Thesis for The Degree of Licentiate of Engineering

Graded Modal Type Theory, Formalized

Oskar Eriksson

Department of Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden, 2025

Graded Modal Type Theory, Formalized

Oskar Eriksson

Department of Computer Science and Engineering
Division of Computing Science
University of Gothenburg
SE-412 96 Göteborg,
Sweden
Phone: +46(0)31 772 1000

Printed by Chalmers Digitaltryck,
Gothenburg, Sweden 2025.

Graded Modal Type Theory, Formalized

Oskar Eriksson

Department of Computer Science and Engineering
University of Gothenburg

Abstract

A graded modal type theory equips a type theory with a semiring of grades, which enables encoding additional information of some kind. A typical application of such a system is to encode quantitative information, specifically how many times variables are used or referenced at runtime. Concrete examples include erasure or linear types. In this thesis, we present such a type theory with dependent types, designed primarily with encoding quantitative information in mind. The theory supports Π-types, Σ-types, unit, and empty types, as well as a hierarchy of universes. It also supports natural numbers, and special attention is given to establishing a method for counting variable uses for the recursive eliminator of this type. We prove some key meta-theoretical results for this system, as well as two main correctness results. Firstly, we show correctness for erasure in the sense that it is safe to remove erased information during compilation without affecting the result of evaluation. Secondly, we show that the system is capable of correctly tracking how many times variables should be referenced through an abstract machine in which heap lookups (corresponding to variable references) are tracked. The thesis is accompanied by an Agda formalization in which the results have been mechanized.

Keywords

graded modal type theory, quantitative type theory, dependent types, erasure, linearity, abstract machine, formalization

List of Publications

Appended publications

This thesis is based on the following publications:

[**Paper A**] A. Abel, N. A. Danielsson, **O. Eriksson**
A Graded Modal Dependent Type Theory with a Universe and Erasure, Formalized
Proc. ACM Program. Lang. 7, ICFP, Article 220 (August 2023).

[**Paper B**] **O. Eriksson**, N. A. Danielsson, A. Abel
A Resource-Correct Graded Modal Dependent Type Theory with Recursion, Formalized
Extended version of paper submitted for review.

Acknowledgment

First and foremost, I want to thank my supervisor, Andreas Abel, for his invaluable support throughout this thesis and for introducing me to graded modal type theory as the subject of my master's thesis project, which served as the starting point for the work in this thesis. I also wish to express my sincere gratitude to my co-supervisor, Nils Anders Danielsson, for his equally important guidance and insight along the way. I feel very fortunate to have both of you as my supervisors. Lastly, I am grateful to all my friends, family, and colleagues for their continuous support and encouragement.

This research was supported by *Vetenskapsrådet* (the Swedish Research Council) via project 2019-04216 *Modal typteori med beroende typer* (Modal Dependent Type Theory) and the *Knut and Alice Wallenberg Foundation* via project 2019.0116.

Contents

Part I

Summary

Chapter 1

Introduction

It is well-established that static type systems allow certain classes of bugs to be eliminated by giving the programmer guarantees about the behaviour of their programs. The class of bugs which can be avoided in this manner depends on what kind of information the type system allows to be encoded. Dependent types are a powerful example of this by allowing types to depend on values. The classic example is the data type of vectors, lists of a given length, here defined in Agda:

```
data Vec (A : Set) : ℕ → Set where
  [] : Vec A 0
  _::_ : {n : ℕ} → A → Vec A n → Vec A (suc n)
```

The length of the vector is encoded in the type which can be leveraged by the programmer, for instance, by defining a safe head function:

```
head : Vec A (suc n) → A
head (x :: xs) = x
```

The type system guarantees that this function can only be applied to non-empty lists, so its behaviour for empty vectors does not need to be defined. Another example is the append function, where the type guarantees that the length of the resulting vector is the sum of the lengths of the input vectors:

```
_++_ : Vec A n → Vec A m → Vec A (n + m)
[] ++ ys = ys
(x :: xs) ++ ys = x :: xs ++ ys
```

Guarantees like these come with a cost: unlike the usual definition of lists, vectors carry around information about their length. In fact, since a vector of length n consists of $n+1$ sub-vectors it carries with it all numbers from 0 to n — a substantial memory overhead compared to lists [Brady et al., 2003]. While these are important for type-checking, these examples do not use them in the actual implementations, meaning that they serve no purpose after type-checking. Consequently, it should be safe to remove them during compilation, turning vectors into lists at runtime and avoiding the unnecessary space overhead while keeping the guarantees given by the type.

Removing such computationally irrelevant information is known as *erasure*. Agda has support for erasure via annotations on types, marking arguments as being computationally irrelevant or "erased" [The Agda Team, 2024]. Such arguments are used during type-checking but are removed during compilation (at least by some of the compiler backends). For the vector type, a version with erased length information can be defined as follows, the only differences are the `@0` annotations on the length arguments.

```
data Vec (A : Set) : @0 ℕ → Set where
  [] : Vec A 0
  _::_ : {@0 n : ℕ} → A → Vec A n → Vec A (suc n)

head : {@0 n : ℕ} → Vec A (suc n) → A
head (x :: xs) = x

_++_ : {@0 n m : ℕ} → Vec A n → Vec A m → Vec A (n + m)
[] ++ ys = ys
(x :: xs) ++ ys = x :: xs ++ ys
```

Of course, one cannot mark anything as erased; the type-checker only allows such annotations for arguments that it can determine to be used zero times at runtime. The following definition of a length function is rejected by Agda since the length index is used in a computationally relevant way.

```
length : {@0 n : ℕ} → Vec A n → ℕ
length {n = n} xs = n
```

This does not mean that the argument needs to be completely unused, that is, not appear in the code, but it may only appear in other erased contexts such as an erased function argument.

Going beyond erasure, one can imagine type systems tracking other numbers of uses as well. Typical examples of this include *linear types* [Wadler, 1990] for which arguments are required to be used *exactly* once, or *affine types* for which they are used *at most* once. One could also go in the opposite direction of erasure and instead track computationally relevant arguments, that is, arguments that are used *at least* once. One might also consider combinations of these [Choudhury et al., 2021; Abel and Bernardy, 2020]. All of these, including erasure, are examples of type systems tracking *quantitative* information.

One application of such systems is utilizing the information about runtime usage in optimizations like the one already discussed for erasure. For linear or affine types, one can apply the knowledge that certain data is used at most once to perform safe in-place updates, achieving similar space optimizations [Bernardy et al., 2017; Lorenzen et al., 2024]. Another use case is to take advantage of how the increased expressiveness of the type system allows more static guarantees such as implementing session types for ensuring correctness of communication protocols [Brady, 2021] or ensuring that indices are used only once in a library for tensor algebra [Bernardy and Jansson, 2025].

Dependent type systems with the capability to *correctly* track such quantitative information is the main topic of this thesis. The approach considered is *graded modal type theory* in which the type system is equipped with a semiring which is used, in this case, to encode information about how many times variables are referenced. The implementation of erasure in Agda is based on such a system. Graded modal type theory is described in more detail in Section 1.1.

The contents of this thesis have been formalized using Agda. This includes all definitions and results presented in the appended papers unless stated otherwise. The formalization is described in more detail in Section 2.3.

1.1 Graded Modal Type Theory

A graded modal type theory, or just graded type theory, is a type theory equipped with some algebraic structure. Its elements, *grades*, are used to annotate certain terms and/or types in order to encode some kind of information. Typical examples are quantitative

information [McBride, 2016; Atkey, 2018; Choudhury et al., 2021; Moon et al., 2021] as discussed above or confidentiality classification for ensuring that secret information cannot be leaked [Choudhury et al., 2022; Orchard et al., 2019]. In this thesis, the focus is primarily on the former. Usually, the algebraic structure is not fixed and the type theory can in this regard be considered to be parametrized over the structure. It can then be instantiated with different concrete structures depending on the kind of information that should be encoded.

While the specifics of how graded type theories are defined differ depending on the intended applications, some aspects are notably common. For one, the algebraic structure is normally a partially ordered semiring. That is, it has addition and multiplication operators with multiplication distributing over addition as well as a unit (1) and zero (0) grade. The operators and order relation are used to define the typing rules as explained below and the type of information one can encode thus depends on how the operators and order are defined. A few semiring instances of note for this thesis are sketched in Section 1.2. For some applications, the semiring is given some additional structure. As an example, quantitative type theory (QTT) imposes that $p + q = 0$ should only be allowed if $p = q = 0$ [McBride, 2016] and $pq = 0$ only if either p or q is equal to zero [Atkey, 2018]. In information flow applications, the grades are often assumed to form a lattice, representing security levels [Choudhury et al., 2022; Orchard et al., 2019].

In its simplest form, one might describe a graded type theory as assigning a grade to any free variable, such as a function argument, in addition to a type. The grade signifies some information about how the variable should be used. For quantitative settings, this means that the grade represents how many times the variable should be, or is allowed to be, referenced while information flow instances assign a constraint on what security clearance is needed in order to reference the variable. Besides assigning grades to variables, some theories allow assigning grades to constructor arguments, similarly signifying how these should be used.

In some systems these grades can be inferred [Tejiščák, 2020] while others require the syntax to be annotated with grades. In such cases, annotations might be needed only on types as for Moon et al. [2021] or in the Agda example above. There, we annotated just some function domains, marking the natural number arguments with grade `@0`, indicating that they are referenced zero times at runtime. In other cases, further annotations are required also on terms, for instance, functions (lambdas) and/or applications [Choudhury et al., 2022, 2021]. In this thesis, the latter approach is used.

Regardless of where grade annotations are required, the type system is tasked with ensuring that they are correct. There are, in principle, two main styles of defining the typing rules for grades. The first [McBride, 2016; Choudhury et al., 2021] can be described, more or less, as counting variable uses with the given semiring. A single variable occurrence is assigned grade 1 and a lack of occurrences is assigned grade 0. Several occurrences, in particular from sub-terms, are added using the addition of the semiring. The multiplication represents the uses of one term by another with the key example being function applications. A function using its argument p times will use a variable pq times if that variable is used q times in the function argument. A notable special case of this is when $p = 0$, representing an unused (erased) function argument. In this case, the q variable uses of the function argument end up not being counted. This represents that erased information is allowed to be used in other erased contexts.

Depending on the application, it might not be desirable to fully distinguish between all grades. This is the case, for instance, for affine functions which can use their argument either zero or one times, meaning that there might be several valid grade assignments. In particular, both erased and linear function arguments may be considered to be affine. This kind of *compatibility* between grades is expressed through the partial order with the

grade typing allowing a form of subsumption or weakening by allowing a variable used at grade p to also be used at a lower grade q. For instance, code accessing a variable of a high confidentiality level will continue to work if the variable is assigned a lower confidentiality level.

The second style of defining grade typing is in many ways similar to the first but with the typing judgement too annotated with a grade, in this thesis referred to as the *mode* (possibly with restrictions on which grades are allowed as modes) [Atkey, 2018; Choudhury et al., 2022]. Variable counting mostly progresses as before but the grade corresponding to a single occurrence is now given by the current mode. In an information flow interpretation, the mode might be interpreted as the security clearance of the current observer; a well-typed program does not reveal any secrets of a higher security level than the current mode. This formulation also allows different rules under different modes. For instance, Atkey [2018] allows different forms of eliminations for Σ-types (referred to as dependent tensor product types) under erased and computationally relevant modes. This thesis uses both approaches, with Paper A mostly using the first while Paper B uses the second.

The description above has not been specific to dependent types and is applicable both for simple and dependent types. In terms of grade typing, dependent types introduce one complication that does not appear for simple types, however. Namely, since types may depend on values they can use or reference variables which may affect how variable counting should be performed. A simple approach to this is to consider types to be computationally irrelevant, meaning that uses of the function argument [Atkey, 2018; Choudhury et al., 2022, 2021] are not accounted for in the (dependent) codomain of Π-types. An alternative option is to annotate the Π-type with two grades, one corresponding to the grade of function argument in the function body as before and the other corresponding to the use of the function argument in the type of the codomain [Moon et al., 2021; Abel, 2018]. This is the method used in this thesis.

1.2 Some Instances of Graded Modal Types

Some parts of this thesis put particular emphasis on specific instantiations of graded type theories. These are erasure, affine types, and linear types, all of which have been discussed above. Here, we will see how one can instantiate a graded modal type theory with semirings in order to achieve these interpretations. Note that while the instantiations shown here are consistent with the type theory presented in this thesis, other instantiations may be more suitable for other systems.

An instance for erasure is the two element semiring $\{0, \omega\}$ where ω serves as the unit grade. Addition and multiplication are defined as shown in Table 1 and $\omega \leq 0$. The grade 0 corresponds to the annotation `@0` from the Agda examples and denotes erased information. Conversely, the grade ω can be seen as representing computationally relevant information but, the partial order also allows computationally irrelevant information to be marked with ω. A more accurate interpretation of ω is thus that it represents *any* number of uses, including zero.

Table 1: Addition and multiplication for a semiring for erasure.

$+$	0	ω
0	0	ω
ω	ω	ω

$\cdot$	0	ω
0	0	0
ω	0	ω

For affine types, one can use the three element semiring $\{0, 1, \omega\}$ with addition and multiplication as shown in Table 2 and $\omega \leq 1 \leq 0$. The grades 0 and ω can be interpreted

similarly to the erasure instance and grade 1 represents zero or one use since $1 \leq 0$. A linear-types instance can be retrieved by changing only the partial order to $1 \geq \omega \leq 0$ with 0 and 1 being incomparable. Here, ω still represents any number of uses since it is smaller than all other grades but 1 no longer allows zero uses and thus represents linear information.

Table 2: Addition and multiplication for semirings for affine or linear types.

$+$	0	1	ω
0	0	1	ω
1	1	ω	ω
ω	ω	ω	ω

$\cdot$	0	1	ω
0	0	0	0
1	0	1	ω
ω	0	ω	ω

Chapter 2

Summary of Included Work

This thesis is a collection thesis consisting of the two papers "A Graded Modal Dependent Type Theory with a Universe and Erasure, Formalized" and "A Resource-Correct Graded Modal Dependent Type Theory with Recursion, Formalized" which are included in Part II. This chapter gives a summary of the contents of the articles and their results. Both articles are formalized in Agda and an overview of their formalizations is given in Section 2.3.

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

Paper A consists of three main parts. In the first part, we define a graded modal dependent type theory and prove some of its meta-theoretical properties. Our main motivation is erasure, but the theory is intended to be useable also for other instances, including non-quantitative ones. The type theory we consider consists of a single (Russell) universe, graded Π-types, two kinds of Σ-types (one with a form of pattern matching and one with projections and η-equality), natural numbers and an empty type. We annotate both types and terms, notably lambda abstractions and applications, as well as the eliminators for natural numbers and Σ-types, assigning grades to the variables bound by the terms. The eliminator for the empty type is also annotated with a grade representing the uses of the impossible branch. Of particular note is that this allows impossible branches to be erased.

Our structure of grades, which we refer to as a *modality*, is a partially ordered semiring with meets. Thus, any two grades always have an infimum in our setting. The purpose of this is to support programs with branching behaviour that have different variable uses in the branches—taking the infimum yields a grade that is compatible with either branch. In our case, the most notable example of this are Σ-types with projections as the two components of a pair can be seen as two different branches depending on which projection is applied.

We define our judgement for grade typing, $\boxed{\gamma \blacktriangleright t}$ separately from the usual typing. The normal typing judgments are mostly defined without concern for grades but ensure that grades on lambdas and applications match. The grade typing judgement is referred to as the *usage relation* and we say that t is *well-resourced*[1] whenever $\gamma \blacktriangleright t$ holds for some context γ which assigns grades to the free variables of t. For the most part, the usage relation is defined in a standard way for a non-moded form of grade typing (that is, as outlined in Section 1.1). A notable exception is pairs which allow projections where,

[1]This terminology hints at a quantitative interpretation of variables "consuming" some resource as the term is evaluated. For a well-resourced program, there are sufficiently many resources available to evaluate the program. This interpretation is discussed more in Paper B.

as explained above, the grades of the two components are combined using meet instead of addition. For the Σ-type with pattern matching, we generally allow matching on an erased term as long as the two components of the pair are just used in erased positions. In this case, we speak of an *erased match*. This concept would also apply to other types with a single constructor, but we only consider Σ-types in this work.

Some attention needs to be given to the usage rule for the eliminator for natural numbers where the recursion, and its unknown depth, makes variable "counting" less straightforward. In order to assign grades for such programs, our definition of modalities comes with an additional operator, which is required to satisfy some properties and is used only for this purpose. However, as discussed in Paper B, this solution is not fully suitable for interpretations beyond erasure.

Having defined our graded type theory, we prove some of its meta-theoretical properties. In particular, we show a substitution lemma, subject reduction, and decidability of the usage relation.

The second part is a case study of a particular use case of the type theory we introduced in the first part, namely erasure. Here, we consider only instantiations of the theory with modalities where the zero satisfies some additional properties. We say that such modalities have a *well-behaved zero*. We compile programs in the graded source language to an untyped target language such that parts of programs that are marked as erased are replaced with a special term representing an undefined value. Since erased parts are unused during evaluation, this should be safe in the sense that it does not affect the outcome of running the program.

That erasure is indeed safe is proven for the case of programs of type $\mathbb{N}$ by a logical relation argument. The logical relation, relating terms of the source and target languages, is defined by recursion over (a structure representing) the type of the source language term. It is defined in such a way as to make sure that related natural number terms reduce to the same numeral. The fundamental lemma of the logical relation states that well-typed and well-resourced terms are related to their compiled counterpart from which it follows that programs of type $\mathbb{N}$ evaluate to the same numeral before and after compilation.

Our correctness theorem holds both for closed and for open programs under the assumption that all free variables are used only in erased positions. In the latter case, we require that the types of all free variables, which might represent postulates, are logically consistent with each other and the base theory.[2] We also disallow erased matches. While we show that this is a necessary assumption for our correctness theorem to hold, we conjecture that a weaker version, with canonicity only for the target language, holds even if such matches are allowed.

Finally, in the third part, we extend the theory with graded Σ-types where the first component of pairs is assigned a grade representing how it should be used. In order to support this, the usage relation is modified to the style with a mode $\boxed{\gamma \blacktriangleright^m t}$ (in our case, only 0 and 1 are allowed as modes). This change is most notably used to allow taking the first projection of a pair with an erased first component in erased settings (mode 0) while disallowing it in non-erased settings (mode 1). We show that the meta-theoretical results still hold with this change for modalities with a well-behaved zero and we similarly show that the results of the erasure case study still hold.

Statement of contributions The paper is written by Andreas Abel, Nils Anders Danielsson and me. The technical content of the first two parts was devised by me and Andreas and formalized by me while the third part was done by Nils Anders. The initial draft relating to the first two parts was written by me and the third part by Nils Anders. All authors revised the initial draft.

[2]The base theory has been shown to be consistent.

2.2 A Resource-Correct Graded Modal Dependent Type Theory with Recursion, Formalized

Paper B builds on the work done in Paper A by showing that the graded type theory presented there is correct in a wider sense than erasure and also addresses an issue with the usage counting for the eliminator for natural numbers. The setting is almost the same as in Paper A but now includes a hierarchy of universes as well as two kinds of unit types, matching the two kinds of Σ-types. While these additions round off the feature set of the type theory we study, neither of them has a major impact on results or proof techniques in this paper. As in the erasure case study, we work only with modalities with a well-behaved zero. Our presentation uses the moded usage relation defined at the end of that paper, but the results also hold for the variant without modes.

The main goal of this paper is to prove correctness for modalities with quantitative interpretations. We do this by defining a heap- and stack-based abstract machine where the heap tracks variable lookups. When new entries are added to the heap, they are assigned a grade representing how many lookups are expected or allowed. After a lookup has been performed, "fewer" further lookups are expected, and the corresponding grade is thus updated accordingly. The machine also prevents lookups that would cause entries to be used too many times. For instance, looking up an entry with grade 0 once is prevented. We thus require the modalities we work with to support a form of partial subtraction where lookup is allowed only when subtraction is defined, and the grade of the entry is updated to be the result of the subtraction.

Entries are added to the heap when evaluating, for instance, function applications where the function argument is placed on the heap. The grade annotation on applications represents how many times the function argument should be used or referenced, and that grade is accordingly associated with that heap entry. The expectation is that once evaluation finishes, the entry will have been looked up that many times, or, more accurately, a compatible number of times. This is expressed as the grades of the final heap being bounded from above by 0. In particular, for the linear types modality, all linear arguments must, at the end of evaluation, be associated with grade 0, meaning that they have been looked up exactly once. For the affine types modality, however, entries in the final heap may also be assigned grade 1, since $1 \leq 0$, but its subtraction ensures that any affine argument can only have been looked up at most once.

As in Paper A, our results hold both for closed programs and for open programs where all free variables are used only in erased positions. In the latter case, we again assume that programs are typed in a consistent context and the matching on erased pairs or unit elements is prohibited.

As mentioned in Section 2.1, our method of usage counting for the eliminator for natural numbers in Paper A is unsuitable for modalities with interpretations that go beyond erasure. In particular, we show here that the addition function for natural numbers, defined in the usual way, is not considered to be linear in that system. Furthermore, adding a number n to itself is incorrectly[3] considered to be a linear function in n. Consequently, our correctness theorem for the abstract machine is not proven using the usage rule from Paper A.

Instead, we define a new usage rule. The idea behind it is that while counting variable uses for a recursive function is difficult due to the unknown recursion depth, it is possible to do for a known recursion depth by unfolding the recursion. A valid usage count is then

[3]It is possible to define a linear function for doubling a natural number but defining it by addition in this way should not be linear.

one that is compatible with any recursion depth, that is, a lower bound of all (infinitely many) possibilities. In particular, our usage rule takes the infimum. Our correctness theorem is proven for this rule given some additional assumptions about infima for the modality. The main meta-theoretical results and the results from the erasure case study of Paper A are also shown to still hold under the same assumptions. We also sketch how this method might be used for usage counting for other inductive data structures, but we do not formalize this claim.

Statement of contributions The paper is written by me, Nils Anders Danielsson and Andreas Abel. I was responsible for most of the technical content and formalization with input and guidance from Nils Anders and Andreas. The initial draft of the paper was written by me and was revised by all authors.

2.3 The Formalization

Both papers are accompanied by Agda formalizations in which all results have been proven. In both cases, formalized definitions and results are highlighted in blue which link (in the digital version) to an HTML rendering[4] of the corresponding Agda code. The formalization of Paper B [Eriksson et al., 2025] is an extension of the one of Paper A [Abel et al., 2023] and so includes the results of both. It is developed using the `--safe` flag, which turns off features known to be inconsistent, as well as the `--without-K` flag, which (mainly) disables uniqueness of identity proofs [The Agda Team, 2024].

The formalization is based on an earlier formalization of dependent type theory, originally by Abel et al. [2017] with later contributions by Gaëtan Gilbert and Wojciech Nawrocki. This formalization was for an ungraded system with the main goal of proving decidability of type equality. This was done through a logical relation which additionally allowed showing several meta-theoretical results about the type theory such as subject reduction, admissibility of substitution, normalization, and consistency which we have made use of.[5]

The full formalization now consists of around 140 000 lines of code, which can be compared to 49 000 for Paper A and 15 000 for the original development. It should be noted, however, that this number includes parts of the formalization not covered by this thesis. A notable example is that the formalization includes identity types (added by Nils Anders Danielsson). While these are not covered by the included papers, (variants of) the results presented in them also hold when identity types are considered.

[4] Available at: `https://graded-type-theory.github.io/graded-type-theory/lic2025/`

[5] In the latest version of our formalization, some of these results are proven without using the logical relation.

Chapter 3

Discussion

We conclude this part by discussing some of the topics covered by this thesis as well as possible directions of future work within the scope of the formalization project. Further discussion can be found in the appended papers.

3.1 Canonicity for Open Terms

In both papers we show a form of canonicity for natural numbers. Such results are to be expected for closed terms but usually do not hold for open terms. While one would typically only run closed programs, one might want to add postulates to the theory which would act as free variables. The reason canonicity breaks in this case is that evaluation gets stuck upon reaching such a postulate. For instance, in a context where variable x has type $\mathbb{N}$, the program x is well-typed but will not evaluate to a numeral. Our proof avoids such settings by requiring that free variables may only be used in erased positions; in contrast, in the example above, x is computationally relevant.

Our argument relies on the consistency of all added axioms. Without this assumption, it would be possible to construct a natural number term using the eliminator for the empty type that does not evaluate to a numeral. The same is true for closed programs but Abel et al. [2017] establish that our base type theory is consistent. Note that this still allows using the eliminator for the empty type to rule out impossible branches. This is essentially what happens behind the scenes in the head function from Chapter 1 in order to rule out the nil branch. Canonicity is not broken since such branches never will be evaluated.

We also require that erased matches are disallowed. In the source language (given the reduction rules we are using) this is necessary since matching gets stuck if the scrutinee does not reduce to a canonical value (for instance, a pair). However, in the target language, such erased matches are compiled away, allowing evaluation to progress. We believe that such matches should be safe in the target language since matching on a single constructor data type does not incur any branching and the result of matching (for instance, the components of the pair) is not used. In Paper B we consider only the source language and therefore only consider the case without erased matches as they invoke the same issues as in Paper A.

Postulates do not need to be erased in order to still obtain canonicity. Coquand et al. [2017] have shown a similar result under the assumption that the types of all free variables are *negative* (either the empty type or a function with a negative type as codomain). The argument is similar to the one for erased axioms, namely that an element with a negative type has no computational content. Specifically, one cannot use such axioms to produce

natural numbers. Like us, this argument is done under the assumption that all postulates are consistent for similar reasons.

3.2 Formalizations of Type Theory

There are several formalization projects covering topics related to this thesis. Already mentioned is Abel et al. [2017] on which our formalization is built.

Wood and Atkey [2021] have formalized, in Agda, a graded system with simple types. Notably, they establish some meta-theory regarding substitution for graded systems which forms the basis for the substitution lemma presented in Paper A. Torczon et al. [2024] present another simply typed graded system, formalized in Rocq, for effects and coeffects. They show that their system correctly tracks how resources are used during evaluation. In contrast to the heap-based call-by-name evaluation we use in Paper B, their evaluation method is environment-based call-by-push-value. Supporting dependent types, Liu et al. [2024, 2025] have formalized, in Rocq, a graded type theory. Their main focus is on indistinguishability, ensuring that an observer cannot distinguish between two programs which use information that is secret to the observer. In their system, grades affect equality in order to enable programs with no discernable difference to the observer to be treated as equal.

Another notable example is the MetaRocq (formerly MetaCoq) project, formalizing Rocq in Rocq. Though not a graded system, parts of the project cover extraction which supports a form of erasure. Extraction is supported to an untyped lambda calculus, similar to the one we employ in Paper A and extracted programs are shown to preserve the behaviour of the source program [Sozeau et al., 2019]. Extracted programs can be further extracted into OCaml and this procedure has also been formally verified [Forster et al., 2024].

3.3 Future Work

While the focus of this thesis has been primarily on quantitative modalities, the original design was intended to be applicable in other cases as well. Paper A discusses to some degree how our results can be interpreted for a specific information flow instance, but a more general investigation on to what degree our system (correctly) handles such instances is missing. This could tie in to generalizing the mode system for the usage relation. At present, only grades 0 and 1 are supported while one might want modes corresponding to any grade in a security instance, denoting the security clearance of the user. In addition, the moded usage relation is only supported for modalities with a well-behaved zero in order to show important meta-theoretical properties. While this assumption works well for quantitative instances, this might not be the case for all instances of interest. Changing the mode system could thus enable support for a wider class of modalities.

Other directions for future work are more direct extensions of the results presented in the papers. For Paper A, the main open issue is to prove our conjecture that one can obtain canonicity in the source language also in the presence of erased matches. At this point, erased matches are not limited to Σ-types but also include identity types which complicate matters further. In Paper B, we sketched how one could define a usage rule for inductive data types beyond natural numbers, but our results are not formalized and some details are not spelled out. Adding general inductive types (in some form), extending the correctness proofs presented in this thesis, could be a way forward in this regard.

Bibliography

Andreas Abel. 2018. Resourceful Dependent Types. In *Presentation at 24th International Conference on Types for Proofs and Programs (TYPES 2018), Braga, Portugal.* http://www.cse.chalmers.se/~abela/types18.pdf abstract.

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, 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, 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

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

Jean-Philippe Bernardy and Patrik Jansson. 2025. Domain-specific tensor languages. *Journal of Functional Programming* 35 (2025), e9. https://doi.org/10.1017/S0956796825000048

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

Edwin C. Brady, Conor McBride, and James McKinna. 2003. Inductive Families Need Not Store Their Indices. In *Types for Proofs and Programs, International Workshop, TYPES 2003, Torino, Italy, April 30 - May 4, 2003, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 3085).* Springer, 115–129. https://doi.org/10.1007/978-3-540-24849-1_8

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

Thierry Coquand, Nils Anders Danielsson, Martín Escardó, Ulf Norell, and Chuangjie Xu. 2017. Negative consistent axioms can be postulated without loss of canonicity. (2017). https://martinescardo.github.io/papers/negative-axioms.pdf Note from Oct 2013, updated Oct 2017.

Oskar Eriksson, Nils Anders Danielsson, and Andreas Abel. 2025. *An Agda Formalization of Graded Modal Type Theory.* https://doi.org/10.5281/zenodo.15189791

Yannick Forster, Matthieu Sozeau, and Nicolas Tabareau. 2024. Verified Extraction from Coq to OCaml. *Proc. ACM Program. Lang.* 8, PLDI, Article 149 (June 2024), 24 pages. https://doi.org/10.1145/3656379

Yiyun Liu, Jonathan Chan, Jessica Shi, and Stephanie Weirich. 2024. Internalizing Indistinguishability with Dependent Types. *Proc. ACM Program. Lang.* 8, POPL, Article 44 (Jan. 2024), 28 pages. https://doi.org/10.1145/3632886

Yiyun Liu, Jonathan Chan, and Stephanie Weirich. 2025. Consistency of a Dependent Calculus of Indistinguishability. *Proc. ACM Program. Lang.* 9, POPL, Article 7 (Jan. 2025), 27 pages. https://doi.org/10.1145/3704843

Anton Lorenzen, Leo White, Stephen Dolan, Richard A. Eisenberg, and Sam Lindley. 2024. Oxidizing OCaml with Modal Memory Management. *Proc. ACM Program. Lang.* 8, ICFP, Article 253 (Aug. 2024), 30 pages. https://doi.org/10.1145/3674642

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

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

Matthieu Sozeau, Simon Boulier, Yannick Forster, Nicolas Tabareau, and Théo Winterhalter. 2019. Coq Coq correct! verification of type checking and erasure for Coq, in Coq. *Proc. ACM Program. Lang.* 4, POPL, Article 8 (Dec. 2019), 28 pages. https://doi.org/10.1145/3371076

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. 2024. *Agda User Manual, Release 2.7.0.1.* https://readthedocs.org/projects/agda/downloads/pdf/v2.7.0.1/

Cassia Torczon, Emmanuel Suárez Acevedo, Shubh Agrawal, Joey Velez-Ginorio, and Stephanie Weirich. 2024. Effects and Coeffects in Call-by-Push-Value. *Proc. ACM Program. Lang.* 8, OOPSLA2, Article 310 (Oct. 2024), 27 pages. https://doi.org/10.1145/3689750

Philip Wadler. 1990. Linear Types can Change the World!. In *Programming concepts and methods: Proceedings of the IFIP Working Group 2.2, 2.3 Working Conference on Programming Concepts and Methods, Sea of Galilee, Israel, 2-5 April, 1990*, Manfred Broy and Cliff B. Jones (Eds.). North-Holland, 561.

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

Part II

Appended Papers

Paper A

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

A. Abel, N. A. Danielsson, **O. Eriksson**

Proc. ACM Program. Lang. 7, ICFP, Article 220 (August 2023)

10.1145/3607862

The layout has been revised

Abstract

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.

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].
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.

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

[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] 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".

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.

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.

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

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 $[\![t]\!] : D^{p_1}[\![A_1]\!] \times \cdots \times D^{p_n}[\![A_n]\!] \to [\![B]\!]$ 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

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

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).

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

[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 $p = 0$.

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:

$$x \leq q + rx \tag{3.1}$$
$$x \leq p \tag{3.2}$$

- 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 $\circledast_r$:

$$\begin{aligned}
(p \wedge p') \circledast_r q &\leq (p \circledast_r q) \wedge (p' \circledast_r q) \\
p \circledast_r (q \wedge q') &\leq (p \circledast_r q) \wedge (p \circledast_r q') \\
(p \circledast_r p')q &\leq pq \circledast_r p'q \\
(p \circledast_r q) + (p' \circledast_r q') &\leq (p + p') \circledast_r (q + q')
\end{aligned}$$

- 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. (3.1) and (3.2) and the distributivity laws. The affine and linear types interpretations can also be combined using the four-element set $\{0, 1, 0\!\!1, \omega\}$, where 1 and $0\!\!1$ represent linear and affine uses, respectively. For this instance $p + q = \omega$ unless either p or q is 0, and multiplication is given by $0\!\!1 \cdot 0\!\!1 = 0\!\!1$ and $p \cdot q = \omega$ otherwise, with the obvious exceptions for 0 and 1. The partial order is the reflexive, transitive closure of $\omega \leq 0\!\!1$ and $1 \geq 0\!\!1 \leq 0$, and the largest solution to Eqs. (3.1) and (3.2) and the distributivity laws are given by $p \circledast_0 q = p \wedge q$ and $p \circledast_1 q = p + \omega q$ and $p \circledast_{0\!\!1} q = p \wedge \omega q$ 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 ($\bot$) 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 $\otimes$ 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 ($\mathsf{prodrec}$) and optionally erased matches (and Agda supports all of these variants).

[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.

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 $\mathsf{prodrec}$ which each bind one variable, as well as the third arguments u to natrec and $\mathsf{prodrec}$ 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 ($\uparrow$), for increasing variable indices by one, and lifting ($\Uparrow$), 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 $\mathsf{tail}\,(\sigma, t) = \sigma$.

$\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}_{\mathcal{M}}$ 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).

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

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

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

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

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

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

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

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

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

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* $(x :^p S) \otimes T$ 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 $\Sigma^q_k\, F\, G$ 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).

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

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

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

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

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

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

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

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

$$\frac{\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma \vdash t = t' : F \quad \Gamma \vdash u = u' : G[t]}{\Gamma \vdash (t, u)_k = (t', u')_k : \Sigma^q_k\, F\, G} \qquad \frac{\Gamma, \mathbb{N} \vdash A \quad \Gamma \vdash z : A[\mathsf{zero}] \quad \Gamma, \mathbb{N}, A \vdash s : A[\uparrow^2 \mathsf{id}, \mathsf{suc}\, x_1]}{\Gamma \vdash \mathsf{natrec}^q_{p,r}\, A\, z\, s\, \mathsf{zero} = z : A[\mathsf{zero}]}$$

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

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

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

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

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

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

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

$$\frac{\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma, \Sigma^{q'}_{\otimes}\, F\, G \vdash A \quad \Gamma, F, G \vdash u : A[\uparrow^2 \mathsf{id}, (x_1, x_0)_{\otimes}] \quad \Gamma \vdash t \longrightarrow t' : \Sigma^{q'}_{\otimes}\, F\, G}{\Gamma \vdash \mathsf{prodrec}^q_r\, A\, t\, u \longrightarrow \mathsf{prodrec}^q_r\, A\, t'\, u : A[t]}$$

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

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

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

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

The operational semantics of $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}_{\mathcal{M}}$ 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 Section 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, 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 $\mathsf{prodrec}$ both components are used, and the usage rule for the pair constructor accounts for both components.

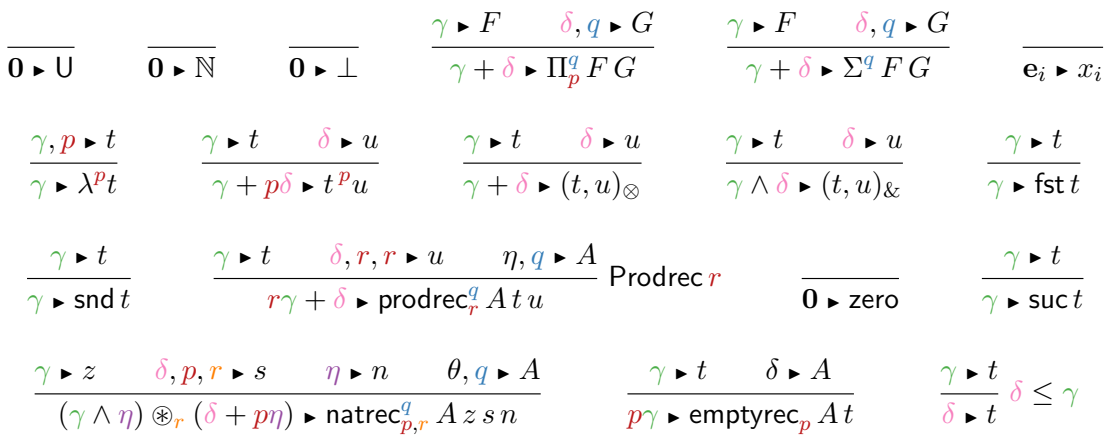

Figure 4: The grade assignment relation $\gamma \blacktriangleright t$.

The grade annotation r on $\mathsf{prodrec}^q_r\, t\, u$ 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 $\mathsf{prodrec}$, because type annotations do not contribute to computation. The configurable side condition $\mathsf{Prodrec}\, r$ 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, $\mathsf{emptyrec}_p\, A\, t$, is similar to $\mathsf{prodrec}$. 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 $\mathsf{prodrec}$, 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 Section 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 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 $\forall x_i.\, \mathbf{e}_i\Psi \triangleright x_i[\sigma]$.

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.

[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.

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

Proof. By induction on $\gamma \blacktriangleright t$. □

Note that for β-reductions, with $\gamma, p \blacktriangleright t$ and $\delta \blacktriangleright u$ and the matrix constructed as above, we get $\gamma + p\delta \blacktriangleright t[u]$, 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* $\gamma \blacktriangleright t$ *and* $\Gamma \vdash t \longrightarrow u : A$ *then* $\gamma \blacktriangleright u$.

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 Section B.

6 Erasure Case Study

In order to show our system in practice, we now turn our attention to a specific use case of $\lambda_{\mathcal{M}}^{\Sigma\Pi\mathbb{U}\mathbb{N}}$, 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 $0 \not\leq 1$. □

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

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

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 $\mathsf{id} := \lambda^0\lambda^\omega x_0$ 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):

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

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 $((\lambda^0\lambda^\omega x_0)\,{}^0\mathbb{N}\,{}^\omega\mathsf{zero})^\bullet = (\lambda\,\lambda\, x_0)\, \lightning\, \mathsf{zero}$.

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 $\mathsf{prodrec}^q_0$: $\mathsf{prodrec}$ does not reduce if applied to a neutral pair. We disallow matching on erased weak pairs in non-erased positions by requiring $\mathsf{Prodrec}\,0$ 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 Section 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 $\langle \Pi^q_p FG/\mathscr{F}/\mathscr{G} \rangle$ 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 Section 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 Section 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, $t \mathrel{®}_\ell v : A \,/\, \mathscr{A}$, 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 $t \mathrel{®}_\ell v : A \,/\, \mathscr{A}$ iff $\Delta_0 \vdash t : \mathsf{U}$.
- If $\mathscr{A} = \langle \mathbb{N}\rangle$ then $t \mathrel{®}_\ell v : A \,/\, \mathscr{A}$ iff $t \mathrel{®}_{\mathbb{N}} v$, 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 $u \mathrel{®}_{\mathbb{N}} w$.
- If $\mathscr{A} = \langle \Pi^q_\omega FG/\mathscr{F}/\mathscr{G}\rangle$ then $t \mathrel{®}_\ell v : A \,/\, \mathscr{A}$ iff $t\,{}^{\omega}a \mathrel{®}_\ell v\, w : G[a] \,/\, \mathscr{G}(a)$ for all a and w with $\Delta_0 \Vdash_\ell a : F \,/\, \mathscr{F}$ and $a \mathrel{®}_\ell w : F \,/\, \mathscr{F}$.
- If $\mathscr{A} = \langle \Pi^q_0 FG/\mathscr{F}/\mathscr{G}\rangle$ then $t \mathrel{®}_\ell v : A \,/\, \mathscr{A}$ iff $t\,{}^{0}a \mathrel{®}_\ell v\, \lightning : G[a] \,/\, \mathscr{G}(a)$ for all a with $\Delta_0 \Vdash_\ell a : F \,/\, \mathscr{F}$.
- If $\mathscr{A} = \langle \Sigma^q_k FG/\mathscr{F}/\mathscr{G}\rangle$ then $t \mathrel{®}_\ell v : A \,/\, \mathscr{A}$ iff there are terms t_1, t_2, v_1, v_2 such that
  - $\Delta_0 \vdash t \longrightarrow^* (t_1, t_2)_k : \Sigma^q_k\, F\, G$,
  - $v \longrightarrow^* (v_1, v_2)$,
  - $\Delta_0 \Vdash_\ell t_1 : F \,/\, \mathscr{F}$,
  - $t_1 \mathrel{®}_\ell v_1 : F \,/\, \mathscr{F}$, and
  - $t_2 \mathrel{®}_\ell v_2 : G[t_1] \,/\, \mathscr{G}(t_1)$.
- If $\mathscr{A} = \langle \mathrm{emb}/\mathscr{A}'\rangle$ then $t \mathrel{®}_\ell v : A \,/\, \mathscr{A}$ iff $t \mathrel{®}_0 v : A \,/\, \mathscr{A}'$.
- If $\mathscr{A} = \langle \bot\rangle$ or $\mathscr{A} = \langle \mathsf{Ne}\, K\rangle$ then $t \mathrel{®}_\ell v : A \,/\, \mathscr{A}$ 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 $\Delta \Vdash^s \sigma : \Gamma \,/\, \mathscr{T}$, 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 $\Gamma \Vdash^v_\ell A \,/\, \mathscr{T}$. This in turn means that A is reducible under any valid substitution. Again we refer to Section A for details.

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

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

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

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 $t' \mathrel{®}_\ell v' : A \,/\, \mathscr{A}$ then $t \mathrel{®}_\ell v : A \,/\, \mathscr{A}$.*

Proof. By induction on $t' \mathrel{®}_\ell v' : A \,/\, \mathscr{A}$. □

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

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

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 $\mathscr{A} :: \Gamma \Vdash^v_1 A \,/\, \mathscr{T}$ such that $\gamma \blacktriangleright \Gamma \Vdash_1 t : A \,/\, \mathscr{T} \,/\, \mathscr{A}$.*

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.

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

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

Theorem 6.11. *If* $\mathscr{T} :: \Vdash^v \Gamma$ *and* $\mathscr{S} :: \Delta_0 \Vdash^s \sigma : \Gamma \,/\, \mathscr{T}$ *then* $\sigma \mathrel{®_\ell} \sigma' : \Gamma \blacktriangleleft \mathbf{0} \,/\, \mathscr{T} \,/\, \mathscr{S}$ *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* $t \mathrel{®_1} t^\bullet : A \,/\, \mathscr{A}$.

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 $\mathsf{suc}\, t$ 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.

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

Theorem 6.13. *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* $t^\bullet \longrightarrow^{s*} \underline{n}$.

Proof. By Theorem 6.12 we have $t \mathrel{®_1} t^\bullet : \mathbb{N} \,/\, \langle \mathbb{N} \rangle$. 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 $p \circledast_r q \coloneqq \omega(p \wedge q)$ 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 $p \circledast_r q \coloneqq x_n$. 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, $p \circledast_r q \coloneqq r^*(p \wedge q)$ is a candidate. With this definition Equation (3.1) can easily be shown to hold. If in addition $p^* \leq 0$ or $p^* \leq 1$ hold for all p, then Equation (3.2) 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).

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 $t \coloneqq \mathsf{prodrec}^{0}_{0}\, \mathbb{N}\, x_0\, \mathsf{zero}$, 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.

[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.

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 $\mathsf{prodrec}_0^q\, A\, t\, u$ might not reduce even though $(\mathsf{prodrec}_0^q\, A\, t\, u)^\bullet$ 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):

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

The eliminator $\mathsf{unitrec}$ extends the pattern of $\mathsf{prodrec}$ 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:

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

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 $\mathbf{0} \triangleright \star_\&$, 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 $\gamma \triangleright \star_\&$ 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 $\gamma \triangleright t$ 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 $\mathsf{prodrec}$ 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].

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

$$\frac{\gamma \blacktriangleright^{m \odot p} F \qquad \delta, \overline{m} q \blacktriangleright^m G}{\gamma + \delta \blacktriangleright^m \Sigma^q_{k,p} \, F \, G} \qquad \frac{}{\overline{m}\mathbf{e}_i \blacktriangleright^m x_i} \qquad \frac{\gamma, \overline{m} p \blacktriangleright^m t}{\gamma \blacktriangleright^m \lambda^p t} \qquad \frac{\gamma \blacktriangleright^m t \qquad \delta \blacktriangleright^{m \odot p} u}{\gamma + p\delta \blacktriangleright^m t \, {}^p u}$$

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

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

$$\frac{\gamma \blacktriangleright^m t}{\gamma \blacktriangleright^m \mathsf{suc} \, t} \qquad \frac{\gamma \blacktriangleright^m z \qquad \delta, \overline{m} p, \overline{m} r \blacktriangleright^m s \qquad \eta \blacktriangleright^m n \qquad \theta, 0 \blacktriangleright^{0_\mathsf{M}} A}{(\gamma \wedge \eta) \circledast_r (\delta + p\eta) \blacktriangleright^m \mathsf{natrec}^q_{p,r} \, A \, z \, s \, n}$$

$$\frac{\gamma \blacktriangleright^{m \odot p} t \qquad \delta \blacktriangleright^{0_\mathsf{M}} A}{p\gamma \blacktriangleright^m \mathsf{emptyrec}_p \, A \, t} \qquad \frac{\gamma \blacktriangleright^m t}{\delta \blacktriangleright^m t} \; \delta \leq \gamma$$

Figure 7: Moded grading $\gamma \blacktriangleright^m t$. New variable occurrences are highlighted in colour.

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 $t \, {}^0 u$.

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 used in the rule for $\mathsf{prodrec}$: 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 $\mathsf{prodrec}$, natrec and $\mathsf{emptyrec}$ still include assumptions about the motives ("A"), but now these assumptions use the mode 0_M, and in the case of $\mathsf{prodrec}$ 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):

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

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 $\mathsf{prodrec}$ 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.

Acknowledgments

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, $\Gamma \Vdash_{\ell} A$, $\Gamma \Vdash_{\ell} t : A \,/\, \mathscr{A}$, $\Gamma \Vdash_{\ell} A = B \,/\, \mathscr{A}$, and $\Gamma \Vdash_{\ell} t = u : A \,/\, \mathscr{A}$, for reducible types, terms, equality of reducible types, and equality of reducible terms, respectively. All relations

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

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

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

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

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

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 $\mathsf{Ne}\, t$ 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, $\Gamma \Vdash_\ell t : A / \mathscr{A}$, equality of reducible types, $\Gamma \Vdash_\ell A = B / \mathscr{A}$, and equality of reducible terms, $\Gamma \Vdash_\ell t = u : A \,/\, \mathscr{A}$, all of which are defined by recursion on $\mathscr{A}$, a proof of the reducibility of 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 $\mathscr{A} = \langle \Pi^q_p F G / \mathscr{F} / \mathscr{G} \rangle$ 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 $\mathscr{A} = \langle \Sigma^q_{\&} F G / \mathscr{F} / \mathscr{G} \rangle$ 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 $u = (u_1, u_2)$,
  - $\Gamma \Vdash_\ell \mathsf{fst}\, u : F[\mathsf{id}] \,/\, \mathscr{F}(\mathsf{id})$, and
  - $\Gamma \Vdash_\ell \mathsf{snd}\, u : (G[\Uparrow\mathsf{id}])[\mathsf{fst}\, u] \,/\, \mathscr{G}(\Uparrow\mathsf{id}, \mathsf{fst}\, u)$.
- If $\mathscr{A} = \langle \Sigma^q_{\otimes} F G / \mathscr{F} / \mathscr{G} \rangle$ 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 $u = (u_1, u_2)$,
  - $\Gamma \Vdash_\ell u_1 : F[\mathsf{id}] \,/\, \mathscr{F}(\mathsf{id})$ and
  - $\Gamma \Vdash_\ell u_2 : (G[\Uparrow\mathsf{id}])[u_1] \,/\, \mathscr{G}(\Uparrow\mathsf{id}, u_1)$.
- 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 $\Gamma \Vdash_\ell A = B \,/\, \mathscr{A}$, which is defined by recursion on $\mathscr{A}$ as follows:

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

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 $\Gamma \Vdash_\ell t = u : A \,/\, \mathscr{A}$, which is defined by recursion on $\mathscr{A}$ as follows:

- If $\mathscr{A} = \langle \mathsf{U} \rangle$ then $\Gamma \Vdash_\ell t = u : A \,/\, \mathscr{A}$ 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
  - $\Gamma \Vdash_\ell t = u \,/\, \mathscr{T}$.
- If $\mathscr{A} = \langle \mathbb{N} \rangle$ then $\Gamma \Vdash_\ell t = u : A \,/\, \mathscr{A}$ 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 $\Gamma \Vdash_\ell t = u : A \,/\, \mathscr{A}$ 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 $\Gamma \Vdash_\ell t = u : A \,/\, \mathscr{A}$ 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 $\mathscr{A} = \langle \mathrm{emb}/\mathscr{A}' \rangle$ 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 $\mathsf{prodrec}$ 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 $\mathsf{prodrec}$ 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:

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

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 $\gamma \triangleright t$ for some γ, then $|t| \triangleright t$ and $\gamma \leq |t|$.*

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

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

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

Proof. A lemma showing that either $|t| \triangleright t$, or that $\gamma \triangleright t$ does not hold for any γ, can be shown by induction on t. Decidability of $\gamma \triangleright t$ 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 $|x_i[\sigma]|$.

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. □

Bibliography

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

Paper B

A Resource-Correct Graded Modal Dependent Type Theory with Recursion, Formalized

O. Eriksson, N. A. Danielsson, A. Abel

Extended version of paper submitted for review

Preliminary version

Abstract

We present a graded modal type theory and prove that it handles resources correctly, in the sense that an abstract machine accesses resources the "correct" number of times. The theory is parametrized, and can for instance be instantiated with modalities for erasure, linear types, or affine types. Compared to previous work our theory supports what we believe to be a unique combination of features: (1) a flexible eliminator for natural numbers that enables different resource-usage patterns and works correctly in a setting with linear types, (2) an eliminator for empty types that allows the eliminated argument to be erased, supporting the use of erased information to discard impossible branches, and (3) universes that allow the definition of types using (for instance) recursive functions. The paper is accompanied by an Agda formalization.

1 Introduction

In graded modal type theories the type system is equipped with an algebraic structure, typically some form of semiring, containing *grades*. Grades can be used for data privacy [Choudhury et al., 2022; Volpano et al., 1996; Moon et al., 2021], for memory safety [Bernardy et al., 2017; Lorenzen et al., 2024], to enforce correctness (for instance of communication protocols) [Brady, 2021; Bernardy and Jansson, 2025], and for quantitative aspects like erasure, linear types and affine types [McBride, 2016; Atkey, 2018; Abel et al., 2023b; Choudhury et al., 2021; Moon et al., 2021; Abel and Bernardy, 2020; Brunel et al., 2014; Ghica and Smith, 2014].

In the case of quantitative aspects one might like to ensure that variables are referenced the "right" number of times during evaluation, as specified by an operational semantics. This approach is taken for instance by Choudhury et al. [2021], who define a heap-based abstract machine with the capability to count heap lookups, corresponding to variable uses, and prove a notion of correctness.

Following Choudhury et al. we present such an argument for a graded modal type theory based on the work of Abel et al. [2023b]. Compared to previous work our theory supports what we believe to be a unique combination of features:

Recursion. We support a flexible eliminator for natural numbers, see Section 4. We say that it is flexible because it supports multiple usage patterns. For instance, when the linearity modality is used one can define functions that are linear in the natural number, but it is also possible to track linear uses of variables in the zero and successor branches (if only erased recursion takes place).

Abel et al. [2023b] present a similar eliminator, but as we discuss this eliminator is problematic in a setting with linear types. We present an alternative approach and prove its correctness. We view this as a step towards a more general theory of recursive data types in graded type systems, and in Section 6 we sketch how the proposed method might be adapted to other inductive types. Other graded type theories with support for recursion for natural numbers, or other recursive types, are discussed in Section 7.

Empty type. Following Abel et al. [2023b] we include an eliminator for empty types that allows the eliminated argument to be erased. Such an eliminator supports the use of erased information to discard impossible branches. Due to the presence of this eliminator we make use of the *consistency* of the type theory in our correctness proof. Without consistency, the eliminator could be used to construct a function which is considered to be linear but does not use its argument during evaluation.

Universes We support universes that allow the definition of types using (for instance) recursive functions, building on the work of Abel et al. [2017]. Note that the inclusion of universes tends to complicate the meta-theory.

Our correctness proof is mechanized in Agda. We believe that this is the first complete formalization of correctness for linearity for a graded type theory with dependent types: the work of Choudhury et al. [2021] is partly formalized.

Before we present our approach to the eliminator for natural numbers in Section 4 we present the type theory—without the eliminator—in Section 2, and in Section 3 we present an abstract machine for that theory, along with a correctness proof.

1.1 Formalization

With the exception of Section 6 this work is fully formalized in Agda, building on the formalization of graded type theory by Abel et al. [2023a]. Our formalization is not directly based on the formalization accompanying their paper, but rather on a more recent version to which some additional contributions have been made.[1]

The formalization contains the file `README.agda` which lists our definitions, theorems and claims in the order they appear in the paper, linking to their formalized counterparts. Where applicable, the file also includes some discussion about how the formalized statement relates to the version in the paper, in particular if there are differences between the formalization and the paper. The paper provides links to an HTML rendering of this file for each entry. We mark such links in blue.

In some regards the presentation in the paper differs from the formalization. Notably, the formalized syntax is well-scoped. Several definitions and properties therefore have hidden side conditions or assumptions enforcing well-scopedness which are left out of the paper for the sake of readability. Another notable difference is that the formalization supports identity types. We have chosen to exclude them from the paper but our results hold also when they are included. We refer the interested reader to the formalization for details [Eriksson et al., 2025]. These differences, as well as others, are explained further in `README.agda`.

2 A Graded Modal Dependent Type Theory

We base our work on the graded modal type theory by Abel et al. [2023b] and its formalization. In this section we will provide some background by describing the key parts of the language. Readers who are already familiar with this work can safely skip this section. Conversely, we refer the reader who is interested in more detail to the formalization [Eriksson et al., 2025] or Abel et al. [2023b] (but note that, as mentioned, there have been some additions to the formalization since their work). The description we give here will be limited to the fragment that we cover in Section 3. In particular, it does not include the eliminator for natural numbers, which we will return to in Section 4, nor identity types, which are excluded for brevity.

2.1 Modalities

One key aspect of the language is the modalities. Modalities are semiring-like structures whose elements (called grades) are used to annotate certain terms of the language. For our purposes, we will primarily be interested in modalities with a quantitative interpretation, allowing us to express how many times some resource will be used when the program is evaluated. We restate the definition of modalities from Abel et al. [2023b] here, reformulated slightly:

Definition 2.1. A modality structure is a 6-tuple $(\mathcal{M}, +, \cdot, \wedge, 0, 1)$ consisting of (from left to right) a type $\mathcal{M}$, three binary operations (addition, multiplication and meet) 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 distributes over addition. As usual, we will often abbreviate multiplication of p and q as pq.

[1] Some notable additions: An identity type by Nils Anders Danielsson, support for two kinds of unit types, matching the two kinds of Σ-types, by Oskar Eriksson (Abel et al. [2023b] support one unit type), and a hierarchy of universes by Nils Anders Danielsson and Ondřej Kubánek.

- $(\mathcal{M}, \wedge)$ forms a semilattice: Meet is commutative, associative and idempotent.
- Both multiplication and addition distribute over meet.

In comparison with the definition of Abel et al. [2023b] we have omitted the operator $\circledast$ used to assign grades to the eliminator for natural numbers. We will return to this in Section 4. The definition used in the formalization requires modalities to come with an additional special grade, satisfying some properties, which is used to assign grades to the eliminators for identity types. Since we do not discuss identity types, we have omitted this here.

For our purposes, we may think of grades as representing a *quantity*—the number of times some resource, like a function argument, is used. In this regard, a useful mental picture is that of grades representing a set of natural numbers corresponding to how many times some resource should be used.[2] This view uses sets of numbers rather than single numbers because grades do not necessarily represent an exact use count. Indeed, we will often let our modalities include the grade ω, representing any number of uses which is represented by the set of all natural numbers in this view.

From the perspective of grades representing quantities, the grades 0 and 1 represent *no uses* and a *single use* of some resource, respectively. Addition represents the combined resource usage from different places (usually from two sub-terms) while multiplication represents a single resource being used several times (the main example of this is application). Meet is used both directly and indirectly via the partial order that is induced by the semilattice structure: $p \leq q$ iff $p = p \wedge q$. We interpret this relation as a form of *compatibility* between grades, in the sense that if $p \leq q$ then a resource that is used q times may alternatively be thought of as being used p times (see the example modalities below for more concrete examples of this). With this interpretation, the meet operator can be thought of as representing the usage count of a branching computation. If one branch uses some resource p times and the other q times then $p \wedge q$ will give a usage count that is compatible with either branch (since $p \wedge q$ is smaller than or equal to both p and q). In the view of modalities as sets of natural numbers, the partial order represents the superset relation—a grade is compatible with another if its allowed usage counts are fully included in the other.

In our resource correctness proof we will make use of some additional properties. The purpose of these is to give 0 its intended quantitative interpretation of representing no uses of some resource, in particular that resources cannot "disappear" by adding or scaling them. The properties we use coincide with those for modalities with a *well-behaved zero* as defined by Abel et al. [2023b], and those are in turn based on similar assumptions used in QTT [McBride, 2016; Atkey, 2018]. We restate their definition of modalities with a well-behaved zero here (we have reformulated it slightly):

Definition 2.2. A modality is said to have a well-behaved zero if it satisfies the following properties:

- If $p + q = 0$ then $p = 0$ and $q = 0$.
- If $pq = 0$ then either $p = 0$ or $q = 0$.
- If $p \wedge q = 0$ then $p = 0$ and $q = 0$.
- $1 \neq 0$.

Below we will only consider modalities with a well-behaved zero. We also assume that equality with zero is decidable.

Let us now consider some concrete modalities [McBride, 2016] and their intended interpretations. The two element modality $\{0, \omega\}$ with 0 as the multiplicative zero and ω as the unit, $\omega + \omega = \omega$ and $\omega \leq 0$ allows an interpretation of erasure. The three element

[2]Note that this should only serve as a form of intuition for modalities in a quantitative sense. It is not the case that sets of natural numbers form modalities in general.

modality $\{0, 1, \omega\}$ with 0 and 1 as zero and unit, respectively, and $1 + 1 = 1 + \omega = \omega + \omega = \omega\omega = \omega$ gives a choice for the partial order. We will refer to the instance with $\omega \leq 1 \leq 0$ as the affine types modality and the instance with $0 \geq \omega \leq 1$ with 0 and 1 incomparable as the linear types modality. The difference between these is how the grade 1 is interpreted. In the former case, 1 is compatible with 0, allowing it to be thought of as representing either 1 or 0 uses, whereas in the latter case, 1 is not compatible with any other grade so it represents exactly one use. Both these instances can be generalized to include grades for all natural numbers. Addition and multiplication then becomes the usual addition and multiplication for natural numbers and $\omega + n = \omega$ and $\omega n = \omega$ if $n \neq 0$ (and 0 otherwise). The partial order can be chosen such that it becomes the usual order for natural numbers, but flipped, or with all natural number grades being incomparable with each other with ω as the least grade in both cases. In the former case the interpretation of grade n becomes "at most n uses" while the latter case gives "exactly n uses", analogous to the affine and linear instances respectively. We will refer to these modalities to provide concrete examples, most notably the linearity and affine types instances.

2.2 Syntax, Typing and Usage

The language we are considering is a dependently typed language with Π- and Σ-types, a natural number type, unit and empty types as well as a hierarchy of Russell universes. The syntax is given in Fig. 1. Matching the formalization, we use a nameless representation of variables as de Bruijn indices, writing x_i for the variable with index i. There are two different kinds of Σ and unit types, *weak* with a form of pattern matching (using $\mathsf{prodrec}$ and $\mathsf{unitrec}$) and *strong* with projections and η-equality. We keep the two types separate because of their different behaviour in terms of resource usage, as we discuss further below. We therefore label Σ and unit types as well as pairs and unit elements with $\otimes$ or & to indicate the weak and strong versions, respectively. In cases where no distinction is needed, we may omit the label.

Some terms are annotated by grades indicating, for modalities with a quantitative interpretation, how many times some resource should be used. For lambdas and pairs, the grade represents how many times the function argument is used and the number of times the first component should be used, respectively. Their eliminators—applications, projections and $\mathsf{prodrec}$—each have a corresponding annotation. Π_p^q- and Σ_p^q-types have two annotations, with p having the same meaning as the grade on lambdas or pairs, and q instead representing the number of times the function argument is used in the codomain or how many times the first component is used in type of the second component. In addition to the grade already mentioned, the eliminator for $\Sigma_\otimes$-types, $\mathsf{prodrec}_r^p$ has one more annotation, r, which represents how many times the components of the pair is used by u, similarly to how the annotation on application represents how many times the function argument is used by the function. Similar annotations are found on the eliminators for the weak unit type ($\mathsf{unitrec}_p$) and the empty type ($\mathsf{emptyrec}_p$), indicating how many times their arguments are used.

Weakenings are defined in Fig. 1. They act on indices in the following way:

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

The identity weakening id leaves variable indices unchanged, $\uparrow\rho$ applies ρ and then increases all indices by one, and $\Uparrow\rho$ leaves 0 unchanged and applies ρ to all other indices and increases them by one. We write $i[\rho]$ or $t[\rho]$ for the application of weakenings to indices or terms and $\uparrow^n\rho$ and $\Uparrow^n\rho$ for multiple applications of $\uparrow$ and $\Uparrow$.

Substitutions are maps from variable indices to terms and we write $t[\sigma]$ also for the application of a substitution σ to t. Given a substitution σ we will write σ, t for the

$$
\begin{array}{rcll}
i & \in & \mathsf{Nat} & \text{de Bruijn index} \\
l & \in & \mathsf{Nat} & \text{Universe level} \\
p, q, r & \in & \mathcal{M} & \text{Grade} \\
s & ::= & \otimes \mid \& & \Sigma\text{-type strength}
\end{array}
\qquad
\begin{array}{rcll}
\rho & ::= & \mathsf{id} \mid \uparrow\rho \mid \Uparrow\rho & \text{Weakening} \\
\Gamma, \Delta & ::= & \epsilon \mid \Gamma.\, A & \text{Typing Context} \\
\gamma, \delta & ::= & \epsilon \mid \gamma.\, p & \text{Grade Context} \\
m & ::= & 0 \mid 1 & \text{Mode}
\end{array}
$$

$$
\begin{array}{rcll}
A, B, t, u & ::= & \mathsf{x}_i \mid \lambda^p t \mid t^p u \mid (^p t, u)_s \mid \mathsf{fst}_p\, t \mid \mathsf{snd}_p\, t \mid \mathsf{prodrec}^p_r\, A\, t\, u & \text{Term} \\
& \mid & \star_s \mid \mathsf{unitrec}_p\, A\, t\, u \mid \mathsf{emptyrec}_p\, A\, t \mid \mathsf{zero} \mid \mathsf{suc}\, t & \\
& \mid & \mathsf{U}_l \mid \Pi^q_p\, A\, B \mid \Sigma^q_{s,p}\, A\, B \mid \top_s \mid \bot \mid \mathbb{N} & \text{Type formers}
\end{array}
$$

Figure 1: The syntax of the language as well as weakenings, typing and usage contexts, and modes.

substitution that replaces 0 with t and the remaining indices by σ (after re-indexing). For the common case of only substituting x_0 with a term u we will write $t[u]$ instead of $t[\mathsf{id}, u]$. We may also compose substitutions (or substitutions and weakenings) in the usual way, $t[\sigma \circ \sigma'] = (t[\sigma'])[\sigma]$. The weakening operators are overloaded for substitutions.

The type system consists of five judgments: $\vdash \Gamma$ for well-formed contexts, $\Gamma \vdash A$ and $\Gamma \vdash t : A$ for well-formed types and terms, and $\Gamma \vdash A = B$ and $\Gamma \vdash t = u : A$ for equality of types and terms. Contexts Γ, Δ are snoc-lists of types, see Fig. 1. The definition of the typing judgments, shown in Figs. 2 and 3, should be unsurprising except for the inclusion of grades: typing ensures that the annotations on constructors match the annotations on the corresponding eliminators (for instance, $(\lambda^p t)^q u$ is well-typed only if $p = q$).

Typing does not ensure that assigned grades are correct (i.e. that resources are used as many times as indicated). This is instead handled through the *usage relation*, $\gamma \blacktriangleright^m t$, defined in Fig. 4. It checks that t is well-resourced under grade context γ which assigns a grade to each free variable in t, corresponding to the number of times it should be used (when a single copy of t is evaluated). Similarly to typing contexts, grade contexts are lists of grades, see Fig. 1. We write $\gamma(i)$ for the grade γ associates to i. We define addition, $\gamma + \delta$, and meet $\gamma \wedge \delta$ between contexts of the same size by pointwise application of the respective operator. Similarly, scaling, $p \cdot \gamma$ of a context by a grade is defined by pointwise multiplication from the left and we also lift the order relation, $\gamma \leq \delta$, pointwise to contexts.

We are using the extended usage relation from Abel et al. [2023b, Section 8] which checks that t is well-resourced at a given *mode* m which can be either 0 or 1. Mode 1 is in some sense the default mode, representing computationally relevant settings while mode 0 represents erased settings and is used to check terms with no computational relevancy such as the function argument of an erased application. Intuitively, no variable uses should be counted in this setting since the term will not be evaluated, and we may also be a bit more permissive in the erased mode, owing again to the idea that erased terms are not evaluated. We can convert grades into modes with $\overline{p} = 0$ iff $p = 0$ and 1 otherwise. In the other direction we implicitly convert mode 0 to grade 0 and mode 1 to grade 1.

For quantitative modalities we may think of the usage relation as counting how many times resources, or variables, are used [Abel et al., 2023b]. To that end, a variable x_i is well-resourced with respect to the context $m\boldsymbol{e}_i$ which maps index i to m and everything else to 0. In other words, this assigns grade 1 to x_i under mode 1 and grade 0 under mode 0. For the constructors, or values, the general principle is that a usage context is assigned to each sub-term, using a corresponding grade annotation to assign a grade to any free variables. The contexts of all sub-terms are then scaled by a grade corresponding to how many times it should be used and then added together. In particular, this means that

$\boxed{\mathsf{x}_i : A \in \Gamma}$

$$\frac{}{\mathsf{x}_0 : A[\uparrow \mathsf{id}] \in \Gamma.\, A} \qquad \frac{\mathsf{x}_i : A \in \Gamma}{\mathsf{x}_{i+1} : A[\uparrow \mathsf{id}] \in \Gamma.\, B}$$

$\boxed{\vdash \Gamma}$

$$\frac{}{\vdash \epsilon} \qquad \frac{\Gamma \vdash A}{\vdash \Gamma.\, A}$$

$\boxed{\Gamma \vdash A}$

$$\frac{\vdash \Gamma}{\Gamma \vdash \mathsf{U}_l} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \mathbb{N}} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \bot} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \top_s}$$

$$\frac{\Gamma.\, A \vdash B}{\Gamma \vdash \Pi^q_p\, A\, B} \qquad \frac{\Gamma.\, A \vdash B}{\Gamma \vdash \Sigma^q_{s,p}\, A\, B} \qquad \frac{\Gamma \vdash A : \mathsf{U}_l}{\Gamma \vdash A}$$

$\boxed{\Gamma \vdash t : A}$

$$\frac{\vdash \Gamma}{\Gamma \vdash \mathsf{U}_l : \mathsf{U}_{l+1}} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \mathbb{N} : \mathsf{U}_0} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \bot : \mathsf{U}_0} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \top_s : \mathsf{U}_0}$$

$$\frac{\Gamma \vdash A : \mathsf{U}_{l_1} \qquad \Gamma.\, A \vdash B : \mathsf{U}_{l_2}}{\Gamma \vdash \Pi^q_p\, A\, B : \mathsf{U}_{l_1 \sqcup l_2}} \qquad \frac{\Gamma \vdash A : \mathsf{U}_{l_1} \qquad \Gamma.\, A \vdash B : \mathsf{U}_{l_2}}{\Gamma \vdash \Sigma^q_{s,p}\, A\, B : \mathsf{U}_{l_1 \sqcup l_2}}$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash A = B}{\Gamma \vdash t : B} \qquad \frac{\vdash \Gamma \qquad \mathsf{x}_i : A \in \Gamma}{\Gamma \vdash \mathsf{x}_i : A} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \mathsf{zero} : \mathbb{N}} \qquad \frac{\Gamma \vdash t : \mathbb{N}}{\Gamma \vdash \mathsf{suc}\, t : \mathbb{N}}$$

$$\frac{\Gamma.\, A \vdash B \qquad \Gamma.\, A \vdash t : B}{\Gamma \vdash \lambda^p t : \Pi^q_p\, A\, B} \qquad \frac{\Gamma.\, A \vdash B \qquad \Gamma \vdash t : A \qquad \Gamma \vdash u : B[t]}{\Gamma \vdash ({}^p t, u)_s : \Sigma^q_{s,p}\, A\, B}$$

$$\frac{\Gamma \vdash t : \Pi^q_p\, A\, B \qquad \Gamma \vdash u : A}{\Gamma \vdash t^p u : B[u]} \qquad \frac{\Gamma.\, A \vdash B \qquad \Gamma \vdash t : \Sigma^q_{\&,p}\, A\, B}{\Gamma \vdash \mathsf{fst}_p\, t : A}$$

$$\frac{\Gamma.\, A \vdash B \qquad \Gamma \vdash t : \Sigma^q_{\&,p}\, A\, B}{\Gamma \vdash \mathsf{snd}_p\, t : B[\mathsf{fst}_p\, t]} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash t : \bot}{\Gamma \vdash \mathsf{emptyrec}_p\, A\, t : A}$$

$$\frac{\Gamma.\, \Sigma^q_{\otimes,p}\, A\, B \vdash C \qquad \Gamma \vdash t : \Sigma^q_{\otimes,p}\, A\, B \qquad \Gamma.\, A.\, B \vdash u : C[\uparrow^2 \mathsf{id}, ({}^p \mathsf{x}_1, \mathsf{x}_0)_\otimes]}{\Gamma \vdash \mathsf{prodrec}^p_r\, C\, t\, u : C[t]}$$

$$\frac{\vdash \Gamma}{\Gamma \vdash \star_s : \top_s} \qquad \frac{\Gamma.\, \top_\otimes \vdash A \qquad \Gamma \vdash t : \top_\otimes \qquad \Gamma \vdash u : A[\star_\otimes]}{\Gamma \vdash \mathsf{unitrec}_p\, A\, t\, u : A[t]}$$

Figure 2: Well-formed variables, $\mathsf{x}_i : A \in \Gamma$, contexts, $\vdash \Gamma$, types, $\Gamma \vdash A$, and terms, $\Gamma \vdash t : A$. Here $l_1 \sqcup l_2$ denotes the maximum of l_1 and l_2.

$\boxed{\Gamma \vdash A = B}$ $\quad$ $\boxed{\Gamma \vdash t = u : A}$

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

$$\frac{\Gamma \vdash A = A' \qquad \Gamma.\, A \vdash B = B'}{\Gamma \vdash \Pi^q_p\, A\, B = \Pi^q_p\, A'\, B'} \qquad \frac{\Gamma \vdash A = A' \qquad \Gamma.\, A \vdash B = B'}{\Gamma \vdash \Sigma^q_{s,p}\, A\, B = \Sigma^q_{s,p}\, A'\, B'}$$

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

$$\frac{\begin{array}{c}\Gamma \vdash A = A' : \mathsf{U}_{l_1} \\ \Gamma.\, A \vdash B = B' : \mathsf{U}_{l_2}\end{array}}{\Gamma \vdash \Pi^q_p\, A\, B = \Pi^q_p\, A'\, B' : \mathsf{U}_{l_1 \sqcup l_2}} \qquad \frac{\begin{array}{c}\Gamma \vdash A = A' : \mathsf{U}_{l_1} \\ \Gamma.\, A \vdash B = B' : \mathsf{U}_{l_2}\end{array}}{\Gamma \vdash \Sigma^q_{s,p}\, A\, B = \Sigma^q_{s,p}\, A'\, B' : \mathsf{U}_{l_1 \sqcup l_2}}$$

$$\frac{\Gamma \vdash t = u : \mathbb{N}}{\Gamma \vdash \mathsf{suc}\, t = \mathsf{suc}\, u : \mathbb{N}} \qquad \frac{\Gamma \vdash t : \top_{\&} \qquad \Gamma \vdash u : \top_{\&}}{\Gamma \vdash t = u : \top_{\&}} \qquad \frac{\begin{array}{c}\Gamma \vdash t = t' : \Pi^q_p\, A\, B \\ \Gamma \vdash u = u' : A\end{array}}{\Gamma \vdash t^p u = t'^p u' : B[u]}$$

$$\frac{\Gamma.\, A \vdash B \qquad \Gamma.\, A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda^p t)^p u = t[u] : B[u]} \qquad \frac{\begin{array}{c}\Gamma.\, A \vdash B \qquad \Gamma \vdash t : \Pi^q_p\, A\, B \qquad \Gamma \vdash u : \Pi^q_p\, A\, B \\ \Gamma.\, A \vdash t[{\uparrow}\mathsf{id}]^p \mathsf{x}_0 = u[{\uparrow}\mathsf{id}]^p \mathsf{x}_0 : B\end{array}}{\Gamma \vdash t = u : \Pi^q_p\, A\, B}$$

$$\frac{\Gamma.\, A \vdash B \qquad \Gamma \vdash t = t' : A \qquad \Gamma \vdash u = u' : B[t]}{\Gamma \vdash ({}^p t, u)_s = ({}^p t', u')_s : \Sigma^q_{s,p}\, A\, B} \qquad \frac{\begin{array}{c}\Gamma.\, A \vdash B \qquad \Gamma \vdash t : A \\ \Gamma \vdash u : B[t]\end{array}}{\Gamma \vdash \mathsf{fst}_p\, ({}^p t, u)_{\&} = t : A}$$

$$\frac{\begin{array}{c}\Gamma.\, A \vdash B \qquad \Gamma \vdash t : \Sigma^q_{\&,p}\, A\, B \qquad \Gamma \vdash u : \Sigma^q_{\&,p}\, A\, B \\ \Gamma \vdash \mathsf{fst}_p\, t = \mathsf{fst}_p\, u : A \qquad \Gamma \vdash \mathsf{snd}_p\, t = \mathsf{snd}_p\, u : B[\mathsf{fst}_p\, t]\end{array}}{\Gamma \vdash t = u : \Sigma^q_{\&,p}\, A\, B} \qquad \frac{\begin{array}{c}\Gamma.\, A \vdash B \\ \Gamma \vdash t = u : \Sigma^q_{\&,p}\, A\, B\end{array}}{\Gamma \vdash \mathsf{fst}_p\, t = \mathsf{fst}_p\, u : A}$$

$$\frac{\Gamma.\, A \vdash B \qquad \Gamma \vdash t : A \qquad \Gamma \vdash u : B[t]}{\Gamma \vdash \mathsf{snd}_p\, ({}^p t, u)_{\&} = u : B[\mathsf{fst}_p\, ({}^p t, u)_{\&}]} \qquad \frac{\Gamma.\, A \vdash B \qquad \Gamma \vdash t = u : \Sigma^q_{\&,p}\, A\, B}{\Gamma \vdash \mathsf{snd}_p\, t = \mathsf{snd}_p\, u : B[\mathsf{fst}_p\, t]}$$

$$\frac{\begin{array}{c}\Gamma.\, \top_{\otimes} \vdash A = A' \qquad \Gamma \vdash t = t' : \top_{\otimes} \\ \Gamma \vdash u = u' : A[\star_{\otimes}]\end{array}}{\Gamma \vdash \mathsf{unitrec}_p\, A\, t\, u = \mathsf{unitrec}_p\, A'\, t'\, u' : A[t]} \qquad \frac{\Gamma \vdash A = B \qquad \Gamma \vdash t = u : \bot}{\Gamma \vdash \mathsf{emptyrec}_p\, A\, t = \mathsf{emptyrec}_p\, B\, u : A}$$

$$\frac{\begin{array}{c}\Gamma.\, \Sigma^{q'}_{\otimes,p}\, A\, B \vdash C \qquad \Gamma \vdash t : A \qquad \Gamma \vdash t' : B[t] \\ \Gamma.\, A.\, B \vdash u : C[{\uparrow}^2\mathsf{id}, ({}^p \mathsf{x}_1, \mathsf{x}_0)_{\otimes}]\end{array}}{\Gamma \vdash \mathsf{prodrec}^p_r\, C\, ({}^p t, t')_{\otimes}\, u = u[t, t'] : C[({}^p t, t')_{\otimes}]} \qquad \frac{\Gamma.\, \top_{\otimes} \vdash A \qquad \Gamma \vdash u : A[\star_{\otimes}]}{\Gamma \vdash \mathsf{unitrec}_p\, A\ \star_{\otimes}\ u = u : A[\star_{\otimes}]}$$

$$\frac{\begin{array}{c}\Gamma.\, \Sigma^q_{\otimes,p}\, A\, B \vdash C = C' \qquad \Gamma \vdash t = t' : \Sigma^q_{\otimes,p}\, A\, B \\ \Gamma.\, A.\, B \vdash u = u' : C[{\uparrow}^2\mathsf{id}, ({}^p \mathsf{x}_1, \mathsf{x}_0)_{\otimes}]\end{array}}{\Gamma \vdash \mathsf{prodrec}^p_r\, C\, t\, u = \mathsf{prodrec}^p_r\, C'\, t'\, u' : C[t]} \qquad \frac{\Gamma \vdash t = u : A \qquad \Gamma \vdash A = B}{\Gamma \vdash t = u : B}$$

Figure 3: Equality of types, $\Gamma \vdash A = B$ and terms $\Gamma \vdash t = u : A$.

$\boxed{\gamma \blacktriangleright^m t}$

$$\frac{}{\mathbf{0} \blacktriangleright^m \mathsf{U}_l} \qquad \frac{}{\mathbf{0} \blacktriangleright^m \mathbb{N}} \qquad \frac{}{\mathbf{0} \blacktriangleright^m \top_s} \qquad \frac{}{\mathbf{0} \blacktriangleright^m \bot} \qquad \frac{\gamma \blacktriangleright^{m\overline{p}} A \qquad \delta.\, mq \blacktriangleright^m B}{\gamma + \delta \blacktriangleright^m \Pi^q_p\, A\, B}$$

$$\frac{\gamma \blacktriangleright^{m\overline{p}} A \qquad \delta.\, mq \blacktriangleright^m B}{\gamma + \delta \blacktriangleright^m \Sigma^q_{s,p}\, A\, B} \qquad \frac{}{m\boldsymbol{e}_i \blacktriangleright^m \mathsf{x}_i} \qquad \frac{\gamma.\, mp \blacktriangleright^m t}{\gamma \blacktriangleright^m \lambda^p t} \qquad \frac{\gamma \blacktriangleright^m t \qquad \delta \blacktriangleright^{m\overline{p}} u}{\gamma + p\delta \blacktriangleright^m t^p u}$$

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

$$\frac{\gamma \blacktriangleright^{m\overline{p}} t \qquad \delta \blacktriangleright^m u}{p\gamma + \delta \blacktriangleright^m ({}^p t, u)_{\otimes}} \qquad \frac{\gamma \blacktriangleright^{m\overline{r}} t \qquad \delta.\, mrp.\, mr \blacktriangleright^m u \qquad \eta.\, 0 \blacktriangleright^0 A}{r\gamma + \delta \blacktriangleright^m \mathsf{prodrec}^p_r\, A\, t\, u}\ \mathsf{Prodrec}\, m\, r\, p$$

$$\frac{}{\mathbf{0} \blacktriangleright^m \mathsf{zero}} \qquad \frac{\gamma \blacktriangleright^m t}{\gamma \blacktriangleright^m \mathsf{suc}\, t} \qquad \frac{}{m\gamma \blacktriangleright^m \star_{\&}} \qquad \frac{}{\mathbf{0} \blacktriangleright^m \star_{\otimes}}$$

$$\frac{\gamma \blacktriangleright^{m\overline{p}} t \qquad \delta \blacktriangleright^m u \qquad \eta.\, 0 \blacktriangleright^0 A}{p\gamma + \delta \blacktriangleright^m \mathsf{unitrec}_p\, A\, t\, u}\ \mathsf{Unitrec}\, m\, p \qquad \frac{\gamma \blacktriangleright^{m\overline{p}} t \qquad \delta \blacktriangleright^0 A}{p\gamma \blacktriangleright^m \mathsf{emptyrec}_p\, A\, t}\ \mathsf{Emptyrec}\, m\, p$$

$$\frac{\gamma \blacktriangleright^m t}{\delta \blacktriangleright^m t}\ \delta \leq \gamma$$

Figure 4: The usage relation, $\gamma \blacktriangleright^m t$.

constant terms are assigned context $\mathbf{0}$ which maps grade 0 to every index. The scaling is notable only for the first component of pairs as all other constructors expect their contents to be used once, meaning that scaling is by 1. Strong pairs differ somewhat from this general principle. Since these only allow projections, only one of the components will be available from a single copy of the pair, while the other is discarded. To account for this we take the meet of their contributions instead of their sum, thus ensuring a compatible usage count regardless of which projection is taken. A similar exception occurs for $\star_{\&}$, the element of the strong unit type, for which there are no restrictions on the usage count in mode 1, allowing it to consume any amount of resources. Under mode 0, the usage context is restricted to $\mathbf{0}$ in order to preserve the invariant that variables are considered unused in mode 0.

Eliminators also generally follow a common pattern. Since eliminators possibly use several copies of one of their sub-terms (as indicated by an annotation), the corresponding usage count is scaled be the same factor. For instance, an application $t^p u$ uses the function argument p times, meaning that p copies of u are needed. This is reflected by the usage rule saying that one copy of u requires δ resources but one copy of $t^p u$ requires $\gamma + p\delta$ resources, corresponding to one copy of t and p copies of u. We also see that u is checked in mode mp, rather than m, ensuring that it is checked in the erased mode when $p = 0$. The eliminators corresponding to pattern matching, $\mathsf{prodrec}$, $\mathsf{unitrec}$ and $\mathsf{emptyrec}$ all behave in the same way, requiring, for instance, r copies of a weak pair t since the components are used r times in u. For the projections, the usage count of $\mathsf{fst}_p\, t$ or $\mathsf{snd}_p\, t$ is the same as that for t which, as mentioned, is a context compatible with either component of t. For the first projection the side condition restricts p to be bounded by 1 when checked in mode 1. This is because only one of the p copies of the first component

remains after taking the projection. The side condition ensures that this discarding is allowed by requiring compatibility between p and 1.

We are left with one usage rule, subsumption, which justifies the idea that the modality's partial order represents compatibility. It allows us to consider any term t using γ resources as instead using δ resources whenever $\delta \leq \gamma$. In particular, a single variable occurrence can be assigned grade ω (in any of our example modalities) and for a constant term one can similarly consider any variable to be used ω times. Under the affine types modality a variable may be assigned grade 1 even if it is not used—giving 1 the interpretation of a variable that is used either 0 or 1 times—while such an assignment is not allowed when the linearity modality is used.

Finally we note that some of the eliminators have side conditions. These involve custom predicates that make it possible to disallow certain invocations of eliminators, for instance those that use *erased matches* [Abel et al., 2023b].

Since a usage context corresponds to the number of variable uses we can define weakening for these contexts by inserting grade 0 at the correct positions to account for the non-occurring variable indices they introduce. We write $\gamma[\rho]$ for this operation and define it as follows:

$$\gamma[\mathsf{id}] = \gamma \qquad \gamma[\uparrow\rho] = \gamma[\rho].\,0 \qquad (\gamma.\,p)[\Uparrow\rho] = \gamma[\rho].\,p$$

As one would expect, usage is preserved by weakenings. That is, if $\gamma \blacktriangleright^m t$ then $\gamma[\rho] \blacktriangleright^m t[\rho]$ for any weakening ρ.

The reduction relation $\Gamma \vdash t \to u : A$ evaluates terms to weak head normal form (WHNF) in a call-by-name style. Notably, the reduction relation is typed, meaning that only well-typed terms will reduce. The reduction rules are otherwise reasonably standard and shown in Fig. 5. As usual, $\Gamma \vdash t \to^* u : A$ denotes the reflexive, transitive closure of reduction. As shown by Abel et al. [2017]—for a somewhat different theory—reduction is deterministic and all well-typed terms reduce to weak head normal form (but not further).

3 A Resource Aware Abstract Machine

Abel et al. [2023b] show that a usage preservation lemma holds for all modalities, and that the usage relation captures the concept of erasure correctly for modalities with a well-behaved zero. However, they explicitly leave "a serious soundness argument for linearity" for future work. The goal of this section is to provide such an argument.

When the linearity modality is used our intended interpretation of $\gamma \blacktriangleright^1 t$ is that, if $\gamma(i) = 1$, then x_i is used linearly (exactly once) in t. We specify what it means to be used through an abstract machine that keeps track of resource usage, and then we prove that the number of times a variable is used is consistent with the usage relation. Our proof holds for the linearity modality in particular but also for a larger class of modalities with a well-behaved zero which allow a form of subtraction which we define below.

3.1 Subtraction of Grades

While we will be using grades to represent how many times a resource may be used, they do not a priori provide a way to track how this number changes as resources are used. There are two questions that we would like to be able to answer. Firstly, "given p copies of some resource, are there q copies available?" and secondly, "if so, how many copies would be left if we used those q copies?". The second question seems to suggest some form of subtraction, but defining subtraction as a binary operator in the usual sense would be problematic since a modality with a well-behaved zero only has an additive inverse for zero.

$\boxed{\Gamma \vdash t \to u : A}$

$$\frac{\Gamma \vdash t \to u : A \qquad \Gamma \vdash A = B}{\Gamma \vdash t \to u : B} \qquad \frac{\Gamma \vdash t \to t' : \Pi^q_p\, A\, B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,{}^p u \to t'\,{}^p u : B[u]}$$

$$\frac{\Gamma.\, A \vdash B \qquad \Gamma.\, A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda^p t)\,{}^p u \to t[u] : B[u]} \qquad \frac{\Gamma.\, A \vdash B \qquad \Gamma \vdash t \to t' : \Sigma^q_{\&,p}\, A\, B}{\Gamma \vdash \mathsf{fst}_p\, t \to \mathsf{fst}_p\, t' : A}$$

$$\frac{\Gamma.\, A \vdash B \qquad \Gamma \vdash t \to t' : \Sigma^q_{\&,p}\, A\, B}{\Gamma \vdash \mathsf{snd}_p\, t \to \mathsf{snd}_p\, t' : B[\mathsf{fst}_p\, t]} \qquad \frac{\Gamma.\, A \vdash B \qquad \Gamma \vdash t : A \qquad \Gamma \vdash u : B[t]}{\Gamma \vdash \mathsf{fst}_p\, ({}^p t, u)_\& \to t : A}$$

$$\frac{\Gamma.\, A \vdash B \qquad \Gamma \vdash t : A \qquad \Gamma \vdash u : B[t]}{\Gamma \vdash \mathsf{snd}_p\, ({}^p t, u)_\& \to u : B[\mathsf{fst}_p\, ({}^p t, u)_\&]}$$

$$\frac{\Gamma.\, \Sigma^q_{\otimes,p}\, A\, B \vdash C \qquad \Gamma.\, A.\, B \vdash u : C[\uparrow^2 \mathsf{id}, ({}^p \mathsf{x}_1, \mathsf{x}_0)_\otimes] \qquad \Gamma \vdash t \to t' : \Sigma^q_{\otimes,p}\, A\, B}{\Gamma \vdash \mathsf{prodrec}^p_r\, A\, t\, u \to \mathsf{prodrec}^p_r\, C\, t'\, u : C[t]}$$

$$\frac{\Gamma.\, \Sigma^q_{\otimes,p}\, A\, B \vdash C \qquad \Gamma \vdash t_1 : A \qquad \Gamma \vdash t_2 : B[t] \qquad \Gamma.\, A.\, B \vdash u : C[\uparrow^2 \mathsf{id}, ({}^p \mathsf{x}_1, \mathsf{x}_0)_\otimes]}{\Gamma \vdash \mathsf{prodrec}^p_r\, C\, ({}^p t_1, t_2)_\otimes\, u \to u[t_1, t_2] : C[({}^p t_1, t_2)_\otimes]}$$

$$\frac{\Gamma.\, \top_\otimes \vdash A \qquad \Gamma \vdash u : A[\star_\otimes] \qquad \Gamma \vdash t \to t' : \top_\otimes}{\Gamma \vdash \mathsf{unitrec}_p\, A\, t\, u \to \mathsf{unitrec}_p\, A\, t'\, u : A[t]}$$

$$\frac{\Gamma.\, \top_\otimes \vdash A \qquad \Gamma \vdash u : A[\star_\otimes]}{\Gamma \vdash \mathsf{unitrec}_p\, A\, \star_\otimes\, u \to u : A[\star_\otimes]} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash t \to t' : \bot}{\Gamma \vdash \mathsf{emptyrec}_p\, A\, t \to \mathsf{emptyrec}_p\, A\, t' : A}$$

$\boxed{\Gamma \vdash t \to^* u : A}$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t \to^* t : A} \qquad \frac{\Gamma \vdash t \to u : A \qquad \Gamma \vdash u \to^* v : A}{\Gamma \vdash t \to^* v : A}$$

Figure 5: Weak head reduction of terms in one step, $\Gamma \vdash t \to u : A$, and many steps $\Gamma \vdash t \to^* u : A$.

It seems reasonable to say that we should be allowed to use q copies of p if there is some r such that $p = r + q$. This could be interpreted in the following way: "having p copies" is the same as "having q copies (to be used now) and an additional r copies (left for later use)". Taking subsumption into account we can refine this to $p \leq r + q$ with the interpretation changing from "is the same as" to "is compatible with". This alone is not enough to uniquely define subtraction, as there may be several r such that $p \leq r + q$. For instance, for the linearity modality we have $\omega \leq 1 + 1$ and $\omega \leq \omega + 1$. We can think of each such r as a number of copies that is safe or compatible to have left after q copies are used. In the linearity example, using 1 copy out of ω it is safe to consider this as leaving us with either ω copies or 1 copy. Of course, in the latter case we will be prevented from using more than 1 copy in the future. In order to avoid this we want to always pick the least candidate for r as this will leave us with as many copies to use as possible. With this in mind, we define subtraction in the following way:

Table 1: Subtraction for the five example modalities (subtraction is undefined for cases not mentioned). For the linearity and affine types modalities, their definitions are the same. The same is true for the two natural number modalities regardless of how the order relation is defined. Here, $m \ominus n$ and $n \leq m$ refer to the usual subtraction and order relation for natural numbers.

Erasure	Linear/Affine		Nat
$\omega - \omega = \omega$	$\omega - \omega = \omega$	$1 - 1 = 0$	$\omega - \omega = \omega$
$\omega - 0 = \omega$	$\omega - 1 = \omega$	$1 - 0 = 1$	$\omega - n = \omega$
$0 - 0 = 0$	$\omega - 0 = \omega$	$0 - 0 = 0$	$m - n = m \ominus n$, if $n \leq m$

Definition 3.1. The relation $p - q \leq r$ holds iff $p \leq r + q$. The relation $p - q = r$ holds iff r is the least grade such that $p - q \leq r$, i.e. iff $p - q \leq r$ and for every r' such that $p - q \leq r'$ we have $r \leq r'$.

Subtraction then satisfies the following properties:

Theorem 3.2. *For any modality, the following properties hold:*

- *Subtraction is functional: If $p - q = r$ and $p - q = r'$ then $r = r'$.*
- *Subtraction is monotone in its first argument: If $p \leq p'$ and $p - q = r$ and $p' - q = r'$ then $r \leq r'$.*
- *Subtraction is antitone in its second argument: If $q' \leq q$ and $p - q = r$ and $p - q' = r'$ then $r \leq r'$.*
- *Subtraction by 0 is always defined and is the identity: $p - 0 = p$.*
- *If p is the least element of the modality then $p - q = p$ for any q.*
- *If the modality has a well-behaved zero and $0 - p = q$ then $p = q = 0$.*

For our purposes the last two properties are perhaps the most notable. The first says that the least element of a modality represents an unlimited amount of resources. This is consistent with the interpretation of the partial order representing compatibility, as the least grade must be compatible with any (other) number of uses. Conversely, the last property implies that the grade 0 really does represent no available resources as attempting to use any (non-zero) number of resources is disallowed when 0 resources are available.

We say that a modality *supports subtraction* if $p - q \leq r'$ implies that there is some r such that $p - q = r$. We have not proved that this holds for all modalities. Below we only consider modalities that support subtraction. Note that all modalities mentioned above support subtraction, see Table 1.

3.2 The Abstract Machine

We will now define an abstract machine for the step-wise evaluation of terms. Structure-wise, it is similar to the lazy abstract machine by Sestoft [1997] but adapted to a call-by-name setting. We choose to use call-by-name to mirror the reduction mentioned above, simplifying the proof of bisimilarity, but we suspect that we could achieve similar results for call-by-value or call-by-need with relatively small changes. In regards to resource tracking we draw inspiration from Choudhury et al. [2021].

For the moment we will only consider evaluation of closed programs, but we generalize to open programs in Section 5.

We follow Sestoft and use machine states with four parts: a heap, an evaluation stack, the term that is being evaluated (the *head*) and an environment, a mapping from variables

$$
\begin{array}{rll}
e & ::= (t, \rho) & \text{Heap entries} \\
H & ::= \epsilon \mid H.\, e^p & \text{Heaps} \\
c & ::= \bullet^p u\ [\rho] \mid \mathsf{fst}_p\ \bullet \mid \mathsf{snd}_p\ \bullet \mid \mathsf{prodrec}^p_r\ A \bullet u\ [\rho] & \text{Continuations} \\
 & \quad \mid \mathsf{unitrec}_p\ A \bullet u\ [\rho] \mid \mathsf{emptyrec}_p\ A \bullet\ [\rho] \mid \mathsf{suc}\ \bullet & \\
S & ::= \epsilon \mid c.\, S & \text{Stacks} \\
s & ::= \langle H;\ t;\ \rho;\ S \rangle & \text{Machine states}
\end{array}
$$

Figure 6: The syntax of the abstract machine.

$\boxed{H \vdash \mathsf{y}_i \mapsto^q e\ ;\ H'}$

$$
\frac{p - q = r}{H.\, (t, \rho)^p \vdash \mathsf{y}_0 \mapsto^q (t, \uparrow\rho)\ ;\ H.\, (t, \rho)^r}
\qquad
\frac{H \vdash \mathsf{y}_i \mapsto^q (t, \rho)\ ;\ H'}{H.\, e^p \vdash \mathsf{y}_{i+1} \mapsto^q (t, \uparrow\rho)\ ;\ H'.\, e^p}
$$

$\boxed{H \vdash \mathsf{y}_i \mapsto e}$

$$
\frac{}{H.\, (t, \rho)^p \vdash \mathsf{y}_0 \mapsto (t, \uparrow\rho)}
\qquad
\frac{H \vdash \mathsf{y}_i \mapsto (t, \rho)}{H.\, e \vdash \mathsf{y}_{i+1} \mapsto (t, \uparrow\rho)}
$$

Figure 7: Heap lookup relations $H \vdash \mathsf{y}_i \mapsto^q e\ ;\ H'$ and $H \vdash \mathsf{y}_i \mapsto e$ for lookup of q copies with heap update and lookup without heap update respectively.

to pointers. The syntax for machine states, heaps and stacks is given in Fig. 6. We use de Bruijn indices as pointers (writing y_i for the pointer with index i). Like the syntax, the abstract machine is intrinsically well-scoped. For readability we do not include side conditions or assumptions related to this in our presentation. Note however that this ensures that pointers in entries can only refer to preceding entries. This allows us to use weakenings to represent environments. Heaps are then essentially lists of entries (which consist of a term and a weakening) together with a grade indicating how many copies of the entry are available.

The statement $H \vdash \mathsf{y}_i \mapsto^q e\ ;\ H'$, defined in Fig. 7, means that it is possible to dereference the pointer y_i into the heap H, obtaining the entry e and a new heap H'. In the new heap, the usage count of the entry has been decreased by q. This kind of lookup can fail if the heap does not contain sufficiently many copies of that entry, i.e. if it is not possible to subtract q from the number of available copies. In the returned heap H' the usage count of the looked up entry has been updated. We also define a separate lookup relation $H \vdash \mathsf{y}_i \mapsto e$ that always succeeds and does not involve a heap update, see Fig. 7.

The evaluation stack is a list of *continuations* that represent the context "around" the term that is currently being evaluated, see Fig. 6. Mostly these are eliminators with the scrutinee removed, sometimes along with a weakening for translation between variables and pointers. For instance, when an application is evaluated the function argument is put on the stack while the function is evaluated. While most continuations correspond to eliminator terms, there is one exception: the continuation $\mathsf{suc}\ \bullet$, which corresponds to the successor constructor. We use this to fully evaluate programs to numerals.

We note that a machine state can be seen as a decomposition of a (closed) term that can be recovered from the machine state. From this perspective heaps can be seen as substitutions where each variable is replaced by its corresponding entry in the

$$
\begin{aligned}
(\!|\epsilon|\!)\, t &= t & (\!|\mathsf{prodrec}^p_r\ A \bullet u\ [\rho]|\!)\, t &= \mathsf{prodrec}^p_r\ A[\Uparrow\rho]\ t\ u[\Uparrow^2\rho] \\
(\!|e.\, S|\!)\, t &= (\!|S|\!)\, \big((\!|e|\!)\, t\big) & (\!|\mathsf{unitrec}_p\ A \bullet u\ [\rho]|\!)\, t &= \mathsf{unitrec}_p\ A[\Uparrow\rho]\ t\ u[\rho] \\
(\!|\bullet^p u\ [\rho]|\!)\, t &= t^p(u[\rho]) & (\!|\mathsf{emptyrec}_p\ A \bullet\ [\rho]|\!)\, t &= \mathsf{emptyrec}_p\ A[\rho]\ t \\
(\!|\mathsf{fst}_p \bullet|\!)\, t &= \mathsf{fst}_p\ t & (\!|\mathsf{suc} \bullet|\!)\, t &= \mathsf{suc}\ t \\
(\!|\mathsf{snd}_p \bullet|\!)\, t &= \mathsf{snd}_p\ t
\end{aligned}
$$

Figure 8: Applying a stack or continuation to a term.

heap. Concretely, we define such substitutions recursively by $(\!|\epsilon|\!) = \mathsf{id}$ and $(\!|H.\,(t,\rho)^p|\!) = (\!|H|\!), t[(\!|H|\!) \circ \rho]$. We sometimes leave out the brackets when it is clear from the context that a heap is used as a substitution: we write $t[H]$ rather than $t[(\!|H|\!)]$. Stacks make up the "spine" of a term, missing only the scrutinee to form a complete term. We will write $(\!|S|\!)\, t$ for the term we get by plugging in t as the missing scrutinee as shown in Fig. 8. Given a state $s = \langle H;\, t;\, \rho;\, S\rangle$ the corresponding term can then be recovered as $(\!|s|\!) = \big((\!|S|\!)\, (t[\rho])\big)[H]$ (note the use of ρ to translate the variable indices of t to pointer indices).

Going in the other direction there can of course be several different states all corresponding to a term t. The most natural is perhaps $\langle \epsilon;\, t;\, \mathsf{id};\, \epsilon\rangle$ which we call the *initial* state of t and denote by $\langle t\rangle$. It is worth noting that the process of turning a heap into a substitution removes the resource tracking information. This makes applying substitutions or recovering a term potentially unsafe operations since they risk using resources more times than allowed. As such, care should be taken in how these operations are used. Notably, we will not make use of these when defining reduction for the machine.

Before we define the semantics of the machine, we note that since eliminators do not necessarily use their scrutinee only once, the semantics should provide a way of evaluating several copies at once. In particular, when evaluating $\mathsf{prodrec}^p_\omega\ A\, \mathsf{x}_0\ u$, if the evaluation does not get stuck then we will eventually need to look up (the pointer corresponding to) x_0 in the heap, giving a pair whose components are used an unrestricted number of times in u. Since these are used an unrestricted number of times we should not allow lookups of entries which may not be used unrestrictedly. For that reason we keep track of the number of copies of the head that the current state is using. For similar reasons Choudhury et al. [2021] parametrize their reduction relation by a grade r, indicating the number of copies that are currently being evaluated. We will use a slightly different approach by instead noting that this information can be retrieved from the stack.

We write $|S| \equiv p$ (see Fig. 9) to denote that a given stack indicates that p copies should be evaluated and similarly for continuations. We will refer to the grade p as the *multiplicity* of the stack or continuation. For the continuations, the multiplicity is determined by examination of their usage rule, similarly to what we did for $\mathsf{prodrec}$ above. For a stack, all its continuations are taken into account by multiplying their multiplicities. Note that the multiplicity relation is functional. We will therefore use $|S|$ to denote the multiplicity of S with the implicit assumption that it exists. In fact, with the current definition, the stack multiplicity always exists but this will no longer be the case once the eliminator for natural numbers is added.

We will define the semantics of the machine in a way that mirrors the call-by-name weak head reduction of the language. This means that we will define it as a small step semantics in a call-by-name style. This allows us to translate already known properties of the call-by-name reduction to the semantics of the machine via a bisimilarity argument.

$\boxed{|S| \equiv p}$

$$\frac{}{|\epsilon| \equiv 1} \qquad \frac{|S| \equiv p \qquad |e| \equiv q}{|e.\, S| \equiv pq}$$

$\boxed{|e| \equiv p}$

$$\frac{}{|\bullet^p u\ [\rho]| \equiv 1} \qquad \frac{}{|\mathsf{fst}_p\ \bullet| \equiv 1} \qquad \frac{}{|\mathsf{snd}_p\ \bullet| \equiv 1} \qquad \frac{}{|\mathsf{prodrec}^p_r\ A \bullet u\ [\rho]| \equiv r}$$

$$\frac{}{|\mathsf{unitrec}_p\ A \bullet u\ [\rho]| \equiv p} \qquad \frac{}{|\mathsf{emptyrec}_p\ A \bullet [\rho]| \equiv p} \qquad \frac{}{|\mathsf{suc}\ \bullet| \equiv 1}$$

Figure 9: Multiplicity of stacks and continuations.

For a given machine state, we can think of one evaluation step as doing one of three things:

1. looking up the value of a variable in the heap,
2. putting a continuation on the stack, or
3. popping a continuation off the stack.

It turns out to be convenient to know not only that one state reduces to another but also in what way. In particular, we separate the third from the others and define the relation $s \to_e s'$ capturing reductions in the first two groups and $s \to_v s'$ capturing those of the third, see Fig. 10. We denote the union of these by $s \to s'$ and will refer to it as the weak head semantics of the machine. As usual, notation like $s \to^* s'$ is used for reflexive, transitive closures.

The first relation concerns cases where the head of the state is either an eliminator or a variable. For eliminators, a corresponding continuation is put on top of the stack and evaluation continues with the scrutinee in head position. In the variable case, we look up $|S|$ copies of the pointer corresponding to the variable and continue evaluating the result under the updated heap. As mentioned above, $|S|$ corresponds to the number of copies that is currently being evaluated and thus how many copies that should be looked up. Importantly, the variable rule can fail if an attempt is made to look up more copies than are available (as defined by the subtraction relation of the modality) in which case evaluation would halt.

The second relation concerns cases where the head of the state is a value. These rules can fail if the top of the stack does not match the head term, for instance in the case of a projection with a lambda in head position. In the case of a match the continuation on top of the stack is popped off, and new entries are possibly put on the heap. The number of copies put on the heap are given by the grade annotations of the continuation—in analogy with the usage relation—scaled by the stack multiplicity.

We scale because we do not necessarily evaluate 1 copy. For instance, consider the case where the linearity modality is used and $|S| = \omega$. This indicates that some eliminator on the stack uses its argument ω times. If a linear function application is evaluated under the stack S, and only one (1) copy of the argument is put on the heap, then this argument cannot be used (except for in erased settings, see below): heap lookup requires that we can subtract $|S'|$ from the heap entry's grade, where S' is the current stack.

When new entries are put on the heap, the pointer indices of all previous entries are shifted. The environment of the state is thus updated to account for this. Similarly, all environments in the current stack are updated. We write this as $S[\rho]$ which is defined as replacing every environment ρ' occurring in S with $\rho \circ \rho'$.

$\boxed{s \to_e s'}$

$$
\begin{aligned}
\langle H;\, \mathsf{x}_i;\, \rho;\, S\rangle &\to_e \langle H';\, t;\, \rho';\, S\rangle && \text{if } H \vdash \mathsf{y}_{\mathsf{x}_i[\rho]} \mapsto^{|S|} (t, \rho')\,;\, H' \\
\langle H;\, t^p u;\, \rho;\, S\rangle &\to_e \langle H;\, t;\, \rho;\, \bullet^p u\,[\rho].\, S\rangle \\
\langle H;\, \mathsf{fst}_p\, t;\, \rho;\, S\rangle &\to_e \langle H;\, t;\, \rho;\, \mathsf{fst}_p\, \bullet.\, S\rangle \\
\langle H;\, \mathsf{snd}_p\, t;\, \rho;\, S\rangle &\to_e \langle H;\, t;\, \rho;\, \mathsf{snd}_p\, \bullet.\, S\rangle \\
\langle H;\, \mathsf{prodrec}^p_r\, A\, t\, u;\, \rho;\, S\rangle &\to_e \langle H;\, t;\, \rho;\, \mathsf{prodrec}^p_r\, A \bullet u\,[\rho].\, S\rangle \\
\langle H;\, \mathsf{unitrec}_p\, A\, t\, u;\, \rho;\, S\rangle &\to_e \langle H;\, t;\, \rho;\, \mathsf{unitrec}_p\, A \bullet u\,[\rho].\, S\rangle \\
\langle H;\, \mathsf{emptyrec}_p\, A\, t;\, \rho;\, S\rangle &\to_e \langle H;\, t;\, \rho;\, \mathsf{emptyrec}_p\, A \bullet\,[\rho].\, S\rangle
\end{aligned}
$$

$\boxed{s \to_v s'}$

$$
\begin{aligned}
\langle H;\, \lambda^p t;\, \rho;\, \bullet^p u\,[\rho'].\, S\rangle &\to_v \langle H.\,(u, \rho')^{|S|p};\, t;\, \Uparrow\rho;\, S[\uparrow\mathsf{id}]\rangle \\
\langle H;\, ({}^p t_1, t_2)_{\&};\, \rho;\, \mathsf{fst}_p\, \bullet.\, S\rangle &\to_v \langle H;\, t_1;\, \rho;\, S\rangle \\
\langle H;\, ({}^p t_1, t_2)_{\&};\, \rho;\, \mathsf{snd}_p\, \bullet.\, S\rangle &\to_v \langle H;\, t_2;\, \rho;\, S\rangle \\
\langle H;\, ({}^p t_1, t_2)_{\otimes};\, \rho;\, \mathsf{prodrec}^p_r\, A \bullet u\,[\rho'].\, S\rangle &\to_v \\
&\quad \langle H.\,(t_1, \rho)^{|S|rp}.\,(t_2, \Uparrow\rho)^{|S|r};\, u;\, \Uparrow^2\rho';\, S[\uparrow^2\mathsf{id}]\rangle \\
\langle H;\, \star_{\otimes};\, \rho;\, \mathsf{unitrec}_p\, A \bullet u\,[\rho'].\, S\rangle &\to_v \langle H;\, u;\, \rho';\, S\rangle
\end{aligned}
$$

$\boxed{s \to_n s'}$

$$
\begin{aligned}
\langle H;\, \mathsf{suc}\, t;\, \rho;\, \mathsf{suc}^k\, \bullet\rangle &\to_n \langle H;\, t;\, \rho;\, \mathsf{suc}^{k+1}\, \bullet\rangle && \text{if } \neg\, \mathsf{Numeral}\, t \\
\langle H;\, t;\, \rho;\, \mathsf{suc}\, \bullet.\, S\rangle &\to_n \langle H;\, \mathsf{suc}\, t;\, \rho;\, S\rangle && \text{if } \mathsf{Numeral}\, t
\end{aligned}
$$

Figure 10: The machine reduction relations for eliminators, $s \to_e s'$, values, $s \to_v s'$, and numerals $s \to_n s'$. The stack $\mathsf{suc}^k\, \bullet$ consists (only) of k copies of $\mathsf{suc}\, \bullet$.

The observant reader might have noticed that it is possible to look up zero copies of a heap entry whenever the stack multiplicity is 0. As can be gleaned from its definition, the multiplicity of a stack is zero if (and only if) it contains $\mathsf{prodrec}_0$, $\mathsf{unitrec}_0$, or $\mathsf{emptyrec}_0$, i.e. in the case of an *erased match* [Abel et al., 2023b] on a weak pair, for the weak unit type, or for the empty type. In either case, this indicates that the current head is an erased term which does not have any computational relevance. While lookups may still happen there is no risk of any data on the heap being used more times than allowed as any heap insertions will put in 0 copies. Nevertheless, depending on what the theory is used for one might not want to allow such lookups in erased contexts. Erased matches can be disallowed using the corresponding side conditions in the usage relation. The results we present here hold regardless of whether erased matches are allowed or not but we will return to this matter in Section 5 when we extend the machine to handle evaluation of open terms.

We will also make use of a variant of the semantics for which we do not keep track of resource usage: $s \dashrightarrow s'$. Since the variable case is the only one in which this is done we can define this simply by replacing the variable rule of the weak head semantics with the

following one:

$$\frac{H \vdash \mathsf{y}_{i[\rho]} \mapsto (t, \rho')}{\langle H;\, \mathsf{x}_i;\, \rho;\, S\rangle \dashrightarrow_e \langle H;\, t;\, \rho';\, S\rangle}$$

As Theorems 3.9 and 3.13 will show, the weak head semantics mirrors the call-by-name weak-head reduction of the language. In order to prove our correctness theorem, Theorem 3.17 this is not quite sufficient. The reason is that evaluation to WHNF could leave us with states for which the head still contains references to the heap, representing resources that have not yet been used. Since our goal involves verifying that the number of times variables are used is correct we want evaluation to continue until no such references remain. We will therefore evaluate programs (terms of type $\mathbb{N}$) to numerals.

For this purpose, we define an additional reduction relation $s \to_n s'$, see Fig. 10. This allows $\mathsf{suc}t$ to be evaluated further if t is not a numeral. Note that we only allow evaluation under suc when the stack is $\mathsf{suc}^k \bullet$, the stack consisting only of k successor eliminators. We add this stipulation to ensure that the reduction semantics stays deterministic once we add the continuation for the natural number eliminator. Using this, we define an extended reduction relation $s \twoheadrightarrow s'$ to be the union of $s \to_e s'$, $s \to_v s'$ and $s \to_n s'$. We will refer to this as the full semantics of the machine.

Our goal is to show that this semantics is correct in the sense that all programs (of type $\mathbb{N}$) evaluate to a numeral and use resources correctly: evaluation should not get stuck because the heap did not contain enough resources, and once evaluation finishes, the heap should have no leftover resources that were expected to have been used. Note that the latter condition does not mean that the heap should not have any resources remaining (i.e. that all entries are annotated with 0): for the linearity modality it should be fine to have entries annotated with ω, because ω stands for *any* number of uses, including none. It suffices if all heap entries are annotated with grades that are compatible with 0, i.e. *at most* 0. Note that for the linearity modality $0 \leq 0$ and $\omega \leq 0$, but $1 \not\leq 0$.

As we will see, evaluation with the full reduction strategy can be summarized in the following way: First evaluation uses the weak head semantics until a state with $\mathsf{suc}\, t$ in head position is reached (we may also reach zero right away in which case evaluation halts). At this point, a successor continuation is put on the stack and evaluation continues using the weak head semantics. This repeats until a state with zero in head position is reached, at which point evaluation finishes by popping all successor continuations off the stack (one by one). In other words, most interesting steps of evaluation are covered by the reduction steps of the weak head semantics. Although our main goal involves full reduction, we will therefore spend most of our effort considering only weak head reduction.

In particular, we will show that in each iteration of the procedure described above weak head reduction will lead to a state with either zero or $\mathsf{suc}\, t$ as the head, and use this to show that full reduction matches the procedure described above. Looking at the reduction rules, we can see three ways in which weak head evaluation can get stuck or terminate (two of which were mentioned above):

1. variable lookup fails because there are not enough resources on the heap,
2. there is a value in head position with a non-matching continuation on top of the stack, or
3. there is a value in head position and the stack is empty.

In the next few sections we will consider these reasons one by one, leading up to our main correctness proof. The first reason is perhaps the most interesting as it means that a term is trying to access more resources than allowed—for instance attempting to access a linear resource more than once. We will show in Section 3.3 that well-resourced states cannot get stuck in this way. The second reason is an indication of a malformed state, such as applying a projection to a lambda abstraction. As we will show in Section 3.4,

$\boxed{\gamma \blacktriangleright H}$

$$\frac{}{\epsilon \blacktriangleright \epsilon} \qquad \frac{\gamma + p\delta[\rho] \blacktriangleright H \qquad \delta \blacktriangleright^{\overline{p}} t}{\gamma.\, p \blacktriangleright H.\, (t, \rho)^p} \qquad \frac{\gamma \blacktriangleright H}{\delta \blacktriangleright H}\ \gamma \leq \delta$$

$\boxed{\gamma \blacktriangleright^m c}$

$$\frac{\gamma \blacktriangleright^{m\overline{p}} u}{p\gamma[\rho] \blacktriangleright^m \bullet^p u\ [\rho]} \qquad \frac{}{\mathbf{0} \blacktriangleright^m \mathsf{fst}_p \bullet}\ m = 1 \to p \leq 1 \qquad \frac{}{\mathbf{0} \blacktriangleright^m \mathsf{snd}_p \bullet}$$

$$\frac{\gamma.\, mrp.\, mr \blacktriangleright^m u}{\gamma[\rho] \blacktriangleright^m \mathsf{prodrec}^p_r\ A \bullet u\ [\rho]}\ \mathsf{Prodrec}\, m\, r\, p \qquad \frac{\gamma \blacktriangleright^m u}{\gamma[\rho] \blacktriangleright^m \mathsf{unitrec}_p\ A \bullet u\ [\rho]}\ \mathsf{Unitrec}\, m\, p$$

$$\frac{}{\mathbf{0} \blacktriangleright^m \mathsf{emptyrec}_p\ A \bullet [\rho]}\ \mathsf{Emptyrec}\, m\, p$$

$\boxed{\gamma \blacktriangleright S}$

$$\frac{}{\mathbf{0} \blacktriangleright \epsilon} \qquad \frac{\delta \blacktriangleright^{\overline{|S|}} c \qquad \gamma \blacktriangleright S}{\gamma + |S|\delta \blacktriangleright c.\, S}$$

$\boxed{\blacktriangleright s}$

$$\frac{\gamma \blacktriangleright H \qquad \delta \blacktriangleright^{\overline{|S|}} t \qquad \eta \blacktriangleright S}{\blacktriangleright \langle H;\, t;\, \rho;\, S \rangle}\ \gamma \leq |S| \cdot \delta[\rho] + \eta$$

Figure 11: Usage relations for heaps, $\gamma \blacktriangleright H$, continuations, $\gamma \blacktriangleright^m c$, stacks, $\gamma \blacktriangleright S$, and states, $\blacktriangleright s$.

such situations are ruled out by typing. Conversely, the final reason is the expected way for evaluation to terminate and in Section 3.5 we show that all "sufficiently well-behaved" programs indeed terminate in this way and in the process use any resources as many times as specified.

3.3 Usage for the Machine

In order to rule out the possibility of evaluation getting stuck because the heap does not contain enough resources, we will define a usage relation for machine states. The idea is to keep track of both the resources available in the heap and the resources that will be consumed by the head and stack, and make sure that these are compatible, i.e. that the heap contains at least as many resources as will be consumed. We partly follow Choudhury et al. [2021].

The first step is to define a usage relation for heaps, representing the available resources, see Fig. 11. We write this as $\gamma \blacktriangleright H$ with the interpretation that γ represents the number of available copies of each entry. When a heap entry $(t, \rho)^p$, with $\delta \blacktriangleright^{\overline{p}} t$ is added to H, the new entry is assigned grade p in the context, representing that its has p copies available. For all preceding entries, however, the number of available copies seem to decrease (from $\gamma + p\delta[\rho]$ to γ). This is done to account for any resources that the p copies of t may consume, making them unavailable to be used elsewhere. As for terms, we allow a form of subsumption for heaps. This one goes the other way around, because while the usage relation for terms represents resources that will be consumed, usage for heaps represents resources that are available.

The head and stack make up the "consuming" parts of a state. The usage relation from Fig. 4 describes the resources consumed by the head. The same usage relation also forms the basis for finding the usage counts of the stack. In short, the resources consumed by a continuation are the same as the ones for the corresponding term with the resources for the scrutinee removed and the assumptions regarding eliminators' motives dropped. This is expressed by $\gamma \blacktriangleright^m c$, defined in Fig. 11. Since we are currently only considering the abstract machine from the perspective of the weak head semantics, the continuation $\mathsf{suc}\,\bullet$ will never be placed on the stack. Consequently, it has no usage rule. For stacks, the usage counts of all continuations are simply added together, scaled by the stack multiplicity to account for the number of copies being evaluated, see Fig. 11. We write this as $\gamma \blacktriangleright S$.

With this, we can express both how many resources a state has available for use and how many it will need to be evaluated. For a well-resourced state, these should be in balance, i.e. there should always be enough resources available. We define usage of states, $\blacktriangleright s$ in Fig. 11. The first three conditions ensure that the heap, head and stack are all well-resourced with respect to certain usage contexts. The side condition is key for this definition. The right hand side, $|S| \cdot \delta[\rho] + \eta$ represents the total number of resources that will be consumed by the ($|S|$ copies of) the head and the stack, so this condition ensures that the heap contains at least as many resources as will be consumed. The idea is that this invariant should be preserved as the state gets evaluated, and we will now show that this is indeed the case. First we present a lemma for a corresponding property for heap lookups:

Lemma 3.3. *If $H \vdash \mathsf{y}_i \mapsto^q (t, \rho)\ ;\ H'$ and $\gamma \blacktriangleright H$ and $\gamma(i) - q \leq r$ then there is a context δ such that $\delta \blacktriangleright^{\overline{q}} t$ and $\gamma' + q\delta[\rho] \blacktriangleright H'$, where γ' is the context that assigns grade r to i and is otherwise equal to γ.*

This lemma implies that when an entry is looked up in a well-resourced heap, then the obtained term and the updated heap are both well-resourced. Earlier we remarked that as we put things on the heap, resources become unavailable as they get "reserved" by the new entry. Here we see these resources become free again as entries are taken out of the heap. We also see that for the updated heap, the number of available copies of the looked-up entry is now r (due to well-scopedness t cannot refer to y_i, so $\delta[\rho]$ associates grade 0 to i).

With this taken care of, we can now show that the property of being well-resourced is preserved by reduction.

Theorem 3.4. *If $\blacktriangleright s$ and $s \to s'$ or $s \to^* s'$ then $\blacktriangleright s'$.*

Note that this does not say that the number of resources that are available or will be consumed is preserved but rather that after each reduction step, there are still sufficiently many resources available.

To conclude this section, we shall prove what we set out to show: reduction cannot get stuck due to a failing heap lookup. In particular, heap lookup does not fail for well-resourced states:

Theorem 3.5. *If $\blacktriangleright \langle H; \mathsf{x}_i; \rho; S\rangle$ then there is an entry e and heap H' such that $H \vdash \mathsf{y}_{i[\rho]} \mapsto^{|S|} e\ ;\ H'$.*

Taken together with Theorem 3.4, we see that starting with a well-resourced state, reduction will never fail because the variable rule could not be applied due to insufficient resources.

$\boxed{\vdash H : \Gamma}$

$$\frac{}{\vdash \epsilon : \epsilon} \qquad \frac{\vdash H : \Gamma \qquad \epsilon \vdash t[H \circ \rho] : A[H]}{\vdash H.\,(t, \rho)^p : \Gamma.\,A}$$

$\boxed{H \vdash c : (t : A) \rightsquigarrow B}$

$$\frac{\epsilon.\,A \vdash B \qquad \epsilon \vdash u[H \circ \rho] : A}{H \vdash \bullet^p u\ [\rho] : (t : \Pi^q_p\ A\ B) \rightsquigarrow B[u[H \circ \rho]]} \qquad \frac{\epsilon.\,A \vdash B}{H \vdash \mathsf{fst}_p\ \bullet : (t : \Sigma^q_{\&,p}\ A\ B) \rightsquigarrow A}$$

$$\frac{\epsilon.\,A \vdash B}{H \vdash \mathsf{snd}_p\ \bullet : (t : \Sigma^q_{\&,p}\ A\ B) \rightsquigarrow B[(\mathsf{fst}_p\ t)[H]]}$$

$$\frac{\epsilon.\,A.\,B \vdash u[\Uparrow^2 H \circ \Uparrow^2 \rho] : C[\uparrow^2 \mathsf{id}, ({}^p\mathsf{x}_1, \mathsf{x}_0)_\otimes \circ \Uparrow H \circ \Uparrow \rho] \qquad \epsilon.\,\Sigma^{q'}_{\otimes,p}\ A\ B \vdash C[\Uparrow H \circ \Uparrow \rho]}{H \vdash \mathsf{prodrec}^p_r\ C \bullet u\ [\rho] : (t : \Sigma^q_{\otimes,p}\ A\ B) \rightsquigarrow C[t[H] \circ \Uparrow H \circ \Uparrow \rho]}$$

$$\frac{\epsilon \vdash u[H \circ \rho] : A[\star_\otimes \circ \Uparrow H \circ \Uparrow \rho] \qquad \epsilon.\,\top_\otimes \vdash A[\Uparrow H \circ \Uparrow \rho]}{H \vdash \mathsf{unitrec}_p\ A \bullet u\ [\rho] : (t : \top_\otimes) \rightsquigarrow A[t[H] \circ \Uparrow H \circ \Uparrow \rho]}$$

$$\frac{\epsilon \vdash A[H \circ \rho]}{H \vdash \mathsf{emptyrec}_p\ A \bullet\ [\rho] : (t : \bot) \rightsquigarrow A[H \circ \rho]}$$

$\boxed{H \vdash S : (t : A) \rightsquigarrow B}$

$$\frac{}{H \vdash \epsilon : (t : A) \rightsquigarrow A} \qquad \frac{H \vdash c : (t : A) \rightsquigarrow B \qquad H \vdash S : (\langle\!|c|\!\rangle\ t : B) \rightsquigarrow C}{H \vdash c.\,S : (t : A) \rightsquigarrow C}$$

$\boxed{\vdash s : A}$

$$\frac{\vdash H : \Gamma \qquad \epsilon \vdash t[H \circ \rho] : B \qquad H \vdash S : (t[\rho] : B) \rightsquigarrow A}{\vdash \langle H;\ t;\ \rho;\ S\rangle : A}$$

Figure 12: Typing for heaps, $\vdash H : \Gamma$, continuations, $H \vdash c : (t : A) \rightsquigarrow B$, stacks, $H \vdash S : (t : A) \rightsquigarrow B$, and states, $\vdash s : A$.

3.4 Typing for the Machine

We will now move on to show that we can also rule out the possibility of evaluation stopping because of a value in head position with a non-matching continuation on top of the stack. As already hinted at, we will do so by first defining a typing relation for machine states, in much the same way as we defined the usage relation.

We write $\vdash H : \Gamma$ to mean that heap H is well-typed. The "type" of a heap is a context Γ containing the types of all entries in the heap, see Fig. 12. Each entry is checked in an empty context—recall that while the contents of an entry may be an open term this is merely part of a representation of a closed term.

Stacks and continuations are in some sense incomplete terms, waiting for a scrutinee to be applied. We define typing for these in much the same way as for usage, that is in the same way as the corresponding term but without requiring that the scrutinee is

well-typed. Unlike usage however, the scrutinee is still considered, because the type of a continuation (or stack) could depend on it: $H \vdash c : (t : A) \rightsquigarrow B$ means that c is a well-typed continuation that, under the heap H, takes a scrutinee t of type A to a term of type B (which may depend on t). Like entries, sub-terms of the continuations are checked in the empty context. We show the definition in Fig. 12, along with typing for stacks, $H \vdash S : (t : A) \rightsquigarrow B$. As with usage, the successor continuation is not included since we are currently only concerned with the machine from the perspective of weak head reduction.

Typing of machine states, $\vdash s : A$, makes use of the typing relations for continuations and stacks, see Fig. 12. The type of a state corresponds to the type of the term that is being evaluated. In particular, if $\vdash s : A$ then $\epsilon \vdash (\!|s|\!) : A$. Like usage, typing is preserved by reduction.

THEOREM 3.6. *If* $\vdash s : A$ *and* $s \to s'$ *or* $s \to^* s'$ *then* $\vdash s' : A$

For a well-typed state with a non-empty stack, the type of the head (with a certain substitution applied to it) and the "input type" of the top-most continuation must match. This allows us to show that well-typed states cannot get stuck because of a non-matching continuation. Concretely, well-typed states with values in head position and non-empty stacks always reduce:

THEOREM 3.7. *If* $\vdash \langle H; t; \rho; c.\, S\rangle : A$ *where* t *is a value then there is some state* s *such that* $\langle H; t; \rho; c.\, S\rangle \to_v s$.

Taking Theorems 3.6 and 3.7 together we can see that for well-typed states, evaluation cannot get stuck due to a mismatch between the head and the stack. In fact, for states which additionally are well-resourced we have the following property.

THEOREM 3.8. *If* $\vdash s : A$ *and* $\blacktriangleright s$ *for some state* s *that does not evaluate further then the head of* s *is a value and its stack is empty.*

3.5 Bisimilarity and Termination

At this point we have shown that there is only one way in which (well-typed and well-resourced) programs can terminate, but it remains to show that all programs do terminate. We mentioned already an analogue of this for the main call-by-name reduction in Section 2. We will use this property by first showing that the machine and the call-by-name reduction relations are bisimilar (we will refer to the reduction from Section 2 as call-by-name reduction even though the semantics of the abstract machine is given in the same style). The first direction, showing that machine evaluation steps are matched by call-by-name reduction steps, is relatively straightforward:

THEOREM 3.9. *For any machine states* s *and* s'*:*

- *If* $s \to_e s'$ *then* $(\!|s|\!) = (\!|s'|\!)$.
- *If* $\vdash s : A$ *and* $s \to_v s'$ *then* $\epsilon \vdash (\!|s|\!) \to (\!|s'|\!) : A$.
- *If* $\vdash s : A$ *and* $s \to s'$ *then* $\epsilon \vdash (\!|s|\!) \to^* (\!|s'|\!) : A$.
- *If* $\vdash s : A$ *and* $s \to^* s'$ *then* $\epsilon \vdash (\!|s|\!) \to^* (\!|s'|\!) : A$.

Note that $\to_e$ steps correspond to zero call-by-name reduction steps, while $\to_v$ steps correspond to one step.

For the other direction we note that for machine states with values in head position there is a similar one step to one step correspondence between reduction for values and call-by-name reduction. States with values in head position are said to be in normal form. For a general (sufficiently well-behaved) machine state, one step in the call-by-name semantics corresponds to some number of steps using $\to_e$ to reach a state in normal form, followed by a single step using $\to_v$.

First we relate reduction without resource tracking to reduction with resource tracking. As noted above variable lookups cannot fail for reduction without resource tracking, and by Theorems 3.4 and 3.5 we know that this is the case also for the resource tracking semantics if states are well-resourced. Consequently, apart from the resource information in the heap, the two semantics behave in the same way for well-resourced states. We prove the following bisimilarity property, where $H \sim H'$ denotes pointwise equality of heaps ignoring the grade annotations on heap entries.

THEOREM 3.10.

- *If* $s \to \langle H; t; \rho; S\rangle$ *then there is a heap* H' *such that* $s \dashrightarrow \langle H'; t; \rho; S\rangle$ *and* $H \sim H'$.
- *If* $\langle H; t; \rho; S\rangle \dashrightarrow \langle H'; t'; \rho'; S'\rangle$ *and* $\gamma \blacktriangleright H''$ *for some heap* H'' *with* $H \sim H''$ *then there is a heap* H''' *such that* $\langle H''; t; \rho; S\rangle \to \langle H'''; t'; \rho'; S'\rangle$.

Let us now prove that every sufficiently well-behaved state can be evaluated to normal form using $\to_e$. For any state there are three possibilities:

1. The head is a value, in which case we are done.
2. The head is an eliminator, in which case we put the corresponding continuation on the stack and continue.
3. The head is a variable, in which case we look up the variable and continue.

This procedure terminates: in the second case the heap stays unchanged and the head becomes smaller, and in the last case we may evaluate the looked up entry in a smaller heap since it can only contain references to preceding entries. We get the following:

THEOREM 3.11. *For any machine state* s*:*

- *There is some state* s' *in normal form with* $s \dashrightarrow_e^* s'$.
- *If* $\blacktriangleright s$ *then there is some state* s' *in normal form such that* $s \to_e^* s'$.

As mentioned above we also have the following:

THEOREM 3.12. *For any machine state* s *in normal form with* $\vdash s : A$ *and* $\epsilon \vdash (\!|s|\!) \to u : B$ *there is some* s' *such that* $s \to_v s'$ *and* $(\!|s'|\!) = u$.

Putting the pieces together we get the following theorem:

THEOREM 3.13. *If* $\epsilon \vdash (\!|s|\!) \to u : A$ *and* $\vdash s : B$ *and* $\blacktriangleright s$*, then there is some state* s' *such that* $s \to^* s'$ *and* $(\!|s'|\!) = u$*. If additionally* u *is in WHNF then* s' *does not evaluate further: the head of* s' *is a value and the stack is empty.*

As a corollary we get that evaluation with the weak head semantics terminates:

COROLLARY 3.14. *If* $\vdash s : A$ *and* $\blacktriangleright s$ *then* $s \to^* s'$ *for some state* $s' = \langle H; t; \rho; \epsilon\rangle$ *where* t *is a value, and* s' *does not evaluate further.*

3.6 Resource Correctness

In the last few sections we have exclusively been looking at the weak head semantics of the machine but it is now finally time to turn our attention to the full evaluation to numerals. First we will show that every state of type $\mathbb{N}$ evaluates to a numeral. The basic idea is to apply Corollary 3.14 to reach a state with either zero or $\mathsf{suc}\, t$ in head position. In the latter case, a successor continuation is placed on the stack and Corollary 3.14 is applied again repeatedly until the head becomes zero, at which point the continuations are popped off the stack.

One problem is that our proof of Corollary 3.14 relies on there not being any successor continuations on the stack (this is ruled out by the state being well-resourced and well-typed). In particular, our proof uses Theorem 3.9 which does not hold if successor

continuations are allowed on the stack. We circumvent this problem by making sure that the stack is empty when Corollary 3.14 is applied and use the following lemma to add the continuations (the stack $S.\,\mathsf{suc}^k\,\bullet$ refers to the concatenation of stacks S and $\mathsf{suc}^k\,\bullet$):

LEMMA 3.15. *If* $\langle H;\, t;\, \rho;\, S\rangle \twoheadrightarrow^* \langle H';\, t';\, \rho';\, S'\rangle$ *then* $\langle H;\, t;\, \rho;\, S.\,\mathsf{suc}^k\,\bullet\rangle \twoheadrightarrow^* \langle H';\, t';\, \rho';\, S'.\,\mathsf{suc}^k\,\bullet\rangle$.

This allows us to add any number of successor continuations at the bottom of a stack while preserving evaluation. We can thus apply the procedure described above to show that all states evaluate to numerals:

THEOREM 3.16. *If* $\vdash s : \mathbb{N}$ *and* $\blacktriangleright s$ *then there is a numeral* n *and heap* H *such that* $s \twoheadrightarrow \langle H;\, n;\, \rho;\, \epsilon\rangle$ *and* $\blacktriangleright \langle H;\, n;\, \rho;\, \epsilon\rangle$.

Notably, this means that evaluation does not get stuck because some resource is attempted to be used more than allowed. In particular, a linear application places a single copy of the function argument on the heap (assuming that the stack multiplicity is 1), and we now know that evaluation will use it at most once.

Of course, there is still one part missing—namely that the argument is used *at least* once. This follows more or less directly from Theorem 3.16 by noting that since n is a numeral and the stack is empty, the state we end up in consumes no resources. Since the state is well-resourced this means that $\gamma \blacktriangleright H$ for some context $\gamma \leq \mathbf{0}$ from which it follows that the number of copies of each entry in H is bounded by 0.

THEOREM 3.17. *If* $\epsilon \blacktriangleright^1 t$ *and* $\epsilon \vdash t : \mathbb{N}$ *then there is a numeral* n *and heap* H *such that* $\langle t\rangle \twoheadrightarrow^* \langle H;\, n;\, \rho;\, \epsilon\rangle$ *and the grade associated with each entry in* H *is bounded by* 0.

For the linear types modality, the final heap cannot contain any remaining linear entries. Thus, any such entries that were placed on the heap must have been looked up exactly once (recall that one cannot get ω by subtracting from 1 or 0).

We end this section by applying the theorem to a concrete example. While weak pairs do not have projections as primitive operations, we can define the first projection (for simplicity restricted to pairs of natural numbers with grade 1 for the first component) using prodrec by $\mathtt{proj}_1\, t = \mathsf{prodrec}^1_1\, \mathbb{N}\, t\, \mathsf{x}_1$. Because this function discards the second component we would expect it to be disallowed when we use the linear types modality, and we will show this using Theorem 3.17. Specifically, we will show that one cannot construct a usage rule for $\mathtt{proj}_1$, that is, that usage rules like the following one are not admissible for any context δ:

$$\frac{\gamma \blacktriangleright^m t}{\delta \blacktriangleright^m \mathtt{proj}_1\, t}$$

Let us evaluate $\mathtt{proj}_1\ ({}^1\mathsf{zero}, \mathsf{suc}\,\mathsf{zero})_\otimes$ using the machine. During evaluation two entries are added to the heap, both containing zero and both assigned grade 1. The machine then looks up a single copy of the entry corresponding to variable x_1, resulting in the machine terminating with zero in head position and an empty stack. In the resulting heap, one entry is associated with grade 0 and the other is associated with grade 1, indicating that one more copy is available. By Theorem 3.17 we then get $1 \leq 0$, but this is not true for the linear types modality. Thus $\mathtt{proj}_1$ is not well-resourced when that modality is used.

4 Usage Counting for Natural Number Recursion

In the previous sections we omitted the eliminator for natural numbers. In this section we extend the syntax of our language with such a term, $\mathsf{natrec}^r_p\, A\, z\, s\, n$ (its typing and

$\boxed{\Gamma \vdash t : A}$

$$\frac{\Gamma \vdash z : A[\mathsf{zero}] \qquad \Gamma.\mathbb{N}.A \vdash s : A[\uparrow^2 \mathsf{id}, \mathsf{suc}\ \mathsf{x}_1] \qquad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{natrec}^r_p\ A\ z\ s\ n : A[n]}$$

$\boxed{\Gamma \vdash t = u : A}$

$$\frac{\begin{array}{c}\Gamma.\mathbb{N} \vdash A = A' \\ \Gamma \vdash z = z' : A[\mathsf{zero}] \qquad \Gamma.\mathbb{N}.A \vdash s = s' : A[\uparrow^2 \mathsf{id}, \mathsf{suc}\ \mathsf{x}_1] \qquad \Gamma \vdash n = n' : \mathbb{N}\end{array}}{\Gamma \vdash \mathsf{natrec}^r_p\ A\ z\ s\ n = \mathsf{natrec}^r_p\ A'\ z'\ s'\ n' : A[n]}$$

$$\frac{\Gamma \vdash z : A[\mathsf{zero}] \qquad \Gamma.\mathbb{N}.A \vdash s : A[\uparrow^2 \mathsf{id}, \mathsf{suc}\ \mathsf{x}_1]}{\Gamma \vdash \mathsf{natrec}^r_p\ A\ z\ s\ \mathsf{zero} = z : A[\mathsf{zero}]}$$

$$\frac{\Gamma \vdash z : A[\mathsf{zero}] \qquad \Gamma.\mathbb{N}.A \vdash s : A[\uparrow^2 \mathsf{id}, \mathsf{suc}\ \mathsf{x}_1] \qquad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{natrec}^r_p\ A\ z\ s\ (\mathsf{suc}\ n) = s[n, \mathsf{natrec}^r_p\ A\ z\ s\ n] : A[\mathsf{suc}\ n]}$$

$\boxed{\Gamma \vdash t \to u : A}$

$$\frac{\Gamma \vdash z : A[\mathsf{zero}] \qquad \Gamma.\mathbb{N}.A \vdash s : A[\uparrow^2 \mathsf{id}, \mathsf{suc}\ \mathsf{x}_1] \qquad \Gamma \vdash n \to n' : \mathbb{N}}{\Gamma \vdash \mathsf{natrec}^r_p\ A\ z\ s\ n \to \mathsf{natrec}^r_p\ A\ z\ s\ n' : A[n]}$$

$$\frac{\Gamma \vdash z : A[\mathsf{zero}] \qquad \Gamma.\mathbb{N}.A \vdash s : A[\uparrow^2 \mathsf{id}, \mathsf{suc}\ \mathsf{x}_1]}{\Gamma \vdash \mathsf{natrec}^r_p\ A\ z\ s\ \mathsf{zero} \to z : A[\mathsf{zero}]}$$

$$\frac{\Gamma \vdash z : A[\mathsf{zero}] \qquad \Gamma.\mathbb{N}.A \vdash s : A[\uparrow^2 \mathsf{id}, \mathsf{suc}\ \mathsf{x}_1] \qquad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{natrec}^r_p\ A\ z\ s\ (\mathsf{suc}\ n) \to s[n, \mathsf{natrec}^r_p\ A\ z\ s\ n] : A[\mathsf{suc}\ n]}$$

Figure 13: Typing, term equality and reduction rules for natrec.

reduction rules are shown in Fig. 13), and discuss how to properly count usage for it. We take the method used by Abel et al. [2023b] as our starting point. As we will see this approach fails to capture, for instance, linearity. We will instead propose a different method for usage counting for natrec, and extend the abstract machine and its correctness proof to show that the method works correctly.

4.1 Natrec-star

Abel et al. [2023b] state the usage rule[3] for natrec using an additional operator $\circledast$ named natrec-star. The operator is assumed to satisfy some distributivity properties, an "interchange" property, as well as the two inequalities $p \circledast_r q \leq p$ and $p \circledast_r q \leq q + r(p \circledast_r q)$. These are used to prove a substitution lemma and subject reduction for usage. Abel et al. present both a system with modes and one without. We focus on the system with modes,[4] in which they use (basically) the following usage rule, with $\circledast_r$ lifted pointwise to

[3]They also present an alternative usage rule [Abel et al., 2023b, Section 7.1.4]. Similar issues apply to that one.

[4]The described problems apply also to the system without modes.

contexts:

$$\frac{\gamma_{\mathsf{z}} \blacktriangleright^m z \qquad \gamma_{\mathsf{s}}.\, mp.\, mr \blacktriangleright^m s \qquad \eta \blacktriangleright^m n \qquad \theta.0 \blacktriangleright^0 A}{(\gamma_{\mathsf{z}} \wedge \eta) \circledast_r (\gamma_{\mathsf{s}} + p\eta) \blacktriangleright^m \mathsf{natrec}^r_p\, A\, z\, s\, n}$$

Here z and s are the zero and suc branches, respectively, and n is the natural number argument. The s argument binds two variables, one corresponding to the predecessor of the natural number, and one corresponding to the recursive call. These are assigned the grades p and r, respectively ($p$redecessor and $r$ecursive call). Similarly to other eliminators there is an explicit motive A.

A detailed explanation of this rule is given by Abel et al. Their focus is on proving that usage is preserved by reduction, and that erasure "works". However, the rule is not sufficient to express linearity correctly.

One problem arises when considering addition of natural numbers, defined in the following way: $\mathtt{plus}\ k\ n = \mathsf{natrec}^1_0\ \mathbb{N}\ k\ (\mathsf{suc}\,\mathsf{x}_0)\ n$. This corresponds to the following Agda program, which one might expect to be linear in both arguments:

```
plus : ℕ → ℕ → ℕ
plus k zero    = k
plus k (suc n)= suc (plus k n)
```

For the linearity modality, however, the suggested definition of $\circledast$ is $p \circledast_0 q = p \wedge q$ and $p \circledast_1 q = p + \omega q$ and $p \circledast_\omega q = \omega(p + q)$, which is the (pointwise) largest possible definition [Abel et al., 2023a]. This would give `plus` the following admissible usage rule:

$$\frac{\gamma \blacktriangleright^m k \qquad \delta \blacktriangleright^m n}{\gamma \wedge \delta \blacktriangleright^m \mathtt{plus}\ k\ n}$$

For $\mathtt{plus}\,\mathsf{x}_0\,\mathsf{x}_1$ the usage count of both x_0 and x_1 is then $0 \wedge 1 = \omega$. This is safe but suboptimal since it fails to capture linearity. A more problematic situation is provided by $\mathtt{plus}\,\mathsf{x}_0\,\mathsf{x}_0$, for which the usage count is $1 \wedge 1 = 1$, despite the fact that the variable is used twice (i.e. ω times). One could perhaps solve this by choosing a different definition of natrec-star than the largest legal one, in such a way that the usage count becomes ω for $\mathtt{plus}\,\mathsf{x}_0\,\mathsf{x}_0$. However, with any definition of natrec-star the usage count for $\mathtt{plus}\,\mathsf{x}_0\,\mathsf{x}_1$ will be ω for both x_0 and x_1: above the largest definition is used, so no other definition would work better.

4.2 A Resource-Correct Usage Rule

Let us now define an alternative usage rule for natrec. If the pattern of the other eliminators for "positive" types is followed, then the usage count for $\mathsf{natrec}^r_p\, A\, z\, s\, n$ should be the usage count for n scaled by some factor plus the usage contributions from z and s (as usual there is no contribution from the motive A). If we leave the other parts of the usage rule above unchanged, then we get a rule of the following form:

$$\frac{\gamma_{\mathsf{z}} \blacktriangleright^m z \qquad \gamma_{\mathsf{s}}.\, mp.\, mr \blacktriangleright^m s \qquad \eta \blacktriangleright^m n \qquad \theta.0 \blacktriangleright^0 A}{x\eta + \chi \blacktriangleright^m \mathsf{natrec}^r_p\, A\, z\, s\, n}$$

Here x is some grade indicating the number of copies of n that are used and χ is a context representing the usage contributions from z and s.

The main difficulty in finding proper values for x and χ comes from the recursion: whatever usage counts we choose to assign, they should work for all values of n, i.e. regardless of how deep the recursion is. (Perhaps one could have some kind of "dependent grades", but that is not a design we are aiming for at the moment.) We can view this as a

form of branching where each branch corresponds to a given number of unfoldings before the base case is reached and the total usage count should be compatible with any branch.

Starting with χ, we let χ_i be the usage contribution by z and s given i unfoldings. First, χ_0 corresponds to the zero case where only z is used so we get $\chi_0 = \gamma_{\mathsf{z}}$. Similarly, χ_{i+1} corresponds to a successor case which uses s once (with γ_{s} resources) and the recursive call r times. A single recursive call uses χ_i resources, so we get $\chi_{i+1} = \gamma_{\mathsf{s}} + r\chi_i$ (assuming a substitution lemma like the one proven by Abel et al. [2023b]). Doing the same analysis for x we get $x_0 = 1$ since in the base case n is "consumed fully" by the match, and $x_{i+1} = p + rx_i$ since n is used p times by s and the recursive call is again used r times.

Normally we would be able to use the meet operator to combine the usage counts of several branches in a way that ensures compatibility with them all but for an infinite number of branches this is not an option. Instead we let x and χ be the greatest lower bounds of x_i and χ_i, respectively. Given a grade sequence (function from Nat to $\mathcal{M}$) p_i we let $p = \bigwedge_i p_i$ mean that p_i has a greatest lower bound p,[5] and similarly for contexts.

In conclusion, we get the following usage rule:

$$\frac{\gamma_{\mathsf{z}} \blacktriangleright^m z \qquad \gamma_{\mathsf{s}}.\, mp.\, mr \blacktriangleright^m s \qquad \eta \blacktriangleright^m n \qquad \theta.\, 0 \blacktriangleright^0 A}{x\eta + \chi \blacktriangleright^m \mathsf{natrec}^r_p\ A\ z\ s\ n}\ x = \textstyle\bigwedge_i \mathrm{nr}_i\ r\ 1\ p,\ \ \chi = \bigwedge_i \mathrm{nr}_i\ r\ \gamma_{\mathsf{z}}\ \gamma_{\mathsf{s}}$$

Here we have defined a function nr_i as follows (and allow it to be lifted pointwise to contexts in its last two arguments):

$$\begin{aligned} \mathrm{nr}_0\ r\ q_{\mathsf{z}}\ q_{\mathsf{s}} &= q_{\mathsf{z}} \\ \mathrm{nr}_{i+1}\ r\ q_{\mathsf{z}}\ q_{\mathsf{s}} &= q_{\mathsf{s}} + r \cdot \mathrm{nr}_i\ r\ q_{\mathsf{z}}\ q_{\mathsf{s}} \end{aligned}$$

With this rule we can prove subject reduction for the usage relation given some additional assumptions for the modality:

Definition 4.1. A modality is said to have well-behaved greatest lower bounds if it satisfies the following properties[6]:

- If $p = \bigwedge_i p_i$ then $q + p = \bigwedge_i (q + p_i)$.
- If $p = \bigwedge_i p_i$ then $q \cdot p = \bigwedge_i (q \cdot p_i)$.
- If $p = \bigwedge_i p_i$ then $p \cdot q = \bigwedge_i (p_i \cdot q)$.
- If $p = \bigwedge_i \mathrm{nr}_i\ r\ q_{\mathsf{z}}\ q_{\mathsf{s}}$ and $p' = \bigwedge_i \mathrm{nr}_i\ r\ q'_{\mathsf{z}}\ q'_{\mathsf{s}}$ then there is some $\hat{p}$ such that $p + p' \leq \hat{p}$ and $\hat{p} = \bigwedge_i \mathrm{nr}_i\ r\ (q_{\mathsf{z}} + q'_{\mathsf{z}})\ (q_{\mathsf{s}} + q'_{\mathsf{s}})$.

The first three properties are essentially generalizations of distributivity over meet to greatest lower bounds of grade sequences. The final property corresponds to the following "sub-interchange" property between addition and meet which follows from the modality laws: $(p \wedge q) + (p' \wedge q') \leq (p + p') \wedge (q + q')$. Unlike the preceding three properties we restrict this property to sequences of the form nr_i. This is for practical purposes as proving constructively that our main modalities satisfy the sub-interchange property for arbitrary sequences does not seem feasible. All modalities we have mentioned above satisfy these properties.

We can prove subject reduction for modalities with well-behaved greatest lower bounds, using the following "characteristic inequalities" (which also appeared above for natrec-star): if $q = \bigwedge_i \mathrm{nr}_i\ r\ q_{\mathsf{z}}\ q_{\mathsf{s}}$ then $q \leq q_{\mathsf{z}}$ and $q \leq q_{\mathsf{s}} + rq$. In addition, the main results of Abel

[5] That is, $p \leq p_i$ for all i and for any q such that $q \leq p_i$ for all i we have $q \leq p$.

[6] Addition or multiplication of a sequence by a constant grade is defined as pointwise addition or multiplication.

et al. [2023b] still hold with this rule for such modalities, in particular the substitution lemma and correctness of erasure.

It remains to show that the kind of resource correctness that was discussed in Section 3 also holds. However, let us first return to `plus`, for which we now get the following usage rule:

$$\frac{\gamma \blacktriangleright^m k \qquad \delta \blacktriangleright^m n}{p\delta + \eta \blacktriangleright^m \texttt{plus}\ k\ n}\ p = \bigwedge_i \mathrm{nr}_i\ 1\,1\,0,\ \ \eta = \bigwedge_i \mathrm{nr}_i\ 1\,\gamma\,\mathbf{0}$$

For any modality the greatest lower bounds of $\mathrm{nr}_i\ 1\,1\,0$ and $\mathrm{nr}_i\ 1\,\gamma\,\mathbf{0}$ are 1 and γ, respectively, so the usage rule simplifies to the following one:

$$\frac{\gamma \blacktriangleright^m k \qquad \delta \blacktriangleright^m n}{\gamma + \delta \blacktriangleright^m \texttt{plus}\ k\ n}$$

Note that this rule indicates that both arguments are used once.

4.3 Resource Correctness for the Natural Number Eliminator

Having shown subject reduction we move on to showing that we get the same kind of resource correctness as before also in the presence of natrec by extending the abstract machine accordingly. Our proof assumes that the modality has well-behaved greatest lower bounds, so below we restrict ourselves to such modalities.

A natural first step is to add a corresponding continuation $\mathsf{natrec}^r_p\, A\, z\, s \bullet [\rho]$ and define its multiplicity. For completeness we also note that application of the natrec continuation to a term is defined in the following way: $(\!|\mathsf{natrec}^r_p\, A\, z\, s \bullet [\rho]|\!)\, t = \mathsf{natrec}^r_p\, A[\Uparrow\rho]\, z[\rho]\, s[\Uparrow^2\rho]\, t$. The usage rule we defined above indicates that the multiplicity should be the greatest lower bound of $\mathrm{nr}_i\ r\,1\,p$, so we extend the multiplicity definition above (Fig. 9) with:

$$\frac{}{|\mathsf{natrec}^r_p\, A\, z\, s \bullet [\rho]| \equiv q}\ q = \bigwedge_i \mathrm{nr}_i\ r\,1\,p$$

Since this greatest lower bound does not necessarily exist for all modalities, the stack multiplicity is no longer guaranteed to exist. It is still the case, however, that stack multiplicity is functional since the greatest lower bound is unique if it does exist. We therefore still write $|S|$ as before for the unique multiplicity of a stack, with an implicit assumption that this multiplicity exists.

We get three new reduction rules: one for putting the continuation on the stack and two for popping it off, corresponding to the zero and successor case, see Fig. 14. In the successor case, two new entries are put on the heap. The first contains q (scaled by the current stack multiplicity) copies of the natural number, where $q = \bigwedge_i \mathrm{nr}_i\ r\,1\,p$ corresponds to all of the number's uses (both by s and by the recursive call). The second entry contains r copies of the recursive call, which is not yet computed since we are in a call-by-name setting. Its natural number argument refers to the previous entry, prompting a lookup if the recursive call gets evaluated. The reduction relation is still deterministic, because of the constraint on the reduction rule that puts $\mathsf{suc} \bullet$ on the stack.

A consequence of the addition of natrec is that the circumstances under which reduction can get stuck are now slightly different. A state with a value in head position and a non-empty stack is now stuck if it is not the case that (1) the stack and head match and (2) the stack multiplicity exists. Similarly to what we did in Section 3, we will show that for well-resourced and well-typed states, this cannot happen.

$\boxed{s \to_e s',\ s \to_v s'}$

$$\langle H;\ \mathsf{natrec}^r_p\ A\ z\ s\ n;\ \rho;\ S\rangle \ \to_e\ \langle H;\ n;\ \rho;\ \mathsf{natrec}^r_p\ A\ z\ s \bullet [\rho].\ S\rangle$$

$$\begin{array}{rl}
\langle H;\ \mathsf{zero};\ \rho;\ \mathsf{natrec}^r_p\ A\ z\ s \bullet [\rho'].\ S\rangle & \to_v\ \langle H;\ z;\ \rho';\ S\rangle \\
\langle H;\ \mathsf{suc}\ t;\ \rho;\ \mathsf{natrec}^r_p\ A\ z\ s \bullet [\rho'].\ S\rangle & \to_v\ \langle H';\ s;\ \Uparrow^2\rho';\ S[\uparrow^2\mathsf{id}]\rangle \\
\quad \text{if} \quad q = \bigwedge_i \mathrm{nr}_i\ r\ 1\ p & \\
\quad\quad H' = H.\,(t,\rho)^{|S|q}.\,(\mathsf{natrec}^r_p\ A[\Uparrow\uparrow\mathsf{id}]\ z[\uparrow\mathsf{id}]\ s[\Uparrow^2\uparrow\mathsf{id}]\ \mathsf{x}_0, \Uparrow\rho')^{|S|r} &
\end{array}$$

Figure 14: Extension of the machine reductions for eliminators and values from Fig. 10 by natrec.

The usage relation for continuations, Fig. 11, is extended with the following rule:

$$\frac{\gamma_{\mathsf{z}} \blacktriangleright^m z \qquad \gamma_{\mathsf{s}}.\, mp.\, mr \blacktriangleright^m s \qquad \eta.\, 0 \blacktriangleright^0 A}{\chi[\rho] \blacktriangleright^m \mathsf{natrec}^r_p\ A\ z\ s \bullet [\rho]}\ \chi = \textstyle\bigwedge_i \mathrm{nr}_i\ r\ \gamma_{\mathsf{z}}\ \gamma_{\mathsf{s}}$$

This rule is obtained from the usage rule for the corresponding term through the removal of the contribution from the scrutinee (i.e. the natural number argument). The only difference from the pattern established above is that we have included the assumption that the motive is well-resourced. This is used to conclude that the recursive call is well-resourced when added to the heap.

We can now see that for well-resourced states, the stack multiplicity always exists. We get that well-resourced states do not get stuck due to the stack multiplicity not existing, and that the theorems in Section 3.3 still hold.

Adding typing for the natrec continuation is similarly straightforward. We use the following typing rule:

$$\frac{\begin{array}{c}\epsilon \vdash z[H \circ \rho] : A[\mathsf{zero} \circ \Uparrow H \circ \Uparrow\rho] \\ \epsilon.\,\mathbb{N}.\, A[\Uparrow H \circ \Uparrow\rho] \vdash s[\Uparrow^2 H \circ \Uparrow^2\rho] : A[(\uparrow^2\mathsf{id}, \mathsf{suc}\ \mathsf{x}_1) \circ \Uparrow H \circ \Uparrow\rho]\end{array}}{H \vdash \mathsf{natrec}^r_p\ A\ z\ s \bullet [\rho] : (t : \mathbb{N}) \rightsquigarrow A[t[H] \circ \Uparrow H \circ \Uparrow\rho]}$$

Again, this is exactly what is expected following the pattern established above. We can now prove everything from Sections 3.4 to 3.6 as stated. In particular, the main resource correctness result, Theorem 3.17, still holds.

To wrap this section up, let us return to the addition example presented above. Let us evaluate $\mathtt{plus}\, k\, n$, for closed numbers k and n. The evaluation progresses by recursion over n, with each iteration inserting a natural number in the heap as well as an (unevaluated) recursive call: both assigned grade 1. Note that even though the natural number is unused by the successor branch ($p = 0$), it is used once by the recursive call through matching. The correctness theorem shows that once this evaluation finishes, lookups will have been performed in such a way that all these entries will have remaining uses bounded by 0. In particular, for the linearity modality it must then be the case that there are 0 copies left since one cannot obtain ω from (repeated) subtraction from 1.

4.4 Usage Counting for the Natural Number Eliminator

We have shown in Section 4 that our proposed usage rule for natrec gives an accurate estimate of the usage count, the usage rule itself is perhaps not very enlightening as it is typically not immediately clear what the greatest lower bound of $\mathrm{nr}_i\ r\ q_{\mathsf{z}}\ q_{\mathsf{s}}$ is (or whether it even exists). As it turns out, for all mentioned modalities these limits do exist and, as we will see for the linear and affine types modalities, are ω in most cases. This should not

be too surprising as the exact usage count can depend on the specific number of recursive calls (i.e. on the eliminator's natural number argument), and little can therefore be said about the usage count in the general case. However, for a few values of r, q_{z} and q_{s} we get a greatest lower bound that is different from ω. In fact we already saw one such example, when we concluded that `plus` uses both its arguments linearly.

For the linear types modality the greatest lower bound of $\mathrm{nr}_i\ r\ q_{\mathsf{z}}\ q_{\mathsf{s}}$ will be 0 if $\mathrm{nr}_i\ r\ q_{\mathsf{z}}\ q_{\mathsf{s}} = 0$ for all i and likewise 1 if $\mathrm{nr}_i\ r\ q_{\mathsf{z}}\ q_{\mathsf{s}} = 1$ for all i. The affine types modality is a little more lenient. It is still the case that the greatest lower bound of $\mathrm{nr}_i\ r\ q_{\mathsf{z}}\ q_{\mathsf{s}}$ will be 0 if $\mathrm{nr}_i\ r\ q_{\mathsf{z}}\ q_{\mathsf{s}} = 0$ but we now get 1 as the greatest lower bound if $\mathrm{nr}_i\ r\ q_{\mathsf{z}}\ q_{\mathsf{s}} = 1$ or 0 for all i. In both cases a straightforward analysis reveals that we get 0 as the greatest lower bound only for $\mathrm{nr}_i\ r\,0\,0$ and 1 for $\mathrm{nr}_i\ 1\,1\,0$ and $\mathrm{nr}_i\ 0\,1\,1$. For the affine types modality, we additionally have 1 as the greatest lower bound of $\mathrm{nr}_i\ 0\,0\,1$ and $\mathrm{nr}_i\ 0\,1\,0$.

The practical consequence of this is that `natrec` uses its natural number argument linearly only when $p = 0$ and $r = 1$ or if $p = 1$ and $r = 0$ (recall that the number of uses of the natural number is given by the greatest lower bound of $\mathrm{nr}_i\ r\,1\,p$). The former case corresponds to the successor branch using the recursive call linearly and using the natural number only in erased positions (or not at all). The addition function discussed above is an example of this. The latter case conversely corresponds to the natural number being used linearly and the recursive call being erased. An example of this is the predecessor function $\mathtt{pred}\ n = \mathsf{natrec}^{0}_{1}\ \mathbb{N}\ \mathsf{zero}\ \mathsf{x}_1\ n$. This corresponds to the following Agda program, which one might expect to be linear:

```
pred : ℕ → ℕ
pred zero    = zero
pred (suc n)= n
```

This is indeed the case as the following usage rule is admissible:

$$\frac{\gamma \blacktriangleright^{m} n}{\gamma \blacktriangleright^{m} \mathtt{pred}\ n}$$

In the affine types case, both linear cases are naturally also considered affine uses together with the case $p = r = 0$, when both the natural number and the recursive call are erased in the successor branch. We can also see that the natural number argument is never considered to be erased.

We can similarly see under what conditions the zero and successor branches are used linearly. Associating γ_{z} with the usage counts of the zero branch and γ_{s} with the successor branch we have that for $r = 0$, their contribution (χ in the usage rule above) is $\gamma_{\mathsf{z}} \wedge \gamma_{\mathsf{s}}$. In this case the recursive calls are not used so only one of the zero and successor branch will be used (but we do not know which). For $r = 1$ we get $\gamma_{\mathsf{z}} + \omega\gamma_{\mathsf{s}}$, where every recursive call is used once so the zero case will inevitably be used once while the successor cases could be used any number of times. Finally, when $r = \omega$, the recursive calls are used an unrestricted number of times so we get $\omega(\gamma_{\mathsf{z}} + \gamma_{\mathsf{s}})$, using both branches an unrestricted number of times.

5 Resource Correctness for Open Programs

So far we have assumed that the programs we are evaluating have been closed. In this section we will modify the abstract machine to also evaluate open terms. Of course, given an open term t for which we have concrete values for each free variable one can simply construct a heap H containing them and evaluate the state $\langle H; t; \mathsf{id}; \epsilon\rangle$. The situation we will consider here is when this is not case, such as when some "postulated" properties are

used in t. Naturally, evaluation of such a program would get stuck whenever postulates are encountered, so we will assume that any free variables are only used in erased positions. This is similar to the erasure case study of Abel et al. [2023b].

For simplicity, we will keep the same correspondence between variable and pointer indices and will therefore let heaps contain a kind of "dummy entry", $\circ$, representing that a pointer does not point at anything:

$$H ::= \epsilon \mid H.\,c \mid H.\,\circ \qquad \text{Heaps}$$

The substitution interpretation of heaps is updated with the following equation for dummy entries: $(\!|H.\,\circ|\!) = \Uparrow(\!|H|\!)$. This means that variables corresponding to dummy entries are untouched (apart from possible re-indexing). The result of applying a heap as a substitution to a term, or translating a state to a term, is thus a term that contains as many free variables as there are dummy entries in the heap. Similarly, the initial state of t (with n free variables) is now $\langle \epsilon.\,\circ.\,\ldots.\,\circ;\ t;\ \mathsf{id};\ \epsilon \rangle$, where the heap consists (only) of n dummy entries.

The semantics of the abstract machine is unchanged. Notably, variable lookup fails for dummy entries so evaluation can get stuck for states with variables in head position, either if there are not enough copies of the corresponding entry, or if the corresponding entry is a dummy entry (but we will show that neither case is possible for well-formed states).

Typing for states is mostly unchanged, but since states no longer correspond to closed terms, checking is no longer done under the empty context, but rather under some context Δ that contains the types of all preceding dummy entries. Consequently, the typing judgment for heaps now reads $\Delta \vdash H : \Gamma$, with Γ representing the types of the entries as before, and Δ representing the types of dummy entries,[7] see Fig. 15. Similarly, the judgments for continuations and stacks now read $\Delta; H \vdash e : (t : A) \rightsquigarrow B$ and $\Delta; H \vdash S : (t : A) \rightsquigarrow B$, see Fig. 15.

As before, typing is preserved by reduction and states with a value in head position and a non-empty stack always reduce if the stack multiplicity exists (i.e. Theorems 3.6 and 3.7 still hold).

Usage for states is also mostly unchanged. The only difference is that the usage relation for heaps is extended with the following rule to account for the dummy entries, which are naturally considered to contain 0 copies (of nothing):

$$\frac{\gamma \blacktriangleright H}{\gamma.\,0 \blacktriangleright H.\,\circ}$$

We would like the properties we proved in Section 3.3 to still hold. For Theorem 3.4 (usage is preserved by reduction) this is indeed the case. However, showing that heap lookups always succeed for well-resourced states (Theorem 3.5) poses some difficulty, as there is now an additional way for lookup to fail. An example of this is provided by the program $\mathsf{unitrec}_0\,\mathbb{N}\,\mathsf{x}_0\,\mathsf{zero}$, which matches on a possibly erased variable x_0 before returning zero, roughly corresponding to the following Agda program:

```
example : @0 ⊤ → ℕ
example ⋆ = zero
```

Before this would lead to a lookup of 0 copies of the corresponding entry, but now it can also lead to a failed lookup, if x_0 refers to a dummy entry.

[7] Thinking of the heap as a substitution, this can be read as H taking terms from context Γ to context Δ.

$\boxed{\Delta \vdash H : \Gamma}$

$$\frac{}{\epsilon \vdash \epsilon : \epsilon} \qquad \frac{\Delta \vdash H : \Gamma \qquad \Delta \vdash t[H \circ \rho] : A[H]}{\Delta \vdash H.\,(t, \rho)^p : \Gamma.\,A} \qquad \frac{\Delta \vdash H : \Gamma \qquad \Gamma \vdash A}{\Delta.\,A[H] \vdash H.\circ : \Gamma.\,A}$$

$\boxed{\Delta; H \vdash e : (t : A) \rightsquigarrow B}$

$$\frac{\Delta.\,A \vdash B \qquad \Delta \vdash u[H \circ \rho] : A}{\Delta; H \vdash \bullet^p u\,[\rho] : (t : \Pi^q_p\,A\,B) \rightsquigarrow B[u[H \circ \rho]]} \qquad \frac{\Delta.\,A \vdash B}{\Delta; H \vdash \mathsf{fst}_p\,\bullet : (t : \Sigma^q_{\&,p}\,A\,B) \rightsquigarrow A}$$

$$\frac{\Delta.\,A \vdash B}{\Delta; H \vdash \mathsf{snd}_p\,\bullet : (t : \Sigma^q_{\&,p}\,A\,B) \rightsquigarrow B[(\mathsf{fst}_p\,t)[H]]}$$

$$\frac{\Delta.\,A.\,B \vdash u[\Uparrow^2 H \circ \Uparrow^2 \rho] : C[\uparrow^2 \mathsf{id}, ({}^p\mathsf{x}_1, \mathsf{x}_0)_\otimes \circ \Uparrow H \circ \Uparrow \rho] \qquad \Delta.\,\Sigma^{q'}_{\otimes,p}\,A\,B \vdash C[\Uparrow H \circ \Uparrow \rho]}{\Delta; H \vdash \mathsf{prodrec}^p_r\,C \bullet u\,[\rho] : (t : \Sigma^q_{\otimes,p}\,A\,B) \rightsquigarrow C[t[H] \circ \Uparrow H \circ \Uparrow \rho]}$$

$$\frac{\begin{array}{c}\Delta \vdash z[H \circ \rho] : A[\mathsf{zero} \circ \Uparrow H \circ \Uparrow \rho] \\ \Delta.\,\mathbb{N}.\,A[\Uparrow H \circ \Uparrow \rho] \vdash s[\Uparrow^2 H \circ \Uparrow^2 \rho] : A[(\uparrow^2 \mathsf{id}, \mathsf{suc}\,\mathsf{x}_1) \circ \Uparrow H \circ \Uparrow \rho]\end{array}}{\Delta; H \vdash \mathsf{natrec}^r_p\,A\,z\,s \bullet [\rho] : (t : \mathbb{N}) \rightsquigarrow A[t[H] \circ \Uparrow H \circ \Uparrow \rho]}$$

$$\frac{\Delta \vdash u[H \circ \rho] : A[\star_\otimes \circ \Uparrow H \circ \Uparrow \rho] \qquad \Delta.\,\top_\otimes \vdash A[\Uparrow H \circ \Uparrow \rho]}{\Delta; H \vdash \mathsf{unitrec}_p\,A \bullet u\,[\rho] : (t : \top_\otimes) \rightsquigarrow A[t[H] \circ \Uparrow H \circ \Uparrow \rho]}$$

$$\frac{\Delta \vdash A[H \circ \rho]}{\Delta; H \vdash \mathsf{emptyrec}_p\,A \bullet [\rho] : (t : \bot) \rightsquigarrow A[H \circ \rho]}$$

$\boxed{\Delta; H \vdash S : (t : A) \rightsquigarrow B}$

$$\frac{}{\Delta; H \vdash \epsilon : (t : A) \rightsquigarrow A} \qquad \frac{\Delta; H \vdash e : (t : A) \rightsquigarrow B \qquad \Delta; H \vdash S : (\llparenthesis e \rrparenthesis\,t : B) \rightsquigarrow C}{\Delta; H \vdash e.\,S : (t : A) \rightsquigarrow C}$$

$\boxed{\Delta \vdash s : A}$

$$\frac{\Delta \vdash H : \Gamma \qquad \Delta \vdash t[H \circ \rho] : B \qquad \Delta; H \vdash S : (t[\rho] : B) \rightsquigarrow A}{\Delta \vdash \langle H;\, t;\, \rho;\, S \rangle : A}$$

Figure 15: Typing for heaps, $\Delta \vdash H : \Gamma$, continuations, $\Delta; H \vdash e : (t : A) \rightsquigarrow B$, stacks, $\Delta; H \vdash S : (t : A) \rightsquigarrow B$, and states, $\Delta \vdash s : A$.

In a well-resourced heap, $\gamma \blacktriangleright H$, the entry of γ corresponding to a dummy entry is always 0. Thus attempted lookups of dummy entries will only occur when the stack multiplicity is 0, which can happen if the stack contains either $\mathsf{unitrec}_0\,A \bullet u\,[\rho]$, $\mathsf{prodrec}^p_0\,A \bullet u\,[\rho]$ or $\mathsf{emptyrec}_0\,A \bullet [\rho]$. (Note that the multiplicity of natrec is never 0 since the greatest lower bound of $\mathrm{nr}_i\,r\,1\,p$ is not 0 for any modality with a well-behaved zero.) These all correspond to having encountered an erased match. The problems that arise with erased matches are similar to those encountered in the erasure case study of Abel et al. [2023b]. Like them we disallow certain erased matches when evaluating open terms: we do not

allow any occurrences of $\mathsf{prodrec}_0^r\ A\ t\ u$ or $\mathsf{unitrec}_0\ A\ t\ u$ in non-erased positions. For the empty type eliminator we can make the same restriction, but occurrences of emptyrec can also be ruled out by assuming that the program is evaluated in a consistent context (a context Γ is consistent if there is no term t such that $\Gamma \vdash t : \bot$). We will take this route here, but we note that whenever we assume a consistent context we could alternatively assume that there are no erased matches in non-erased positions for emptyrec.

With this in mind, we now get the following variant of Theorem 3.5:

THEOREM 5.1. *If Δ is consistent, erased matches are disallowed, $\blacktriangleright\ \langle H; \mathsf{x}_i; \rho; S\rangle$ and $\Delta \vdash \langle H; \mathsf{x}_i; \rho; S\rangle : A$ then there is an entry e and heap H' such that $H \vdash \mathsf{y}_{i[\rho]} \mapsto^{|S|} e\ ;\ H'$.*

For similar reasons, we widen the definition of normal form states to also include states with a variable in head position that points to a dummy entry. Theorem 3.11 then holds as stated and the bisimilarity properties hold with the change that programs are evaluated in an open context Δ.

Using the same argument as before, we then get almost the same resource correctness property as above. The only differences are some added assumptions about consistency and erased matches and that the resulting heap may additionally contain dummy entries.

THEOREM 5.2. *If Δ is consistent, erased matches for* prodrec *and* unitrec *are disallowed, $\mathbf{0}\ \blacktriangleright^1\ t$, and $\Delta \vdash t : \mathbb{N}$, then there is a numeral n and heap H such that $\langle t\rangle \twoheadrightarrow^* \langle H; n; \rho; \epsilon\rangle$ and the number of copies of each (non-dummy) entry in H is bounded by* 0.

As a reminder, an erased match here refers to an occurrence of $\mathsf{prodrec}_0^p$ or $\mathsf{unitrec}_0$. These can be disallowed using the corresponding side conditions of the usage relation. Note that the assumptions of this theorem (a consistent context, no erased matches, all free variables are erasable) are similar to those of the proof of soundness of erasure given by Abel et al. [2023b, Theorem 6.13]. All the assumptions are necessary: If any one assumption is removed, then the final state of the evaluation does not necessarily contain a numeral in head position, and the stack is not necessarily empty. Furthermore, for modalities satisfying $1 \not\leq 0$ (for instance the linear types modality) the resource correctness result also does not necessarily hold, that is, the grades associated with the entries of the final heap are not necessarily bounded by 0.

We can thus draw basically the same conclusions for programs that use erased assumptions as we did for closed programs, given that such assumptions are consistent and that certain erased matches are not allowed. We could alternatively drop the assumption that Δ is consistent and instead disallow erased matches for emptyrec and get the same result.

6 Usage for General Recursive Data Types

Our only recursive data type is the type of natural numbers, and the formulation of its usage rule is somewhat different from those of the other types. Although the investigation we did in Section 4 was specific to natrec we believe that a similar approach is feasible also for (some) other recursive data types, and we sketch in this section how this could perhaps be done for the simple case of a type of binary trees. Note however that the formalization does not contain this data type, and the ideas we sketch here have not been formalized.

The type we consider is that of binary trees with no data[8] and an eliminator treerec in the same style as natrec, corresponding to the following Agda code:

[8]Though not a particularly interesting type it is not useless—the formalization uses well-founded induction over such trees for some arguments.

```
data Tree : Set where
  leaf : Tree
  node: Tree → Tree → Tree

treerec :
  (P : Tree → Set) → P leaf →
  ((t₁ t₂ : Tree) → P t₁ → P t₂ → P (node t₁ t₂)) →
  (t : Tree) → P t
treerec P l n leaf       = l
treerec P l n (node t₁ t₂)= n t₁ t₂ (treerec P l n t₁) (treerec P l n t₂)
```

We annotate treerec with two grades p and r, where p corresponds to the uses of the two sub-trees and r to the two recursive calls. The two subtrees are not given individual grades, analogously to the usage rule for prodrec, in which the two components of a pair are not given separate grades (except that one grade is scaled).

Following the style used for natrec, the usage count for the eliminator should consist of one part pertaining to the usage contributions of the tree, and one part pertaining to the contributions of the node and leaf branches, that is something like the following:

$$\frac{\gamma_\mathsf{l} \blacktriangleright^m l \qquad \gamma_\mathsf{n}.\, mp.\, mp.\, mr.\, mr \blacktriangleright^m n \qquad \eta \blacktriangleright^m t \qquad \theta.0 \blacktriangleright^0 A}{x\eta + \chi \blacktriangleright^m \mathsf{treerec}^r_p\, A\, l\, n\, t}$$

Here we should find some suitable choices for x and χ, where x depends on p and r, and χ depends on r, γ_l and γ_n.

For natrec we found x and χ by considering all possible unfoldings, taking the greatest lower bound of the usage counts corresponding to all cases. We would like to do the same here but listing all cases is a bit more tricky than for the natural numbers where a suitable order was quite obvious. There are probably several ways in which one can do this but we will enumerate the cases based on the size i (number of nodes) of the tree. As before we will let x_i and χ_i denote the uses of the tree and the usage contribution of the leaf and node branches for trees of size i, respectively.

When $i = 0$ we have the empty tree, so the leaf branch l is used once, and the tree is used once through matching. We get that $\chi_0 = \gamma_\mathsf{l}$ and $x_0 = 1$. In the node case we have a tree of size $i + 1$, consisting of two sub-trees with sizes j and k such that $i = j + k$. Here the node branch is used once, providing γ_n resources, and the two recursive calls are used r times, providing $r\chi_j$ and $r\chi_k$ contributions, respectively. In total we get $\gamma_\mathsf{n} + r(\chi_j + \chi_k)$. Since the size distributions of the two subtrees is unknown, we take the *min-plus convolution* $\bigwedge_{j,k}^{i=j+k}(\chi_j + \chi_k)$, i.e., the meet over the possible distributions, to ensure compatibility in any case. This gives $\chi_{i+1} = \gamma_\mathsf{n} + r \bigwedge_{j,k}^{i=j+k}(\chi_j + \chi_k)$. The uses of the tree argument similarly give p uses of t and r uses of the recursive calls, so we end up with $x_{i+1} = p + r \bigwedge_{j,k}^{i=j+k}(x_j + x_k)$.

As for natrec, the idea is to choose x and χ to be compatible with all x_i and χ_i. Taking the greatest lower bound as before we get the following usage rule:

$$\frac{\gamma_\mathsf{l} \blacktriangleright^m l \quad \gamma_\mathsf{n}.\, mp.\, mp.\, mr.\, mr \blacktriangleright^m n \quad \eta \blacktriangleright^m t \quad \theta.0 \blacktriangleright^0 A}{x\eta + \chi \blacktriangleright^m \mathsf{treerec}^r_p\, A\, l\, n\, t} \left\{ \begin{array}{l} x = \bigwedge_i \mathrm{tr}_i\ r\, 1\, p \\ \chi = \bigwedge_i \mathrm{tr}_i\ r\, \gamma_\mathsf{l}\, \gamma_\mathsf{n} \end{array} \right.$$

Here we have defined tr_i analogously to nr_i:

$$\begin{aligned} \mathrm{tr}_0\ r\, q_\mathsf{l}\, q_\mathsf{n} &= q_\mathsf{l} \\ \mathrm{tr}_{i+1}\ r\, q_\mathsf{l}\, q_\mathsf{n} &= q_\mathsf{n} + r \cdot \textstyle\bigwedge_{j+k=i}(\mathrm{tr}_j\ r\, q_\mathsf{l}\, q_\mathsf{n} + \mathrm{tr}_k\ r\, q_\mathsf{l}\, q_\mathsf{n}) \end{aligned}$$

We suspect that using this usage rule it is possible to both show subject reduction and to extend the abstract machine argument, and believe that the same technique is applicable to (some) other recursive types as well. However, we acknowledge that we have not proved these things, and we might have missed something.

7 Related Work

This work adds to a large collection of theoretical investigations of graded type theory that occupy various points in the design space in regards to, for instance, the types that are included, support for erased matches, and usage counting in types as well as terms [Brunel et al., 2014; Ghica and Smith, 2014; McBride, 2016; Atkey, 2018; Abel and Bernardy, 2020; Moon et al., 2021; Choudhury et al., 2021; Abel et al., 2023b]. The dependently typed functional language Idris 2 [Brady, 2021] is a practical implementation of graded type theory, incorporating many of the language features that we investigate in this work, albeit restricted to the one-none-many modality of Quantitative Type Theory (QTT) [Atkey, 2018].

We build on the formalization presented by Abel et al. [2023b], and do not repeat the discussion of related work presented in that text. Here we discuss correctness for graded modal types, in particular for quantitative interpretations, and we also discuss graded type theories with support for some form of inductive data types.

7.1 Resource Correctness

Our correctness proof is in large part based on the work of Choudhury et al. [2021] who, like us, define an abstract machine that tracks usage counts for variables and prevents lookups if allowed usage would be exceeded. In many respects we follow their approach; in particular, we use call-by-name reduction, we use similar methods to track resources in heaps and to check that sufficient resources are available in each evaluation step, and we base our correctness proof on a bisimilarity argument. Yet in some respects, we departed from their design. For one, in contrast to their big-step semantics we define a small-step semantics that handles continuations explicitly and is thus perhaps closer to an actual machine. We also use slightly different approaches for tracking variable uses. For instance, while our use of a stack-based machine allows us to infer how many copies that are currently being evaluated, their reduction relation is annotated with a grade for this purpose. Heap lookups are also defined differently where they allow lookups of r copies whenever there are $q + r$ copies available and the resulting heap has q copies remaining. While this is similar to our case, this does not uniquely determine the number of remaining copies. We defined subtraction by always taking the least such q, allowing our reduction to be deterministic, though we could likely have gotten similar results for a reduction that is non-deterministic in the grade updates of the heap as well. Another notable difference is that we work in somewhat different settings. In particular, unlike Choudhury et al. [2021], we cover natural numbers, strong Σ-types, and an empty type while they on the other hand cover sum types which we do not.

In terms of correctness in an operational sense, a more common method appears to be to show a form of subject reduction for resources, that is, that the amount of resources used by a term is preserved by reduction. This is the case, for instance, for Orchard et al. [2019] and Moon et al. [2021]. As we have argued in Section 4.1, subject reduction alone is not sufficient for showing correctness in a quantitative sense, though it may serve as a reasonable "sanity check" as part of a larger correctness argument. For instance, Abel et al. [2023b] do not only prove subject reduction, but they also show correctness for erasure in the sense that removing erased content during compilation is safe. Atkey [2018] shows resource-correctness for QTT, a dependent type theory with linearity and erasure, by means of a logical relation between a set-theoretic model of QTT and an interpretation into linear combinatory algebras called BCI algebras. This logical relation is constructed as an instance of a generic model that uses categories with families enriched with quantitative information.

Another option is to approach the correctness argument from a non-operational angle. This is done, for instance, by Abel and Bernardy [2020] who define a relational semantics that is used to obtain "free theorems" for linear (non-dependent) function spaces, in particular that a linear function of type $A \times A \to^1 A \times A$ (polymorphic in A) must be either the identity or the swap function. This notion of correctness is different from the one that we used. While we can conclude that applications of such a linear function would lead to one lookup of the linear argument it does not follow (at least not directly) that the function is either identity or swap.

7.2 Natural Numbers and Recursion

Let us now discuss some existing methods for integrating one or more inductive data types into graded type theory. We have already discussed in some detail how the natrec-star operator due to Abel et al. [2023b] would, at least as presented, be insufficient for quantitative settings in general. However, the main correctness argument of Abel et al. concerned erasure, and we can note that for the erasure modality the usage rule we suggest here coincides with one of theirs.

While QTT[Atkey, 2018] does not include recursive data types, Idris 2 [Brady, 2021] has support for linearity and erasure, based on QTT, and allows user-defined data types. To our knowledge the exact rules for how quantities and data types interact in Idris are not documented. However, we have implemented variants of natrec_p^r corresponding to the different values of p and r for the linear types modality. Through our testing, it appears that our usage rules coincide with those used by Idris in this case.

Nakov and Forsberg [2021] extend QTT with polynomial functors, allowing the encoding of inductive data types. These data types come with eliminators that are linear in the matched argument for which usage rules can be derived. As a concrete example they consider the natural numbers, deriving a usage rule for which the successor branch may use the natural number 0 times and the recursive call 1 time. In our setting, this is the same as setting $p = 0$ and $r = 1$ (see Section 4.4). In our case we also get linear usage of the scrutinee when $p = 1$ and $r = 0$ (this case is not discussed by Nakov and Forsberg).

Atkey [2024] adds some inductive types to (a variant of) QTT in a systematic way. The focus is on polynomial-time computation, and the data types' eliminators are restricted to allow recursion only in the erased fragment. While perhaps a bit restrictive for working with recursive data types, the method used should be generally applicable, in particular showing how usage counting could be done for non-recursive types (it should also be noted that adding unrestricted support for data types was not a stated goal of this work). In addition to the restricted data types there is a type of natural numbers with support for recursion: The eliminator's successor branch only allows uses of the natural number in erased positions, while the recursive call is used once. In addition, both the zero and successor branches are restricted to have no uses of any other variables. The first restriction appeared also in the approach used by Nakov and Forsberg [2021], and the second corresponds to having $\gamma_{\mathsf{z}} = \gamma_{\mathsf{s}} = \mathbf{0}$ in our setting.

Another approach is taken by Brunel et al. [2014], who only support pattern matching on natural numbers and thus does not allow recursion directly. Recursion over natural numbers can instead be achieved by combining matching with a fixpoint combinator for which the "recursive call" is assigned a grade p. The total usage count q of the fixpoint combinator is then given as a solution to $q \leq 1 + pq$ (we have flipped the inequality operator here to be consistent with our definition of subsumption). This approach may seem rather different from ours, but we recognize this inequality (at least up to commutativity of multiplication) as the characteristic inequality we obtained from our greatest lower bounds.

Going back to QTT, a different extension is given by Huang and Yallop [2024]. They extend QTT with a type of lists together with an eliminator, and suggest that the method that is used should be extendable to arbitrary inductive families. In contrast to us, the eliminator is not annotated with grades indicating how many times resources are used by the eliminator. Instead, this information is encoded in the constructors, allowing them to contain several copies of the head and tail of the list (similarly to our Σ-types). The eliminator is then linear in the sense that it uses the supplied list once and requires that all available copies of the head and tail are used (up to subsumption).

8 Conclusions

We have investigated a graded modal language with dependent types in the style of Abel et al. [2023b], proving its usage accounting to be correct with respect to an operational semantics for a certain class of modality instantiations with quantitative interpretations. In particular, we can instantiate this result with modalities for erasure, linearity and affine types. We have also considered how one can support recursive types in this kind of system, in particular natural numbers, and how to correctly define usage counting for their eliminators. We believe that our method is applicable to a larger group of inductive types, and have sketched how it might be applied beyond natural numbers.

Our work is built on the Agda formalization by Abel et al. [2023a], which we have extended, and the formalization is available for further extensions.

Some possible directions for future work:

- One obvious idea is to try to add support for more recursive types to the formalization, perhaps a universe of types in the style of Chapman et al. [2010]. Note, however, that it should be possible to encode some recursive types using natural numbers and large elimination, for instance vectors. One could also consider how to adapt our method to support constructors that take several copies, as supported by Huang and Yallop [2024].
- Our abstract machine is rather high-level. It might be worthwhile to develop similar results for a more low-level machine that uses a more realistic evaluation strategy, and that incorporates actual erasure. It could also be interesting to make actual use of the support for linear or affine types, for instance by adding support for safe mutability in the style of Bernardy et al. [2017].

Acknowledgments

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). Oskar Eriksson additionally acknowledges support by *Knut and Alice Wallenberg Foundation* via project 2019.0116. Nils Anders Danielsson acknowledges support from Vetenskapsrådet (2023-04538).

Bibliography

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, Nils Anders Danielsson, and Oskar Eriksson. 2023a. *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 Oskar Eriksson. 2023b. A Graded Modal Dependent Type Theory with a Universe and Erasure, Formalized. *Proc. ACM Program. Lang.* 7, ICFP, Article 220 (Aug. 2023), 35 pages. https://doi.org/10.1145/3607862

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

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

Robert Atkey. 2024. Polynomial Time and Dependent Types. *Proc. ACM Program. Lang.* 8, POPL, Article 76 (Jan. 2024), 30 pages. https://doi.org/10.1145/3632918

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

Jean-Philippe Bernardy and Patrik Jansson. 2025. Domain-specific tensor languages. *Journal of Functional Programming* 35 (2025), e9. https://doi.org/10.1017/S0956796825000048

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

James Chapman, Pierre-Évariste Dagand, Conor McBride, and Peter Morris. 2010. The gentle art of levitation. In *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming* (Baltimore, Maryland, USA) *(ICFP '10)*.

Association for Computing Machinery, New York, NY, USA, 3–14. https://doi.org/10.1145/1863543.1863547

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

Oskar Eriksson, Nils Anders Danielsson, and Andreas Abel. 2025. *An Agda Formalization of Graded Modal Type Theory.* https://doi.org/10.5281/zenodo.15189791

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

Yulong Huang and Jeremy Yallop. 2024. Towards Quantitative Inductive Families. In *30th International Conference on Types for Proofs and Programs, TYPES 2024 - Abstracts.* 84–85. https://vipwww.itu.dk/research/types2024/abstracts.pdf

Anton Lorenzen, Leo White, Stephen Dolan, Richard A. Eisenberg, and Sam Lindley. 2024. Oxidizing OCaml with Modal Memory Management. *Proc. ACM Program. Lang.* 8, ICFP, Article 253 (Aug. 2024), 30 pages. https://doi.org/10.1145/3674642

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

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

Georgi Nakov and Fredrik Nordvall Forsberg. 2021. Quantitative Polynomial Functors. In *27th International Conference on Types for Proofs and Programs, TYPES 2021, June 14-18, 2021, Leiden, The Netherlands (Virtual Conference) (LIPIcs, Vol. 239)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 10:1–10:22. https://doi.org/10.4230/LIPICS.TYPES.2021.10

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

Peter Sestoft. 1997. Deriving a Lazy Abstract Machine. *Journal of Functional Programming* 7, 3 (1997), 231–264. https://doi.org/10.1017/S0956796897002712

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