi.org/10.2168/LMCS-8(1:29)2012

Robert Atkey. 2018. Syntax and Semantics of Quantitative Type Theory. In *LICS '18: Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. 56–65. https://doi.org/10.1145/3209108.3209189

Jean-Philippe Bernardy, Mathieu Boespflug, Ryan R. Newton, Simon Peyton Jones, and Arnaud Spiwack. 2017. Linear Haskell: Practical Linearity in a Higher-Order Polymorphic Language. *Proc. ACM Program. Lang.* 2, POPL, Article 5 (Dec. 2017), 29 pages. https://doi.org/10.1145/3158093

Edwin Brady. 2021. Idris 2: Quantitative Type Theory in Practice. In *35th European Conference on Object-Oriented Programming, ECOOP 2021 (LIPIcs, Vol. 194)*. 26 pages. https://doi.org/10.4230/LIPIcs.ECOOP.2021.9

Aloïs Brunel, Marco Gaboardi, Damiano Mazza, and Steve Zdancewic. 2014. A Core Quantitative Coeffect Calculus. In *Programming Languages and Systems, 23rd European Symposium on Programming, ESOP 2014 (LNCS, Vol. 8410)*. 351–370. https://doi.org/10.1007/978-3-642-54833-8_19

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, Article 50 (Jan. 2021), 32 pages. https://doi.org/10.1145/3434331

Pritam Choudhury, Harley Eades III, and Stephanie Weirich. 2022. A Dependent Dependency Calculus. In *Programming Languages and Systems, 31st European Symposium on Programming, ESOP 2022 (LNCS, Vol. 13240)*. 403–430. https://doi.org/10.1007/978-3-030-99336-8_15

Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. 2015. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In *21st International Conference on Types for Proofs and Programs, TYPES 2015 (LIPIcs, Vol. 69)*. 34 pages. https://doi.org/10.4230/LIPIcs.TYPES.2015.5

Karl Crary, Stephanie Weirich, and Greg Morrisett. 2002. Intensional polymorphism in type-erasure semantics. *Journal of Functional Programming* 12, 6 (Nov. 2002), 567–600. https://doi.org/10.1017/S0956796801004282

Peng Fu, Kohei Kishida, and Peter Selinger. 2022. Linear Dependent Type Theory for Quantum Programming Languages. *Logical Methods in Computer Science* 18, 3 (Sept. 2022), 44 pages. https://doi.org/10.46298/lmcs-18(3:28)2022

Herman Geuvers. 1994. A short and flexible proof of Strong Normalization for the Calculus of Constructions. In *Types for Proofs and Programs, International Workshop TYPES '94 (LNCS, Vol. 996)*. 14–38. https://doi.org/10.1007/3-540-60579-7_2

Dan R. Ghica and Alex I. Smith. 2014. Bounded Linear Types in a Resource Semiring. In *Programming Languages and Systems, 23rd European Symposium on Programming, ESOP 2014 (LNCS, Vol. 8410)*. 331–350. https://doi.org/10.1007/978-3-642-54833-8_18

Daniel Gratzer, G.A. Kavvos, Andreas Nuyts, and Lars Birkedal. 2021. Multimodal Dependent Type Theory. *Logical Methods in Computer Science* 17, 3 (July 2021), 67 pages. https://doi.org/10.46298/lmcs-17(3:11)2021

Pierre Letouzey. 2003. A New Extraction for Coq. In *Types for Proofs and Programs, International Workshop, TYPES 2002 (LNCS, Vol. 2646)*. 200–219. https://doi.org/10.1007/3-540-39185-1_12

Conor McBride. 2016. I Got Plenty o' Nuttin'. In *A List of Successes That Can Change the World (LNCS, Vol. 9600)*. 207–233. https://doi.org/10.1007/978-3-319-30936-1_12

Nathan Mishra-Linger and Tim Sheard. 2008. Erasure and Polymorphism in Pure Type Systems. In *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008 (LNCS, Vol. 4962)*. 350–364. https://doi.org/10.1007/978-3-540-78499-9_25

Benjamin Moon, Harley Eades III, and Dominic Orchard. 2021. Graded Modal Dependent Type Theory. In *Programming Languages and Systems, 30th European Symposium on Programming, ESOP 2021 (LNCS, Vol. 12648)*. 462–490. https://doi.org/10.1007/978-3-030-72019-3_17

Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. 2019. Quantitative Program Reasoning with Graded Modal Types. *Proc. ACM Program. Lang.* 3, ICFP, Article 110 (July 2019), 30 pages. https://doi.org/10.1145/3341714

Tomas Petricek, Dominic Orchard, and Alan Mycroft. 2014. Coeffects: A Calculus of Context-Dependent Computation. In *ICFP'14, Proceedings of the 2014 ACM SIGPLAN International Conference on Functional programming*. 123–135. https://doi.org/10.1145/2628136.2628160

Frank Pfenning. 2001. Intensionality, Extensionality, and Proof Irrelevance in Modal Type Theory. In *Proceedings, 16th Annual IEEE Symposium on Logic in Computer Science*. 221–230. https://doi.org/10.1109/LICS.2001.932499

Jason Reed and Benjamin C. Pierce. 2010. Distance Makes the Types Grow Stronger: A Calculus for Differential Privacy. In *ICFP'10, Proceedings of the 2010 ACM SIGPLAN International Conference on Functional programming*. 157–168. https://doi.org/10.1145/1863543.1863568

Egbert Rijke, Michael Shulman, and Bas Spitters. 2020. Modalities in homotopy type theory. *Logical Methods in Computer Science* 16, 1 (Jan. 2020), 79 pages. https://doi.org/10.23638/LMCS-16(1:2)2020

Martin Steffen. 1998. *Polarized Higher-Order Subtyping.* Ph. D. Dissertation. Universität Erlangen-Nürnberg.

Sandro Stucki and Paolo G. Giarrusso. 2021. A Theory of Higher-Order Subtyping with Type Intervals. *Proc. ACM Program. Lang.* 5, ICFP, Article 69 (Aug. 2021), 30 pages. https://doi.org/10.1145/3473574

Matúš Tejiščák. 2020. A Dependently Typed Calculus with Pattern Matching and Erasure Inference. *Proc. ACM Program. Lang.* 4, ICFP, Article 91 (Aug. 2020), 29 pages. https://doi.org/10.1145/3408973

The Agda Team. 2023. *Agda User Manual, Release 2.6.3.* https://readthedocs.org/projects/agda/downloads/pdf/v2.6.3/

The Coq Development Team. 2023. *The Coq Reference Manual, Release 8.17.1.* https://github.com/coq/coq/releases/download/V8.17.1/coq-8.17.1-reference-manual.pdf

The Univalent Foundations Program. 2013. *Homotopy Type Theory: Univalent Foundations of Mathematics* (first ed.). https://homotopytypetheory.org/book/

Dennis Volpano, Cynthia Irvine, and Geoffrey Smith. 1996. A sound type system for secure flow analysis. *Journal of Computer Security* 4, 2–3 (April 1996), 167–187. https://doi.org/10.3233/JCS-1996-42-304

James Wood and Robert Atkey. 2021. A Linear Algebra Approach to Linear Metatheory. In *Proceedings Second Joint International Workshop on Linearity & Trends in Linear Logic and Applications (EPTCS, Vol. 353)*. 195–212. https://doi.org/10.4204/EPTCS.353.10

Received 2023-03-01; accepted 2023-06-27