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