CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

An Agda Formalisation of Modalities and Erasure in a Dependently Typed Language

Master's thesis in Computer science and engineering

OSKAR ERIKSSON

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021
www.chalmers.se
www.gu.se

Master's thesis 2021

An Agda Formalisation of Modalities and Erasure in a Dependently Typed Language

Oskar Eriksson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

An Agda Formalisation of Modalities and Erasure in a Dependently Typed Language

Oskar Eriksson

Supervisor: Andreas Abel, Department of Computer Science and Engineering
Examiner: Patrik Jansson, Department of Computer Science and Engineering

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone: +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2021

An Agda Formalisation of Modalities and Erasure in a Dependently Typed Language

Oskar Eriksson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Modal types extend the expressivity of types by giving them different interpretations depending on the used modality. In this thesis, we develop a general modality structure in the setting of a dependently typed language, notably containing dependent products and sums as well as natural numbers, based on the work of Abel and Bernardy [1]. The modality structure is based on a semiring, the elements of which are used as annotations on terms and whose algebraic properties form the basis for the modal type system.

In addition, we instantiate the general modality structure to a modality for erasure in which annotations are interpreted as either computationally relevant, indicating that the annotated term is used during evaluation, or computationally irrelevant, indicating that the marked term is not useful during evaluation. Based on this interpretation, we define an extraction function that translates terms to an untyped lambda calculus, removing terms that have been marked for erasure. Using a logical relation between terms of the two languages we then prove the extraction function to be sound with respect to the semantics of the languages in the special case of natural numbers.

Keywords: modal type theory, Martin-Löf type theory, erasure, logical relation, modalities, Agda

Acknowledgements

I would like to thank my supervisor Andreas Abel for suggesting this project and guiding me in the right direction. I also want to thank Patrik Jansson and Örjan Sunnerhagen for their helpful comments.

Oskar Eriksson, Gothenburg, June 2021

Table of Contents

1. Introduction

Modal types, like their counterparts in logic, allow modifiers to be attached to types. Given different sets of modifiers and different rules for the treatment of such modifiers by the type system, this allows types to be given a range of different interpretations. These interpretations can range from being quantitive in nature, expressing, for instance, linearity or erasure, to a variety of other interpretations such as data privacy [1]. Often, modal type systems are designed with specific interpretations in mind and the modifiers and rules are chosen based on this interpretation. An example of this is McBride's modality for erasure and linearity [2]. Others treat modal types in a more general sense and use a more general set of rules to allow different interpretations to be used in the same system.

In this thesis, we study a modal type system of the latter variety, based on the work of Abel and Bernardy [1]. The setting is a small dependently typed lambda calculus with dependent functions and pairs, natural numbers, unit and bottom types as well a single universe. Modal modifiers are taken from a ring-like structure and treated as annotations on types and terms. The algebraic rules of the ringoid form the basis of the rules of the type system. This allows different instances of the ringoid to be used to achieve different modal systems, all based on a single framework.

In addition, we give a concrete example of the use of our system by studying a modality for erasure. In particular, we use the modality system to mark terms that are not needed during runtime and remove such terms through an extraction function. Using a logical relation, we then prove the extraction to be sound for closed terms of the natural number type.

1.1 Thesis Outline

In Chapter 2, we first give a brief introduction to the concepts of Martin-Löf type theory, modal type theory and erasure. We also introduce Agda, the proof assistant used to formalise this work. We then introduce the base language of our work in Chapter 3. In Chapter 4, we introduce the concept of modalities and extend the syntax and type system of the base language with modality annotations. Chapter 5 instantiates this system to be used for erasures and defines extraction to an untyped language. We also give a short example of erasure in our language to demonstrate its use. In Chapter 6 we then introduce a logical relation which is used to prove soundness of the extraction function before we conclude with some discussion about our design choices and possible venues for future work in Chapter 7.

2. Background

In this chapter we introduce some necessary background, starting with a quick introduction to Martin-Löf type theory and modal type theory of which the studied type system is an instance. Then, we give a brief introduction to the concept of erasure before introducing the proof assistant and programing language Agda as well as the formalisation of this work.

2.1 Martin-Löf Type Theory

Martin-Löf type theory (MLTT) is the foundation on which this work rests. It serves both as the type system of the studied language $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ (which will be properly introduced in Chapter 3) and as the underlying framework used in Agda, which is here used for proof verification (see Section 2.4). MLTT builds on the simply typed lambda calculus (STLC), with some base type(s), most notably by introducing the concept of dependent types.

2.1.1 Dependent Types

The main advantage of MLTT over STLC is given by the inclusion of dependent types. For this work, the most important ones are dependent product and dependent sum types. Dependent product types (or Π-types) are generalisations of normal function types. If A and B are types and x is a free variable in B then $\prod_{a:A} B(x = a)$ is the dependent function type from A to B [3, pp. 26-29]. The dependency refers to B, the type of the co-domain, depending on a, the (value of the) function argument. Of course, if B does not depend on x the type reduces to a normal function type in which the co-domain does not change with a.

Similarly to Π-types, dependent sums (or Σ-types) are generalisations of pair types[1]. Again, if A and B are types and x is a free variable in B then $\sum_{a:A} B(x = a)$ is the dependent pair type where the first component has type A and the second type $B(x = a)$ [3, pp. 39-41]. In this case, the type of the second component of the pair depends on a, the value of the first. As for Π-types, non-dependent pairs are received when B does not depend on the value of a.

Other types which often appear in MLTT, but which we will not consider here, include the identity type, representing propositional equality, and W-types which represent general inductive structures.

[1] We refer to this as pair type instead of the more common product type to avoid confusion with dependent products.

2.1.2 Universes

To increase the expressivity of the system, it is desirable to treat types as proper terms. One consequence of this is that types are required to have types of their own in order to be well-typed. This "type of types" is typically referred to as a (Russel) universe [3, pp. 87-91]. In order to ensure consistency, the universe cannot be its own type and so one often introduces a sequence of universes, with the first being the type of all small, non-universe types and the following each being the type of the previous.

2.2 Modal Type Theory

In modal logic, modal operators are used to qualify logical formulas with different operators allowing qualified formulas to be interpreted in different ways. The archetypal examples are the modal operators for necessity and possibility, used to indicate that a formula must or might hold, but other examples are plentiful. Through the Curry-Howard correspondence, modal type theory similarly allows types to be qualified by modalities. Modalities enrich the type system through their different interpretations, allowing the programmer to express, for instance, data privacy or linearity [1] which are verified during type-checking.

A type system which allows expressing linearity is also an example of a quantitive type theory. Quantitive type theory can, at least for our purposes, be seen as a special case of modal type theory in which modalities are used to express some quantitive information. Quantitive here refers to the resources consumed by a program or how many times the resources are dereferenced at run-time. For instance, in the case of linearity, this means that the type system can guarantee that some resource (variable) is used exactly once.

2.3 Erasure

When writing programs, it is not uncommon that the programmer writes code that does not contribute to the result of the program, whether by accident or enforced by the programming language. An example of the latter kind is type information which, while vital during type-checking, is typically not necessary after compilation. In a dependently typed setting, types are not the only source of information that is irrelevant during run-time, but also, for instance, proofs. An example is a data type for vectors of length n which carry a proof of their length [4]. Since one would typically define this data type inductively, a single vector would carry proofs of the lengths of all sub-vectors as well, leading to increased memory consumption. At best, if the length can be stored in constant memory, this only changes the memory consumption by some multiplicative constant, but if, for instance, a unary representation of numbers is used for the length, the memory consumption will go from linear to quadratic in the length of the list.

Optimally, a compiler would be able to find any instance of run-time irrelevant information and remove it during compilation. This is typically referred to as *erasing* the irrelevant information. In the case of types, this is can easily be done automatically, but in general, requires either some form of static analysis [4] or help from the programmer by annotating what content should be erased [5] (which is then verified by the type-checker to be safe to erase). A combination of these two approaches is also possible where programs are partially annotated which combined with static analysis allows erasability of the whole program to be determined [6]. In either case, the use of quantitive typing can advantageously be used to keep track of erasable content since a quantity of zero indicates that a resource is never used during evaluation.

2.4 Agda

Agda is a dependently typed programming language based on Martin-Löf type theory [7]. Through the Curry-Howard correspondence, dependent products and sums can be interpreted as universal and existential quantification respectively which allows Agda to also be used as a proof assistant. Propositions are written as Agda types and proofs as programs whose validity is verified by Agda during type-checking.

The entirety of this work has been formalised and verified in Agda. In the electronic version of this thesis, definitions and theorems marked in blue, link to an HTML rendering of their formalisation. The full source code is also available online[2].

The formalisation builds on a pre-existing Agda development which formalised a study of decidability of type conversion for a small dependently typed language by Abel et al. [8]. For our purposes, the most interesting parts of this formalisation are the representation of $\lambda^{\Sigma\Pi\mathcal{U}\mathbb{N}}$, the studied language, and parts of the logical relation used in the proof of decidability. The next chapter introduces $\lambda^{\Sigma\Pi\mathcal{U}\mathbb{N}}$ properly, focusing on its syntax, types and semantics, while the key parts of the logical relation are introduced in Sections 6.1 and 6.2.

[2]https://doi.org/10.5281/zenodo.5036607

3. The Language $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}$

This chapter introduces the language $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ (pronounced "lambda-spun") which will be the base language for the modality extension in Chapter 4. The original appearance of the language was in the study by Abel et al. [8] that the Agda development is based on. We keep the description relatively short, referring to this work for further details, though it should be noted that the language, and Agda formalisation, has been extended with unit, bottom and Σ-types since the article was published. First, the abstract syntax of $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ is introduced, then via an introduction to weakenings and substitutions the typing judgements before concluding with the reduction rules defining the semantics of the language.

3.1 Syntax

The language $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ is a dependently typed lambda calculus with six type formers. These are the single universe (U), the type of natural numbers ($\mathbb{N}$), unit ($\top$) and empty ($\bot$) types as well as dependent functions and pairs (Π and Σ-types). Among the other terms are de Bruijn-style variables [9] in which variables with index i are written x_i, lambda abstractions and applications, the unit element $\star$ as well as unary natural numbers using the constructors zero and suc. The natural number and empty types come with recursors natrec and $\mathsf{emptyrec}$ whereas Σ-types come with projections fst and snd.

$$\frac{}{\mathsf{U} \in \mathsf{Term}_n} \qquad \frac{}{\mathbb{N} \in \mathsf{Term}_n} \qquad \frac{}{\top \in \mathsf{Term}_n} \qquad \frac{}{\bot \in \mathsf{Term}_n}$$

$$\frac{F \in \mathsf{Term}_n \quad G \in \mathsf{Term}_{n+1}}{\Pi\, F\, G \in \mathsf{Term}_n} \qquad \frac{F \in \mathsf{Term}_n \quad G \in \mathsf{Term}_{n+1}}{\Sigma\, F\, G \in \mathsf{Term}_n}$$

$$\frac{}{x_i \in \mathsf{Term}_n}\, i < n \qquad \frac{t \in \mathsf{Term}_{n+1}}{\lambda\, t \in \mathsf{Term}_n} \qquad \frac{t \in \mathsf{Term}_n \quad u \in \mathsf{Term}_n}{t\, u \in \mathsf{Term}_n}$$

$$\frac{t \in \mathsf{Term}_n \quad u \in \mathsf{Term}_n}{(t, u) \in \mathsf{Term}_n} \qquad \frac{t \in \mathsf{Term}_n}{\mathsf{fst}\, t \in \mathsf{Term}_n} \qquad \frac{t \in \mathsf{Term}_n}{\mathsf{snd}\, t \in \mathsf{Term}_n}$$

$$\frac{}{\mathsf{zero} \in \mathsf{Term}_n} \qquad \frac{t \in \mathsf{Term}_n}{\mathsf{suc}\, t \in \mathsf{Term}_n} \qquad \frac{}{\star \in \mathsf{Term}_n} \qquad \frac{A \in \mathsf{Term}_n \quad t \in \mathsf{Term}_n}{\mathsf{emptyrec}\, A\, t \in \mathsf{Term}_n}$$

$$\frac{A \in \mathsf{Term}_{n+1} \quad z \in \mathsf{Term}_n \quad s \in \mathsf{Term}_{n+2} \quad t \in \mathsf{Term}_n}{\mathsf{natrec}\, A\, z\, s\, t \in \mathsf{Term}_n}$$

Figure 3.1: Abstract syntax of $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}$.

The syntax of $\lambda^{\Sigma\Pi\mathcal{U}\mathbb{N}}$ is defined inductively and given in Figure 3.1; Term_n is the set of terms containing at most n free variables. Defining the syntax in this way allows the language to be well-scoped by construction, making the introduction of unexpected free variables impossible. Four terms require at least one of their subterms to introduce at least one additional variable, namely G in $\Pi\, F\, G$ and $\Sigma\, F\, G$, both A and s in $\mathsf{natrec}\, A\, z\, s\, t$, and t in $\lambda\, t$. These terms are the binders of the language and are the only places in which new variables can be introduced. This is one of the key points which enables well-scopedness.

3.1.1 Weakening

Since a term t in Term_n contains *at most* n free variables it can also be considered to be a term in Term_m if $m \geq n$. Weakenings formalises this idea by shifting de Bruijn indices upwards, essentially introducing new variables (which do not actually appear in t). Figure 3.2 defines Wk^n_m, the set of weakenings that translates terms in Term_n to terms in Term_m.

$$\frac{}{\mathsf{id} \in \mathsf{Wk}^n_n} \qquad \frac{\rho \in \mathsf{Wk}^n_m}{\uparrow\rho \in \mathsf{Wk}^n_{m+1}} \qquad \frac{\rho \in \mathsf{Wk}^n_m}{\Uparrow\rho \in \mathsf{Wk}^{n+1}_{m+1}}$$

Figure 3.2: Definition of weakenings.

The identity weakening id by itself has no effect when applied to a term but is used as the basis when forming all other weakenings. Shifting a weakening ρ, denoted $\uparrow\rho$, increases the index of all variables by one and then applies ρ. Finally, lifting a weakening, $\Uparrow\rho$ increases the index of all variables except x_0 which is left as is and then applies ρ. It is intended for binders as it leaves the newly bound variable untouched. For convenience, we define $\mathsf{wk1}\, t$ to be the application of the weakening $\uparrow\mathsf{id}$ to t, that is, shifting all variable indices in t up by one.

3.1.2 Substitution

Weakenings have the effect of replacing variables in a term with other variables. Substitutions generalise this by allowing variables to be replaced with any term (with an appropriate amount of free variables). We treat substitutions $\sigma \in \mathsf{Subst}^n_m$ as finite maps, mapping all variable indices up to some index n to terms in Term_m. Weakenings are considered a special case of substitutions that only maps variables to variables and we generalise the lift and shift operations for weakenings to substitutions. The notation $t[\sigma]$ is used to denote the application of both substitutions and weakenings to a term.

In addition to being constructed as a weakening, substitutions can be constructed by extending an existing substitution σ with a term t. Such extensions, written

σ, t, substitute all occurrences of x_0 with t and all other variables by σ with indices shifted down by one[1]. This allows substitutions to be viewed as lists, containing the terms to be substituted and we will use $\mathsf{head}\,\sigma$ and $\mathsf{tail}\,\sigma$ for the head and tail of such lists[2].

In short, the application of a substitution σ to a term t is done as follows. If $t = x_i$, $x_i[\sigma] = \sigma(x_i)$. Otherwise, the substitution is applied to each sub-term of t. Binders in sub-terms are taken care of by lifting σ the corresponding number of times before it is applied. For more details, see Abel et al. [8].

Concluding this section we now introduce shorthands for some common substitutions. First, $t[u]$ which denotes $t[\mathsf{id}, u]$, substitutes x_0 in t with u and shifts all other indices down by one. This corresponds to the substitutions which appear in the Beta rule using de Bruijn style variables [9]. The same pattern extends to several variables with $t[v, u]$, which replaces x_0 and x_1 in t with u and v respectively and shifts the remaining indices down by two, being a shorthand for $t[\mathsf{id}, v, u]$. The substitution $t[\uparrow, u]$, denoting $t[\uparrow\mathsf{id}, u]$, is similar to the single substitution above in that it also replaces x_0 with u, but does not shift the remaining indices.

3.2 Typing

This section describes the type system of $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}$. We first introduce typing contexts, used to keep track of the types of free variables, and then the five typing judgements which make up the type system.

3.2.1 Typing Context

In open terms, determining (or even defining) well-typedness is problematic since there is no way to find the types of free variables. Typing contexts solve this problem by mapping (free) variables to their type. Like terms, the set of typing contexts, Con_n, is defined in terms of a natural number which here is used to denote its length or the number of variables it is keeping track of. Contexts are defined inductively over their length as shown in Figure 3.3.

$$\frac{}{\epsilon \in \mathsf{Con}_0} \qquad \frac{\Gamma \in \mathsf{Con}_n \quad A \in \mathsf{Term}_n}{\Gamma, A \in \mathsf{Con}_{n+1}}$$

Figure 3.3: Definition of typing contexts.

Note that when extending a context with a term it can only contain at most as many

[1] That is, $(t, \sigma)x_0 = t$ and $(t, \sigma)x_{i+1} = \sigma x_i$.
[2] That is, $\mathsf{head}\,\sigma = \sigma x_0$ and $(\mathsf{tail}\,\sigma)x_i = \sigma x_{i+1}$.

free variables as can be looked up in the context. This is the second key property that ensures well-scopedness of $\lambda^{\Sigma\Pi\mathbb{U}\mathbb{N}}$.

Contexts form lists of terms in which the head corresponds to the type of the variable x_0 and the tail contains the type of the remaining $n-1$ variables. This is expressed by the relation $x_i : A \in \Gamma$, defined in Figure 3.4, which is interpreted as "Variable x_i has type A in Γ". Note the use of weakenings to account for the extended context.

$$\frac{\Gamma \in \mathsf{Con}_n \quad A \in \mathsf{Term}_n}{x_0 : (\mathsf{wk1}\, A) \in (\Gamma, A)} \qquad \frac{\Gamma \in \mathsf{Con}_n \quad A, B \in \mathsf{Term}_n \quad x_i : A \in \Gamma}{x_{i+1} : (\mathsf{wk1}\, A) \in (\Gamma, B)}\, i < n$$

Figure 3.4: Variable lookup relation for typing contexts.

3.2.2 Typing Judgements

The typing rules of $\lambda^{\Sigma\Pi\mathbb{U}\mathbb{N}}$ establish five judgements, $\vdash \Gamma$ for well-formed contexts, $\Gamma \vdash A$ and $\Gamma \vdash t : A$ for well-formed types and terms respectively as well as $\Gamma \vdash A = B$ and $\Gamma \vdash t = u : A$ for type and term conversion respectively. The definitions of these judgements are given in Figures 3.5 to 3.9. All depend on each other and should thus be considered to be defined together though they have been separated here for clarity.

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

Figure 3.5: Well-formed contexts.

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

$$\frac{\Gamma \vdash F \quad \Gamma, F \vdash G}{\Gamma \vdash \Pi\, F\, G} \qquad \frac{\Gamma \vdash F \quad \Gamma, F \vdash G}{\Gamma \vdash \Sigma\, F\, G} \qquad \frac{\Gamma \vdash A : \mathsf{U}}{\Gamma \vdash A}$$

Figure 3.6: Inference rules for well-formed types of $\lambda^{\Sigma\Pi\mathbb{U}\mathbb{N}}$.

For the most part, the rules for well-formed types and terms should not appear too surprising but the rule for natrec may need some attention. In $\mathsf{natrec}\, A\, z\, s\, n$, n is the natural number which is being recursed over and A is the resulting type of the computation, dependent on the value of n and thus binds one variable of type $\mathbb{N}$. The following term, z, represents the base case when n is zero and should thus have

$$\frac{\Gamma \vdash F : \mathsf{U} \quad \Gamma, F \vdash G : \mathsf{U}}{\Gamma \vdash \Pi\, F\, G : \mathsf{U}} \qquad \frac{\Gamma \vdash F : \mathsf{U} \quad \Gamma, F \vdash G : \mathsf{U}}{\Gamma \vdash \Sigma\, F\, G : \mathsf{U}} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \mathbb{N} : \mathsf{U}} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \top : \mathsf{U}}$$

$$\frac{\vdash \Gamma}{\Gamma \vdash \bot : \mathsf{U}} \qquad \frac{\Gamma \vdash t : A \quad \Gamma \vdash A = B}{\Gamma \vdash t : B} \qquad \frac{\vdash \Gamma \quad x_i : A \in \Gamma}{\Gamma \vdash x_i : A} \qquad \frac{\Gamma \vdash F \quad \Gamma, F \vdash t : G}{\Gamma \vdash \lambda\, t : \Pi\, F\, G}$$

$$\frac{\Gamma \vdash t : \Pi\, F\, G \quad \Gamma \vdash u : F}{\Gamma \vdash t\, u : G[u]} \qquad \frac{\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma \vdash t : F \quad \Gamma \vdash u : G[t]}{\Gamma \vdash (t, u) : \Sigma\, F\, G}$$

$$\frac{\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma \vdash t : \Sigma\, F\, G}{\Gamma \vdash \mathsf{fst}\, t : F} \qquad \frac{\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma \vdash t : \Sigma\, F\, G}{\Gamma \vdash \mathsf{snd}\, t : G[\mathsf{fst}\, t]}$$

$$\frac{\vdash \Gamma}{\Gamma \vdash \mathsf{zero} : \mathbb{N}} \qquad \frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{suc}\, n : \mathbb{N}} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \star : \top} \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash t : \bot}{\Gamma \vdash \mathsf{emptyrec}\, A\, t : A}$$

$$\frac{\Gamma, \mathbb{N} \vdash A \quad \Gamma \vdash z : A[\mathsf{zero}] \quad \Gamma, \mathbb{N}, A \vdash s : \mathsf{wk1}\,(A[\uparrow, \mathsf{suc}\, x_0]) \quad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{natrec}\, A\, z\, s\, n : A[n]}$$

Figure 3.7: Inference rules for well-formed terms of $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}$.

type A with its dependency instantiated to zero. The next term, s, comes into play in the case that n is not zero (that is, n is $\mathsf{suc}\, n'$). It binds two variables; one of type $\mathbb{N}$ and one of type A (dependent on the former). The first (which is bound to x_1) corresponds to the natural number used in the recursive call (that is, n') and the second to the recursive call itself (which is bound to x_0). The type of s should be the same as the type of natrec for this case, that is, $A[\mathsf{suc}\, n']$ or, more accurately, $A[\mathsf{suc}\, x_1]$. Because A and s are formed in different contexts, we cannot directly require s to have this type and instead additionally apply weakenings to account for the differences.

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

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

Figure 3.8: Type conversion rules of $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}$.

The rules for term and type conversion contain the usual rules of congruence, β and η-equality as well as some rules ensuring reflexivity, symmetry and transitivity. For this work, the conversion judgements have little impact so we do not discuss these further. For the most part, only the judgements for well-formed types and terms will be considered in what follows.

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

$$\frac{\Gamma \vdash t = u : A \quad \Gamma \vdash A = B}{\Gamma \vdash t = u : B} \qquad \frac{\Gamma \vdash F \quad \Gamma \vdash F = H : \mathsf{U} \quad \Gamma, F \vdash G = E : \mathsf{U}}{\Gamma \vdash \Pi\, F\, G = \Pi\, H\, E : \mathsf{U}}$$

$$\frac{\Gamma \vdash f = g : \Pi\, F\, G \quad \Gamma \vdash a = b : F}{\Gamma \vdash f\, a = g\, b : G[a]} \qquad \frac{\Gamma \vdash F \quad \Gamma \vdash F = H : \mathsf{U} \quad \Gamma, F \vdash G = E : \mathsf{U}}{\Gamma \vdash \Sigma\, F\, G = \Sigma\, H\, E : \mathsf{U}}$$

$$\frac{\Gamma \vdash F \quad \Gamma, F \vdash t : G \quad \Gamma \vdash a : F}{\Gamma \vdash (\lambda\, t)\, a = t[a] : G[a]} \qquad \frac{\Gamma \vdash F \quad \Gamma \vdash f : \Pi\, F\, G \quad \Gamma \vdash g : \Pi\, F\, G \quad \Gamma, F \vdash (\mathsf{wk1}\, f)\, x_0 = (\mathsf{wk1}\, g)\, x_0 : G}{\Gamma \vdash f = g : \Pi\, F\, G}$$

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

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

$$\frac{\Gamma \vdash F \quad \Gamma \vdash t : \Sigma\, F\, G \quad \Gamma \vdash \mathsf{fst}\, t = \mathsf{fst}\, u : F \quad \Gamma, F \vdash G \quad \Gamma \vdash u : \Sigma\, F\, G \quad \Gamma \vdash \mathsf{snd}\, t = \mathsf{snd}\, u : G[\mathsf{fst}\, t]}{\Gamma \vdash t = u : \Sigma\, F\, G} \qquad \frac{\Gamma \vdash m = n : \mathbb{N}}{\Gamma \vdash \mathsf{suc}\, m = \mathsf{suc}\, n : \mathbb{N}}$$

$$\frac{\Gamma, \mathbb{N} \vdash A \quad \Gamma, \mathbb{N} \vdash A = A' \quad \Gamma \vdash z = z' : A[\mathsf{zero}] \quad \Gamma, \mathbb{N}, A \vdash s = s' : \mathsf{wk1}\, (A[\uparrow, \mathsf{suc}\, x_0]) \quad \Gamma \vdash n = n' : \mathbb{N}}{\Gamma \vdash \mathsf{natrec}\, A\, z\, s\, n = \mathsf{natrec}\, A'\, z'\, s'\, n' : A[n]}$$

$$\frac{\Gamma, \mathbb{N} \vdash A \quad \Gamma \vdash z : A[\mathsf{zero}] \quad \Gamma, \mathbb{N}, A \vdash s : \mathsf{wk1}\, (A[\uparrow, \mathsf{suc}\, x_0])}{\Gamma \vdash \mathsf{natrec}\, A\, z\, s\, \mathsf{zero} = z : A[\mathsf{zero}]}$$

$$\frac{\Gamma, \mathbb{N} \vdash A \quad \Gamma \vdash z : A[\mathsf{zero}] \quad \Gamma, \mathbb{N}, A \vdash s : \mathsf{wk1}\, (A[\uparrow, \mathsf{suc}\, x_0]) \quad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{natrec}\, A\, z\, s\, (\mathsf{suc}\, n) = s[n, \mathsf{natrec}\, A\, z\, s\, n] : A[\mathsf{suc}\, n]}$$

$$\frac{\Gamma \vdash A = B \quad \Gamma \vdash t = u : \bot}{\Gamma \vdash \mathsf{emptyrec}\, A\, t = \mathsf{emptyrec}\, B\, u : A} \qquad \frac{\Gamma \vdash t : \top \quad \Gamma \vdash u : \top}{\Gamma \vdash t = u : \top}$$

Figure 3.9: Term conversion rules of $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}$.

3.3 Reduction

The semantics of $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ is defined from a set of call-by-name small-step reduction rules. Similarly to the typing judgements, one judgement is used for the reduction of types ($\Gamma \vdash A \longrightarrow B$) and one for terms ($\Gamma \vdash t \longrightarrow u : A$). The reduction judgements are typed, meaning that reductions are constricted by typing rules and in particular that conversion follows from reduction [8]. The rules for both relations are shown in Figure 3.10.

The customary reflexive, transitive closure of the reduction relations, for reducing terms zero or more steps, is likewise typed and defined in Figure 3.11. As shown by

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

$$\frac{\Gamma \vdash t \longrightarrow u : \Pi\, F\, G \quad \Gamma \vdash a : F}{\Gamma \vdash t\, a \longrightarrow u\, a : G[a]} \qquad \frac{\Gamma \vdash F \quad \Gamma, F \vdash t : G \quad \Gamma \vdash a : F}{\Gamma \vdash (\lambda\, t)\, a \longrightarrow t[a] : G[a]}$$

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

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

$$\frac{\Gamma, \mathbb{N} \vdash A \quad \Gamma \vdash z : A[\mathsf{zero}] \quad \Gamma, \mathbb{N}, A \vdash s : \mathsf{wk1}\, (A[\uparrow, \mathsf{suc}\, x_0]) \quad \Gamma \vdash n \longrightarrow n' : \mathbb{N}}{\Gamma \vdash \mathsf{natrec}\, A\, z\, s\, n \longrightarrow \mathsf{natrec}\, A\, z\, s\, n' : A[n]}$$

$$\frac{\Gamma, \mathbb{N} \vdash A \quad \Gamma \vdash z : A[\mathsf{zero}] \quad \Gamma, \mathbb{N}, A \vdash s : \mathsf{wk1}\, (A[\uparrow, \mathsf{suc}\, x_0])}{\Gamma \vdash \mathsf{natrec}\, A\, z\, s\, \mathsf{zero} \longrightarrow z : A[\mathsf{zero}]}$$

$$\frac{\Gamma, \mathbb{N} \vdash A \quad \Gamma \vdash z : A[\mathsf{zero}] \quad \Gamma, \mathbb{N}, A \vdash s : \mathsf{wk1}\, (A[\uparrow, \mathsf{suc}\, x_0]) \quad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{natrec}\, A\, z\, s\, (\mathsf{suc}\, n) \longrightarrow s[n, \mathsf{natrec}\, A\, z\, s\, n] : A[\mathsf{suc}\, n]}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash t \longrightarrow u : \bot}{\Gamma \vdash \mathsf{emptyrec}\, A\, t \longrightarrow \mathsf{emptyrec}\, A\, u : A}$$

Figure 3.10: Small-step reduction rules of $\lambda^{\Sigma\Pi\mathbb{U}\mathbb{N}}$.

the below theorems, these relations are deterministic and reduce terms to weak head normal form (WHNF) but not further. Terms in WHNF thus make up the values of $\lambda^{\Sigma\Pi\mathbb{U}\mathbb{N}}$. For open terms, these are any term in which a free variable prevents further reduction rules from being applied, but for closed terms, these consist of all types, zero, suc, $\star$, pairs and lambda abstractions.

Theorem 3.1. *If $\Gamma \vdash A \longrightarrow^* B$ with A in WHNF, then $A = B$. Likewise, if $\Gamma \vdash t \longrightarrow u : A$ with t in WHNF, then $t = u$.*

Proof. By induction on induction on the reduction relations. □

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \longrightarrow^* A} \qquad \frac{\Gamma \vdash A \longrightarrow A' \quad \Gamma \vdash A' \longrightarrow^* B}{\Gamma \vdash A \longrightarrow^* B}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t \longrightarrow^* t : A} \qquad \frac{\Gamma \vdash t \longrightarrow t' : A \quad \Gamma \vdash t' \longrightarrow^* u : A}{\Gamma \vdash t \longrightarrow^* u : A}$$

Figure 3.11: Reflexive, transitive closures of reduction relations of $\lambda^{\Sigma\Pi\mathbb{U}\mathbb{N}}$.

Theorem 3.2. *If* $\Gamma \vdash A \longrightarrow^* B$ *and* $\Gamma \vdash A \longrightarrow^* B'$ *with* B *and* B' *in WHNF, then* $B = B'$*. Likewise, if* $\Gamma \vdash t \longrightarrow^* u : A$ *and* $\Gamma \vdash t \longrightarrow u' : A$ *with* u *and* u' *in WHNF, then* $u = u'$*.*

Proof. By induction on the reduction relations. □

4. Modalities

In this chapter, we will extend $\lambda^{\Sigma\Pi\mathcal{U}\mathbb{N}}$ with modality annotations. We first give a general definition of modalities as a ringoid and discuss its main properties. Using this, we then introduce $\lambda^{\Sigma\Pi\mathcal{U}\mathbb{N}}_{\mathbb{M}}$, a family of languages, parameterised by the modality structure $\mathbb{M}$, which extends $\lambda^{\Sigma\Pi\mathcal{U}\mathbb{N}}$ with modality annotations. Though their use will change with the specific modality used, these languages will all share a type system built on the properties of the modality structure. One particular interpretation of these annotations, which we will use extensively, is as indicating the amount of resources needed to form terms. With this interpretation, we think of the language as using quantitive types. Although this is not the only use case for modalities, thinking of them as resources helps form intuition for their behaviour and we will often refer to them as such.

4.1 Modalities

In order to build a general theory for modal types, we need our modalities to satisfy a certain set of properties. We begin by giving the definition of modalities and proceed with explaining how the specifics of the definitions will be used afterwards.

Definition 4.1 (Modalities). A modality ringoid is a 7-tuple $(M, +, \cdot, \wedge, \mathsf{nr}_i, 0, 1)$, consisting of (from left to right) a set M, three binary operations (addition, multiplication and meet), one natural number-indexed ternary operation and zero and unit elements, satisfying the following properties:

- $(M, +, \cdot, 0, 1)$ 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 left and right distributive over addition.
- $(M, \wedge)$ forms a semilattice: Meet is commutative, associative and idempotent. The semilattice induces the usual partial ordering relation: $p \leq q$ iff $p = p \wedge q$.
- The ternary operation satisfies $\mathsf{nr}_0\, p\, q\, r = 0$ and $\mathsf{nr}_{i+1}\, p\, q\, r = p \wedge (q + r \cdot \mathsf{nr}_i\, p\, q\, r)$. Further, there exists a fixpoint i_0 such that $\mathsf{nr}_{i_0+1}\, p\, q\, r = \mathsf{nr}_{i_0}\, p\, q\, r$.
- Both multiplication and addition are left and right distributive over meet.
- The semiring is positive in the sense that $0 \leq p + q$ implies $0 \leq p$ and $0 \leq q$.

This definition mostly follows that of Abel and Bernardy [1] with the exception of the ternary operation and the positivity constraint. The latter is more reminiscent of the

structure employed by Atkey [10]. The reasons for these inclusions are, as we shall see, for the treatment of recursion (natrec in particular) and the Σ-type projections respectively. As hinted by the definition, we will often use the metavariables p, q and r to range over elements of a modality set. We will typically use the word *modalities* to refer to such elements rather than the ringoid structure. For the modality structure itself, the metavariable $\mathbb{M}$ will be used. We will see the reasons for introducing these operations as well as the properties we impose when we define the typing rules of our language, but we give an intuitive notion of how the operations will be used already.

The addition and multiplication operations are the primary ways in which modalities from several terms can be combined. Addition is used to find the resource use of a term by combining the resources of sub-terms while multiplication indicates multiple occurrences of some resource-consuming term. An example of the latter is function application in which the argument may occur several times in the function body. The zero and unit elements correspond, in this view, to no occurrences and a single occurrence respectively.

Meet can, in general, be used to match modalities between branches in conditional terms but for the most part, we will use it in terms of the partial ordering relation it induces. This relation should be interpreted as the smaller element being less specific than the larger in the same sense as for subtyping. In particular, if $p \leq q$, we will allow p to be used wherever q is expected. This view is supported by the following theorems which state that the relation is indeed a partial order, that all three binary operations are monotone with regards to it and that the meet operation is a decreasing function. The last property indicates why meet can be used to ensure compatibility between case branches.

Theorem 4.2. *$\leq$ forms a partial order, that is, it is reflexive, transitive and antisymmetric.*

Proof. Reflexivity, transitivity and antisymmetry follow from the idempotency, associativity and commutativity of meet respectively. □

Theorem 4.3. *Multiplication, addition and meet are monotone with respect to $\leq$, that is, if $p \leq q$ and $p' \leq q'$ then $pp' \leq qq'$ (and analogous for addition and meet). As usual, multiplication signs are omitted.*

Proof. The addition and multiplication cases follow from distributivity over meet. The meet case follows from the commutativity, associativity and idempotence of meet. □

Theorem 4.4. *Meet is a decreasing function, that is, $p \wedge q \leq p$ and $p \wedge q \leq q$.*

Proof. Follows from the definition of the ordering relation and the idempotency and associativity of meet. □

Finally, as already hinted at, nr_i will be used to compute the modality usage of recursive terms. More specifically, we will make use of this operator at the fixpoint and we define $p \circledast_r q \coloneqq \mathsf{nr}_{i_0}\, p\, q\, r$ as a more convenient notation for this. Because the operation provides a solution to a specific recurrence relation (see theorem 4.7) we will refer to $\circledast_r$ as the recurrence operator. As the notation suggests, we will view this as a binary operation on the first two arguments of nr, indexed by the third. This view is supported by the following distributivity properties which leave the third argument untouched.

Theorem 4.5. *Multiplication is right distributive over* $\circledast_r$, $(p \circledast_r q) \cdot p' = (p \cdot p') \circledast_r (q \cdot p')$. *Further, addition is super-distributive*[1] *over* $\circledast_r$, $(p \circledast_r q) + (p' \circledast_r q') \leq (p + p') \circledast_r (q + q')$.

Proof. The corresponding properties for nr_i are shown by induction on i. The theorem then follows by instantiating these proofs at the fixpoint. □

The definition of $\circledast_r$ in terms of a fixpoint of an iteratively defined function also provides us with the following properties:

Theorem 4.6. *The recurrence operator is monotone in both its arguments and its index. If* $p \leq p'$, $q \leq q'$ *and* $r \leq r'$, $p \circledast_r q \leq p' \circledast_{r'} q'$.

Proof. The corresponding property for nr_i follows from the monotonicity of addition, multiplication and meet. The theorem then follows by instantiating this proof at the fixpoint. □

Theorem 4.7. *The recurrence operator satisfies the recurrence relation* $p \circledast_r q = p \wedge (q + r(p \circledast_r q))$.

Proof. Follows directly from the existence of a fixpoint and the iterative property of nr_i. □

The significance of this recurrence relation is probably not immediately obvious but is, as we will see later, imperative for the proper treatment of the natural number recursor.

[1] Super and sub-distributivity are weaker notions of distributivity in which equality is replaced with $\geq$ and $\leq$ respectively.

4.1.1 Modality Contexts

For the same reasons that we use a typing context to keep track of the types of free variables, we will make use of modality contexts to keep track of the modality associated with each variable. We denote the set of modality contexts by $\mathsf{Con}_n^{\mathbb{M}}$, signalling that the context corresponds to modality structure $\mathbb{M}$, and define it similarly to typing contexts in Figure 4.1. Continuing the similarities to typing contexts, we will use γ, δ and η to denote modality contexts, that is, lower case of the letters used for typing contexts.

$$\frac{}{\epsilon \in \mathsf{Con}_0^{\mathbb{M}}} \qquad \frac{\gamma \in \mathsf{Con}_n^{\mathbb{M}} \quad p \in M}{\gamma, p \in \mathsf{Con}_{n+1}^{\mathbb{M}}}$$

Figure 4.1: Definition of modality contexts.

Addition and meet are lifted to act pointwise on contexts of equal length and scaling with modality p is similarly defined by multiplying each element in the context with p from the left. The recurrence operator is likewise lifted pointwise on its first two arguments, still being indexed by a single modality element[2]. We will also require the following partial order for modality contexts.

Definition 4.8 (Partial order of modality contexts). $\gamma \leq \delta$ iff $\gamma = \gamma \wedge \delta$.

As for the operators, this is simply the corresponding definition for modalities lifted to act pointwise on modality contexts.

For convenience, we will use $\mathbf{0}$ to indicate the zero context, where all elements are 0, and $\mathbf{e}_i$ to mean a context that is equal to $\mathbf{0}$ at all points except its ith element is unit. In Section 4.2.5 we will treat modality contexts as vectors. From that perspective, $\mathbf{e}_i$ corresponds to a vector in the standard basis. For non-empty contexts, we also introduce $\mathsf{tail}\,(\gamma, p) = \gamma$ and $\mathsf{head}\,(\gamma, p) = p$ as well as $\gamma(x_i)$ for looking up the modality associated with x_i.

We conclude this section by noting that the properties of modalities also hold for contexts.

Theorem 4.9. *The commutativity, associativity, idempotency, distributivity and positivity properties of modalities hold also for modality contexts. Further, for any modality p, and any context γ, $p\mathbf{0} = \mathbf{0}$, $1\gamma = \gamma$. Additionally, theorems 4.2 to 4.7 hold also for modality contexts with necessary changes.*

Proof. By induction on the length of contexts using the corresponding properties

[2]This is another reason for treating it as a binary operator.

for modalities. □

4.2 The Language $\lambda_{\mathbb{M}}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$

For the remainder of this chapter, we will use the above definition of modalities to define $\lambda_{\mathbb{M}}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$, starting with a modification of the syntax with annotations. We then look at the type system and adapt it to take these annotations into account and ensure their consistency. Having defined the language, we will then explore some properties of the system. In particular, we will show modality analogues to typing properties such as the substitution lemma and subject reduction.

4.2.1 Syntax

As already mentioned, the main change we will make to $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ syntax-wise when defining $\lambda_{\mathbb{M}}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ is adding annotations to some terms. Thinking of modalities as resources, these are used to indicate the amount of resources consumed by the term in some sense. As we will see later, the projections for Σ-types cannot be freely used as they would in a modality-free language. For this reason, $\lambda_{\mathbb{M}}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ also introduces $\mathsf{prodrec}_p\, A\, t\, u$, a recursor for Σ-types. The informal semantics of this term is that any occurrences of x_1 and x_0 in u will be replaced with the first and second components of the pair t respectively (as seen in Figure 4.2, u binds two variables).

Terms are as before indexed by the number of free variables in order to keep the language well-scoped, but is now also indexed by a modality ringoid $\mathbb{M}$ from which modality annotations are drawn. The syntax of $\lambda_{\mathbb{M}}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ is given in Figure 4.2; in order to highlight the differences from $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}$, only those terms which have been changed are included.

$$\frac{F \in \mathsf{Term}_n^{\mathbb{M}} \quad G \in \mathsf{Term}_{n+1}^{\mathbb{M}}}{\Pi_p^q\, F\, G \in \mathsf{Term}_n^{\mathbb{M}}}\, p, q \in M \qquad \frac{F \in \mathsf{Term}_n^{\mathbb{M}} \quad G \in \mathsf{Term}_{n+1}^{\mathbb{M}}}{\Sigma^q\, F\, G \in \mathsf{Term}_n^{\mathbb{M}}}\, q \in M$$

$$\frac{t \in \mathsf{Term}_{n+1}^{\mathbb{M}}}{\lambda^p\, t \in \mathsf{Term}_n^{\mathbb{M}}}\, p \in M \qquad \frac{t \in \mathsf{Term}_n^{\mathbb{M}} \quad u \in \mathsf{Term}_n^{\mathbb{M}}}{t\, {}^p u \in \mathsf{Term}_n^{\mathbb{M}}}\, p \in M$$

$$\frac{A \in \mathsf{Term}_{n+1}^{\mathbb{M}} \quad z \in \mathsf{Term}_n^{\mathbb{M}} \quad s \in \mathsf{Term}_{n+2}^{\mathbb{M}} \quad t \in \mathsf{Term}_n^{\mathbb{M}}}{\mathsf{natrec}_p^r\, A\, z\, s\, t \in \mathsf{Term}_n^{\mathbb{M}}}\, p, r \in M$$

$$\frac{t \in \mathsf{Term}_n^{\mathbb{M}} \quad u \in \mathsf{Term}_{n+2}^{\mathbb{M}}}{\mathsf{prodrec}_p\, t\, u \ \in \mathsf{Term}_n^{\mathbb{M}}}\, p \in M \qquad \frac{A \in \mathsf{Term}_n^{\mathbb{M}} \quad t \in \mathsf{Term}_n^{\mathbb{M}}}{\mathsf{emptyrec}_p\, A\, t \in \mathsf{Term}_n^{\mathbb{M}}}\, p \in M$$

Figure 4.2: Abstract syntax of $\lambda_{\mathbb{M}}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ (subset).

The Π-type takes two annotations. The first (p) signifies the modality associated

with the function argument. In terms of resources, this is the number of times the function argument is used to produce the function result. The second annotation (q) instead indicates the resources needed to construct the type G. For a non-dependent function, this annotation would be 0. The annotation for Σ-types shares the same purpose. Only one annotation is needed because, in contrast to functions, pairs do not introduce new variables.

Lambda abstractions and applications both come with one annotation each, indicating the resources used by the function argument and the number of times the argument is consumed respectively. We will see that for well-typed terms these are required to both match each other and the corresponding annotation in the Π-type.

The recursors all take one or two modalities. In natrec, p signifies how many times s uses its first argument (n) and r how many times it uses the second (the recursive call to natrec). The product recursor similarly uses its annotation to indicate how many times the components of the product are used in u but uses the same annotation for both components. We will see the reason for this later. The empty type recursor also uses its annotation to indicate the number of times its argument is used. However, unlike the other recursors, the argument is not consumed to produce a result and the annotation is thus allowed to be set to any value, essentially allowing the term to be formed with any arbitrary amount of resources. The motivation for allowing this is that the occurrences of $\mathsf{emptyrec}$ already implies a contradictory context.

4.2.2 Typing

The type system of $\lambda_{\mathbb{M}}^{\Sigma\Pi\mathcal{U}\mathbb{N}}$ is divided into two main parts. The judgements for well-formed types and terms (which are almost the same as those of $\lambda^{\Sigma\Pi\mathcal{U}\mathbb{N}}$) and the judgement for well-usage of modalities which will be introduced in the next section.

In practice, the addition of modalities has little effect on the concept of well-formedness as most work is handled by the well-usage judgement. The only added complication is that the typing rules for lambda abstractions and applications require their corresponding modalities to match that of the Π-type. For clarity, Figure 4.3 show all rules for well-formedness of types and terms which involve terms with modality annotations. We omit the rules for conversion except for those corresponding to the newly added $\mathsf{prodrec}$-term which we show in Figure 4.4.

Apart from the already mentioned annotation matching, the main things of interest here are the new rules corresponding to the $\mathsf{prodrec}$-term. Similarly to natrec, in $\mathsf{prodrec}_p\, A\, t\, u$, A is the type dependent on the value of the pair t (which should have a Σ-type). The term u binds two variables which correspond to the first and second components of the pair respectively. The type of u is $A[\uparrow^2 \mathsf{id}, (x_1, x_0)]$ which is A with x_0 substituted with the pair (x_1, x_0) and all remaining variables shifted up by one. The reason for this shift is to account for the different contexts in which A and u are formed.

$$\frac{\Gamma \vdash F \quad \Gamma, F \vdash G}{\Gamma \vdash \Pi^q_p\, F\, G : \mathsf{U}} \qquad \frac{\Gamma \vdash F \quad \Gamma, F \vdash G}{\Gamma \vdash \Sigma^q\, F\, G : \mathsf{U}} \qquad \frac{\Gamma \vdash F \quad \Gamma, F \vdash t : G}{\Gamma \vdash \lambda^p\, t : \Pi^q_p\, F\, G}$$

$$\frac{\Gamma \vdash t : \Pi^q_p\, F\, G \quad \Gamma \vdash u : F}{\Gamma \vdash t\,{}^p u : G[u]} \qquad \frac{\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma \vdash t : F \quad \Gamma \vdash u : G[t]}{\Gamma \vdash (t, u) : \Sigma^q\, F\, G}$$

$$\frac{\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma \vdash t : \Sigma^q\, F\, G}{\Gamma \vdash \mathsf{fst}\, t : F} \qquad \frac{\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma \vdash t : \Sigma^q\, F\, G}{\Gamma \vdash \mathsf{snd}\, t : G[\mathsf{fst}\, t]}$$

$$\frac{\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma, \Sigma^q\, F\, G \vdash A \quad \Gamma \vdash t : \Sigma^q\, F\, G \quad \Gamma, F, G \vdash u : A[\uparrow^2 \mathsf{id}, (x_1, x_0)]}{\Gamma \vdash \mathsf{prodrec}_p\, A\, t\, u : A[t]}$$

$$\frac{\Gamma, \mathbb{N} \vdash A \quad \Gamma \vdash z : A[\mathsf{zero}] \quad \Gamma, \mathbb{N}, A \vdash s : \mathsf{wk1}\,(A[\uparrow, \mathsf{suc}\, x_0]) \quad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{natrec}^r_p\, A\, z\, s\, n : A[n]}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash t : \bot}{\Gamma \vdash \mathsf{emptyrec}_p\, A\, t : A} \qquad \frac{\Gamma \vdash F \quad \Gamma, F \vdash G}{\Gamma \vdash \Pi^q_p\, F\, G} \qquad \frac{\Gamma \vdash F \quad \Gamma, F \vdash G}{\Gamma \vdash \Sigma^q\, F\, G}$$

Figure 4.3: Inference rules for well-formed terms and types of $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}_{\mathbb{M}}$ (excerpt).

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

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

Figure 4.4: Conversion inference rules for prodrec.

4.2.3 Modality Usage

The judgements for well-formed types and terms which we considered in the last section are not enough to ensure that modality annotations are correct. For this, we define the relation $\gamma \triangleright t$ in Figure 4.5 and interpret it to mean that context γ contains enough resources to form term t. In particular, thinking of contexts as keeping track of the number of variable occurrences, free variables of t cannot occur more times than specified by γ. The definition is in large parts based on the typing relation used by Abel for resource typing in a smaller dependently typed language [11].

In most cases, the reasons behind these rules are quite straightforward. Constant terms, such as U and zero never have any variable occurrences and so we require nothing of the variables which is represented by the zero-modality context. A single variable amounts to unit usage for that variable and no usage for all other variables.

$$\frac{}{\mathbf{0} \triangleright \mathsf{U}} \qquad \frac{}{\mathbf{0} \triangleright \mathbb{N}} \qquad \frac{}{\mathbf{0} \triangleright \top} \qquad \frac{}{\mathbf{0} \triangleright \bot} \qquad \frac{\gamma \triangleright F \quad \delta, q \triangleright G}{\gamma + \delta \triangleright \Pi^q_p \, F \, G}$$

$$\frac{\gamma \triangleright F \quad \delta, q \triangleright G}{\gamma + \delta \triangleright \Sigma^q \, F \, G} \qquad \frac{}{\mathbf{e}_i \triangleright x_i} \qquad \frac{\gamma, p \triangleright t}{\gamma \triangleright \lambda^p \, t} \qquad \frac{\gamma \triangleright t \quad \delta \triangleright u}{\gamma + p\delta \triangleright t \, {}^p u}$$

$$\frac{\gamma \triangleright t \quad \delta \triangleright u}{\gamma + \delta \triangleright (t, u)} \qquad \frac{\mathbf{0} \triangleright t}{\mathbf{0} \triangleright \mathsf{fst} \, t} \qquad \frac{\mathbf{0} \triangleright t}{\mathbf{0} \triangleright \mathsf{snd} \, t} \qquad \frac{\gamma \triangleright t \quad \delta, p, p \triangleright u}{p\gamma + \delta \triangleright \mathsf{prodrec}_p \, A \, t \, u}$$

$$\frac{}{\mathbf{0} \triangleright \mathsf{zero}} \qquad \frac{\gamma \triangleright t}{\gamma \triangleright \mathsf{suc} \, t} \qquad \frac{\gamma \triangleright z \quad \delta, p, r \triangleright s \quad \eta \triangleright n}{(\gamma \wedge \eta) \circledast_r (\delta + p\eta) \triangleright \mathsf{natrec}^r_p \, A \, z \, s \, n}$$

$$\frac{\gamma \triangleright t}{p\gamma \triangleright \mathsf{emptyrec}_p \, A \, t} \qquad \frac{}{\mathbf{0} \triangleright \star} \qquad \frac{\gamma \triangleright t}{\delta \triangleright t} \, \delta \leq \gamma$$

Figure 4.5: Definition of the usage relation.

Terms which are simple collections of sub-terms, such as $\Pi^q_p \, F \, G$ and $\mathsf{suc} \, t$ (which "collects" only one sub-term) require the combined usage of the sub-terms which is received by adding their respective contexts. Whenever these sub-terms bind new variables and there are corresponding modality annotations, we ensure that the newly bound variable is given exactly the amount of resources as annotated. See, for instance, the lambda rule which requires x_0 in t to use p resources. We also have, as previously mentioned, a subsumption rule which allows the use of less specific modality contexts.

The more complicated cases are application, the recursors and the projections. The application rule states that δ resources are needed to construct u. Since $t \, {}^p u$ indicates that u will be substituted p times into t, the needed resources will be p times larger in addition to the resources needed to form t. The product recursor works similarly by multiplying the resources needed to form t by the number of occurrences in u. The reason for assuming that both components have the same number of occurrences in u is that we cannot infer the resources needed to form either of the two components by themself. We defer the discussion on the projections and natural number recursor to Section 4.2.7 after proving the subject reduction theorem which will simplify the explanation.

In the following, we will make use of the following two properties of the usage relation. The first is the customary inversion lemma.

Theorem 4.10 (Inversion lemma). *If $\gamma \triangleright t$, then:*

- *If $t = \mathsf{U}, \mathbb{N}, \top, \bot, \mathsf{zero}$ or $\star$, then $\gamma \leq \mathbf{0}$.*
- *If $t = x_i$, then $\gamma \leq \mathbf{e}_i$.*

- *If* $t = \Pi_p^q\, F\, G$ *or* $\Sigma^q\, F\, G$, *then* $\exists \delta, \eta.\ \delta \triangleright F$ *and* $\eta, q \triangleright G$ *and* $\gamma \leq \delta + \eta$.
- *If* $t = \lambda^p\, t'$, *then* $\exists \delta.\ \delta, p \triangleright t'$ *and* $\gamma \leq \delta$.
- *If* $t = t'\ ^p u$, *then* $\exists \delta, \eta.\ \delta \triangleright t'$ *and* $\eta \triangleright u$ *and* $\gamma \leq \delta + p\eta$.
- *If* $t = (t', u)$, *then* $\exists \delta, \eta.\ \delta \triangleright t'$ *and* $\eta \triangleright u$ *and* $\gamma \leq \delta + \eta$.
- *If* $t = \mathsf{fst}\, u$ *or* $\mathsf{snd}\, u$, *then* $\mathbf{0} \triangleright u$ *and* $\gamma \leq \mathbf{0}$.
- *If* $t = \mathsf{prodrec}_p\, A\, t'\, u$, *then* $\exists \delta, \eta.\ \delta \triangleright t'$ *and* $\eta, p, p \triangleright u$ *and* $\gamma \leq p\delta + \eta$.
- *If* $t = \mathsf{suc}\, u$, *then* $\exists \delta.\ \delta \triangleright u$ *and* $\gamma \leq \delta$.
- *If* $t = \mathsf{emptyrec}_p\, A\, u$, *then* $\exists \delta.\ \delta \triangleright u$ *and* $\gamma \leq p\delta$.
- *If* $t = \mathsf{natrec}_p^r\, A\, z\, s\, n$, *then* $\exists \delta, \eta, \theta.\ \delta \triangleright z$ *and* $\eta, p, r \triangleright s$ *and* $\theta \triangleright n$ *and* $\gamma \leq (\delta \wedge \theta) \circledast_r (\eta + p\theta)$.

Proof. For each term, by induction on the derivation of the appropriate cases of the usage relation. For each term, there are two cases, the term's corresponding usage rule and the subsumption rule. □

The second property involves the point-wise nature of the contexts and allows the contents of two valid contexts to be freely interchanged.

Theorem 4.11. *If* $\gamma \triangleright t$ *and* $\delta \triangleright t$, *then for any* i, $\gamma_i' \triangleright t$ *where* $\gamma_i'(x_j) = \gamma(x_j)$ *if* $j \neq i$ *and* $\gamma_i'(x_i) = \delta(x_i)$.

Proof. By mutual induction on $\gamma \triangleright t$ and $\delta \triangleright t$. □

A preferable alternative to this theorem would be $\gamma \vee \delta \triangleright t$ where $\vee$ denotes the pointwise supremum (dual to meet). Since we do not have this operation available and it is not necessarily definable for a general modality[3], the above theorem has to suffice. Another alternative to this property will be provided by theorem 4.14 which allows us to calculate an upper bound for γ and δ (though not necessarily the supremum).

4.2.4 Usage Inference

Assuming that annotations are correct, all information about what resources are needed to construct the term can be inferred just by looking at the annotations. We define the following function for this purpose:

[3]For instance, the supremum is not definable for a linearity modality.

Definition 4.12 (Modality inference). The function $|\_| : \mathsf{Term}_n^{\mathbb{M}} \to \mathsf{Con}_n^{\mathbb{M}}$, inferring a modality context of a term is defined recursively over terms as follows:

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

This definition is mostly straightforward and the similarities with the usage relation itself should be obvious. It should be noted that the inference function makes no distinction between well-resourced and non-well-resourced terms and thus always computes some context whether a valid one exists or not. As the following theorem states, the computed context is indeed valid for all well-typed and well-resourced terms.

Theorem 4.13. *If $\Gamma \vdash t : A$ and $\gamma \triangleright t$, then $|t| \triangleright t$.*

Proof. By strutural induction on the typing derivation, using the inversion lemma and theorem 4.11. □

Additionally, the computed context is an upper bound on valid contexts.

Theorem 4.14. *If $\gamma \triangleright t$ then $\gamma \leq |t|$.*

Proof. By structural induction on the derivation of $\gamma \triangleright t$. □

Taken together, theorems 4.13 and 4.14 state that for any term which is well-typed and well-resourced, the inferred modality context is not only valid, it is also the largest (most specific) context which is valid. This means that, in some sense, $|t|$ is the best modality context of t. Looking at the similarities between the inference function and the inversion lemma this result should come as no surprise as the upper bounds in the inversion lemma directly correspond to the inferred contexts.

4.2.5 Substitution

We have already touched very briefly on the consequences of applying a substitution in terms of resource usage when motivating the usage rule for applications. Then,

in the case of substituting a single variable with a term, we claimed that the added resources should be those needed to construct the term, multiplied by the modality corresponding to the substituted variable. We will now build on this intuition to see what happens for a general substitution, building up to a modality analogue of the substitution lemma for typing contexts.

When we apply a substitution σ to a term t, all occurrences of variable x_i are replaced with the term σx_i. Given that σx_i requires δ_i resources to be formed (that is, $\delta \triangleright \sigma x_i$), each occurrence of x_i will add δ_i resources. If we also have $\gamma \triangleright t$, the total amount of resources added by substituting x_i is $\gamma(x_i) \cdot \delta_i$. The combined effect of all variable substitutions is thus that the resources associated with x_j in $t[\sigma]$ are $\sum_i \gamma(x_i)\delta_i(x_j)$. This is simply the multiplication of γ with a matrix Ψ in which the ith column is δ_i.

Following the work of Wood and Atkey [12], we thus think of modality substitutions as matrices and in this regard consider modality contexts to be vectors. More concretely, a substitution $\sigma \in \mathsf{Subst}_m^n$ is represented by $\Psi \in M^{n \times m}$, that is, an $n \times m$-matrix with elements from M. The identity substitution naturally corresponds to the identity matrix with 1 on the diagonal and 0 elsewhere. For the other weakenings, shifting introduces a variable with no occurrences so a row of 0 is prepended while lifting prepends both a row and a column of zeros with 1 in the top left corner, see Figure 4.6.

As another example, for a single substitution (the substitution performed by $t[u]$) we simply add the resources needed to construct u to the left of the identity matrix[4] as in Figure 4.6. It is not difficult to see that applying this matrix to γ, p will result in $\gamma + p\delta$, the same expression that is used in the usage rule for applications.

$$\begin{bmatrix} 0 & \cdots & 0 \\ \Psi_{00} & \cdots & \Psi_{0(m-1)} \\ \vdots & \ddots & \vdots \\ \Psi_{(n-1)0} & \cdots & \Psi_{(n-1)(m-1)} \end{bmatrix} \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & \Psi_{00} & \cdots & \Psi_{0(m-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \Psi_{(n-1)0} & \cdots & \Psi_{(n-1)(m-1)} \end{bmatrix} \begin{bmatrix} \delta(x_0) & 1 & 0 & \cdots & 0 \\ \delta(x_1) & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \delta(x_n) & 0 & 0 & \cdots & 1 \end{bmatrix}$$

Figure 4.6: Substitution matrices corresponding to, from left to right, shifting a substitution matrix Ψ, lifting a substitution matrix Ψ and the single substitution $t[u]$ with $\delta \triangleright u$.

For the most part, matrix multiplication behaves as one would expect in this setting and substitution matrices capture most properties of the modality operators with some minor changes.

Theorem 4.15 (Properties of substitution matrices). *Substitution matrix multiplication satisfies the following properties:*

[4] And for several simultaneous substitutions, such as the one performed by $t[v, u]$, one simply extends the identity matrix further.

- *The identity matrix is a left identity to matrix multiplication.*
- *The zero-context is a right zero (modulo context length),* $\Psi\mathbf{0} = \mathbf{0}$.
- *Matrix multiplication is associative,* $\Phi(\Psi\gamma) = (\Phi\Psi)\gamma$.
- *Matrix multiplication distributes over addition and scaling,* $\Psi(p\gamma + q\delta) = p \cdot \Psi\gamma + q \cdot \Psi\delta$.
- *Matrix multiplication sub-distributes over meet and the recurrence operator,* $\Psi(\gamma \wedge \delta) \leq \Psi\gamma \wedge \Psi\delta$ *and* $\Psi(\gamma \circledast_r \delta) \leq (\Psi\gamma) \circledast_r (\Psi\delta)$.
- *Matrix multiplication is monotone, if* $\gamma \leq \delta$ *then* $\Psi\gamma \leq \Psi\delta$.

Proof. By induction on the length of contexts using corresponding properties for modalities and modality contexts. □

Before we can formulate our substitution lemma we need to introduce the concept of well-resourced substitutions.

Definition 4.16 (Well-resourced substitution). A substitution $\sigma \in \mathsf{Subst}^n_m$ is well-resourced in substitution matrix $\Psi \in M^{n\times m}$, written $\Psi \blacktriangleright \sigma$ iff $\forall i < n.\ \Psi\mathbf{e}_i \triangleright \sigma x_i$.

Recall that, in a linear algebra setting, $\mathbf{e}_i$ is a basis vector in the standard basis and has the effect of picking out the ith column from the matrix. The definition thus states that well-resourcedness requires each column vector of Ψ to be a valid modality context to the corresponding term in σ which is consistent with our reasoning above.

We would, like to solidify our examples from earlier by showing that the constructions yield valid matrices, but first, we need to take a quick detour from the substitution matrices and introduce the following lemma.

Theorem 4.17. *If* $\gamma \triangleright t$ *then* $\gamma' \triangleright t[\Uparrow^n (\uparrow \mathsf{id})]$ *where* $\gamma'(x_i) = \gamma(x_i)$ *for* $i < n$, $\gamma'(x_n) = 0$ *and* $\gamma'(x_i) = \gamma(x_{i+1})$ *for* $i > n$.

Proof. By structural induction on t. □

Put more simply, applying a single shift weakening lifted n times has the effect of inserting a 0 at index n in the modality context. The reason for this seemingly odd theorem is to take care of the liftings that are done when substitutions are pushed down into the binders. This property will follow from the substitution lemma, but that will require a proof that the weakening matrix constructions above are valid which in turn rests on this property.

Theorem 4.18. *The matrix constructions discussed above, in particular, those of Figure 4.6 are well-formed with respect to their corresponding substitutions.*

Proof. The five constructions are shifted and lifted substitutions, the identity substitution and single and multiple-term substitutions.

- The shifting case uses theorem 4.17 with $n = 0$.
- Lifting has two cases, $\Psi\mathbf{e}_0 \triangleright \sigma x_0$ follows from the linearity properties of matrix multiplication and the properties of the modality ringoid. The case $\Psi\mathbf{e}_{i+1} \triangleright \sigma x_{i+1}$ additionally uses the well-formedness of the shifting matrix.
- The single substitution $t[u]$ similarly has two cases, $\Psi\mathbf{e}_0 \triangleright \sigma x_0$ follows directly from $\delta \triangleright u$ and the remaining cases from the identity matrix being the left identity of matrix multiplication.
- The multiple substitution case follows a similar structure as the single substitution case.

The dependency of the lifting and identity cases requiring the shifting case should be seen as an artefact of the Agda implementation using the shifting operation in the construction of lifted matrices and the identity matrix rather than an inherent dependency. □

We can now finally conclude this section with the substitution lemma.

Theorem 4.19 (Substitution lemma). *For any term $t \in \mathsf{Term}_n^{\mathbb{M}}$, modality context $\gamma \in \mathsf{Con}_n^{\mathbb{M}}$ and substitutions $\sigma \in \mathsf{Subst}_m^n$ and $\Psi \in M^{n\times m}$, if $\Psi \blacktriangleright \sigma$ and $\gamma \triangleright t$ then $\Psi\gamma \triangleright t[\sigma]$.*

Proof. By structural induction on the derivation of $\gamma \triangleright t$ using theorem 4.18 for lifted substitutions. □

4.2.6 Substitution Inference

Using our earlier method of inferring valid modality contexts, it is, also possible to infer a valid substitution matrix to a given substitution.

Definition 4.20 (Substitution matrix inference). The substitution matrix inference function, $\|\_\| : \mathsf{Subst}_m^n \to M^{n\times m}$, is defined such that $\|\sigma\|$ is an $n \times m$-matrix where the ith column is given by $|\sigma x_i|$.

As one would expect, the inferred matrix of any substitution in which every substituted term is well-typed and well-resourced will be a valid substitution matrix.

Theorem 4.21. *If* $\forall x_i.\ \exists A, \gamma.\ \Gamma \vdash \sigma x_i : A \wedge \gamma \triangleright \sigma x_i$ *then* $\|\sigma\| \blacktriangleright \sigma$.

Proof. By theorem 4.13 applied to each column of $\|\sigma\|$. □

4.2.7 Reduction

At this point, we are almost ready to provide the final sanity-check on the proposed theory, namely that modalities are preserved under reduction. This is the modality analogue to the type preservation or subject reduction theorem.

First, though, we need to introduce the reduction rules. As with the typing rules, the modality extension does not have much effect on the reduction rules. The only real consideration is ensuring that annotations match in the same way as was done in the rules for well-formed terms. Still, Figure 4.7 shows all reduction rules in which modality annotations are involved.

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

$$\frac{\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma \vdash t \longrightarrow u : \Sigma^q\, F\, G}{\Gamma \vdash \mathsf{fst}\, t \longrightarrow \mathsf{fst}\, u : F} \qquad \frac{\Gamma \vdash F \quad \Gamma, F \vdash G \quad \Gamma \vdash t \longrightarrow u : \Sigma^q\, F\, G}{\Gamma \vdash \mathsf{snd}\, t \longrightarrow \mathsf{snd}\, u : G[\mathsf{fst}\, t]}$$

$$\frac{\Gamma, \mathbb{N} \vdash A \quad \Gamma \vdash z : A[\mathsf{zero}] \quad \Gamma, \mathbb{N}, A \vdash s : \mathsf{wk1}\,(A[\uparrow, \mathsf{suc}\, x_0]) \quad \Gamma \vdash n \longrightarrow n' : \mathbb{N}}{\Gamma \vdash \mathsf{natrec}^r_p\, A\, z\, s\, n \longrightarrow \mathsf{natrec}^r_p\, A\, z\, s\, n' : A[n]}$$

$$\frac{\Gamma, \mathbb{N} \vdash A \quad \Gamma \vdash z : A[\mathsf{zero}] \quad \Gamma, \mathbb{N}, A \vdash s : \mathsf{wk1}\,(A[\uparrow, \mathsf{suc}\, x_0])}{\Gamma \vdash \mathsf{natrec}^r_p\, A\, z\, s\, \mathsf{zero} \longrightarrow z : A[\mathsf{zero}]}$$

$$\frac{\Gamma, \mathbb{N} \vdash A \quad \Gamma \vdash z : A[\mathsf{zero}] \quad \Gamma, \mathbb{N}, A \vdash s : \mathsf{wk1}\,(A[\uparrow, \mathsf{suc}\, x_0]) \quad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{natrec}^r_p\, A\, z\, s\, (\mathsf{suc}\, n) \longrightarrow s[n, \mathsf{natrec}^r_p\, A\, z\, s\, n] : A[\mathsf{suc}\, n]}$$

$$\frac{\begin{array}{ccc}\Gamma \vdash F & \Gamma, F \vdash G & \Gamma, \Sigma^q\, F\, G \vdash A \\ \Gamma, F, G \vdash u : A[\uparrow^2 \mathsf{id}, (x_1, x_0)] & \multicolumn{2}{c}{\Gamma \vdash t \longrightarrow t' : \Sigma^q\, F\, G}\end{array}}{\Gamma \vdash \mathsf{prodrec}_p\, A\, t\, u \longrightarrow \mathsf{prodrec}_p\, A\, t'\, u : A[t]}$$

$$\frac{\begin{array}{ccc}\Gamma \vdash F & \Gamma, F \vdash G & \Gamma, \Sigma^q\, F\, G \vdash A \\ \Gamma \vdash t : F & \Gamma \vdash t' : G[t] & \Gamma, F, G \vdash u : A[\uparrow^2 \mathsf{id}, (x_1, x_0)]\end{array}}{\Gamma \vdash \mathsf{prodrec}_p\, A\, (t, t')\, u \longrightarrow u[t, t'] : A[(t, t')]}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash t \longrightarrow u : \bot}{\Gamma \vdash \mathsf{emptyrec}_p\, A\, t \longrightarrow \mathsf{emptyrec}_p\, A\, u : A}$$

Figure 4.7: Reduction rules of $\lambda^{\Sigma\Pi\mathcal{U}\mathbb{N}}_{\mathbb{M}}$ (excerpt).

Now, as promised, we have the modality preservation theorems.

Theorem 4.22 (Subject reduction). *If $\gamma \triangleright t$ and $\Gamma \vdash t \longrightarrow u : A$ then $\gamma \triangleright u$. Further, if $\delta \triangleright A$ and $\Gamma \vdash A \longrightarrow B$ then $\delta \triangleright B$.*

Proof. By structural induction on the reduction using the inversion lemma for modalities, the substitution lemma and theorem 4.18. □

We conclude this chapter by returning to the discussion of the usage rules for the projections and the natural number recursor. For further discussion on alternative approaches, see Section 7.1.

For some modalities, there is a general problem with freely allowing projections to be used because they drop resources. This becomes a problem in settings where an exact number of resources is required. For instance, a pair in a linear setting requires both components to be used exactly once, and allowing projections to be used freely makes this impossible to satisfy.

Another way of looking at this problem is by considering the subject reduction theorem. For this to hold in the case $\mathsf{fst}\,(t, u) \to t$ we require that t is formed in a context that is equal to (or, by subsumption, larger than) the context forming $\mathsf{fst}\,(t, u)$. If $\gamma \triangleright t$ and $\delta \triangleright u$ we would thus need $\gamma + \delta \leq \gamma$ if no restriction was imposed on the usage rule for the projections[5]. This is problematic whenever $\delta \leq \mathbf{0}$ does not hold, but since only the combined usage of both components is available, ensuring this with a side condition is not possible. Thus, we instead impose the restriction that the pair is formed under the zero context which together with the positivity of the modality ringoid ensures that the projections are well-behaved.

For the natural number recursor, there are two reduction cases of interest. For $\mathsf{natrec}_p^r\, A\, z\, s\, \mathsf{zero} \to z$ we need the context forming natrec (say, χ) to be at least as large as the context forming z. In the successor case, we need χ to be at least as large as the context forming $s[n, \mathsf{natrec}_p^r\, A\, z\, s\, n]$. If $\gamma \triangleright z$, $\delta, p, r \triangleright s$ and $\eta \triangleright \mathsf{suc}\, n$ we thus need (using the substitution lemma) $\chi \leq \gamma$ and $\chi \leq \delta + p\eta + r\chi$. Although satisfying these two constraints allow us to show subject reduction, it does not take the resources required by the natural number into account (only those corresponding to s using n). For this, we need to ensure that at least η resources are available thus additionally requiring $\chi \leq \eta$. These inequalities are the reason for including the recurrence operator in our modality definition as the constraints can be satisfied by letting $\chi = (\gamma \wedge \eta) \circledast_r (\delta + p\eta)$.

Theorem 4.23. *The recurrence operator satisfies the necessary inequalities. That is, $(\gamma \wedge \eta) \circledast_r (\delta + p\eta) \leq \gamma$ and $(\gamma \wedge \eta) \circledast_r (\delta + p\eta) \leq \delta + p\eta + r((\gamma \wedge \eta) \circledast_r (\delta + p\eta))$ and $(\gamma \wedge \eta) \circledast_r (\delta + p\eta) \leq \eta$.*

Proof. Using theorems 4.4 and 4.7. □

[5] That is, the rule follows the style of e.g. suc.

Additionally, the recurrence operator gives the greatest (least specific) solution of the inequalities for any modality instance where 0 is the greatest element.

Theorem 4.24. *If* $\chi \leq \gamma$ *and* $\chi \leq \delta + p\eta + r\chi$ *and* $\chi \leq \eta$ *and* $\forall q.\ q \leq 0$ *then* $\chi \leq (\gamma \wedge \eta) \circledast_r (\delta + p\eta)$.

Proof. By induction on the contexts using the recurrence property and existence of fixpoint of $\mathsf{nr}_i\, p\, q\, r$. □

5. Erasure

We will now turn our attention to a concrete application of the theory introduced in the previous chapter, namely that of erasure. We will first define a modality for erasure that allows the programmer to annotate computationally irrelevant parts of the program and the type system to track what parts of a program are erasable. We will then introduce a small untyped lambda calculus which will serve as the target language of an extraction function which has the purpose of erasing erasable content.

5.1 A Modality for Erasure

To track erasability, the type system only needs to discern between two things: whether a variable is computationally relevant or not. The modality for erasure thus contains two annotations, each representing one of these two cases. From a quantitive standpoint, this corresponds to no usage and any usage respectively which inspires the following definition of a modality for erasure.

Definition 5.1 (Erasure modality). The set $0\omega \coloneqq \{0, \omega\}$ with binary operators for addition $(+)$, multiplication $(\cdot)$ and meet $(\wedge)$ defined by Table 5.1 and nr defined by $\mathsf{nr}_0\, p\, q\, r = 0$, $\mathsf{nr}_{i+1}\, p\, q\, r = p \wedge (q + r \cdot \mathsf{nr}_i\, p\, q\, r)$ constitutes the erasure modality annotations.

Table 5.1: Definition of addition, multiplication and meet for the erasure modality as well as the recurrence operator $(p \circledast_r q)$, independent of the value of r.

$(+)$	0	ω
0	0	ω
ω	ω	ω

$(\cdot)$	0	ω
0	0	0
ω	0	ω

$(\wedge)$	0	ω
0	0	ω
ω	ω	ω

$(\circledast_r)$	0	ω
0	0	ω
ω	ω	ω

The ω-annotation signifies computationally relevant resources and 0, computationally irrelevant resources, indicating that these can be erased. The partial order for 0ω is the reflexive closure of $\omega \leq 0$. Through subsumption, this allows ω to be used to annotate any (even zero) usage, thus allowing erasable content to be marked as computationally relevant if desired. On the other hand, 0 can only be used where no variable usage occurs in a non-erased position, thus ensuring safe erasability (see theorem 5.3).

Before we are able to use these annotations in $\lambda_{\mathbb{M}}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$, we first need to show that they satisfy definition 4.1 which is the result of the following theorem.

Theorem 5.2 (Erasure modality). *The set of erasure annotations, 0ω, with corresponding operators as defined by definition 5.1 with 0 as zero and ω as unit forms a modality ringoid.*

Proof. All properties are easily shown by case distinction on 0 and ω. For $\mathsf{nr}_n\, p\, q\, r$, the fixpoint is reached after one iteration and $p \circledast_r q$ turns out to be independent of r and coincides with addition and meet, see Table 5.1. □

Having shown that 0ω forms a modality, we can instantiate $\lambda_{0\omega}^{\Sigma\Pi\mathbb{U}\mathbb{N}}$, an extension of $\lambda^{\Sigma\Pi\mathbb{U}\mathbb{N}}$ with erasure annotations. This system, and the erasure ringoid itself, naturally satisfy the properties of Chapter 4 but the specific structure of 0ω gives some additional properties. The first is the already mentioned property that well-resourced variables never are erasable in head position.

Theorem 5.3. *If $\gamma \triangleright x_i$ then $\gamma(x_i) = \omega$.*

Proof. Using the variable case of the inversion lemma for modalities (theorem 4.10) and that ω is the least element of 0ω. □

The second property of the erasure modality which we require is that addition of annotations yields smaller (less specific) annotations. Through subsumption, this allows addition of two contexts to be used whenever either one was expected. The same property also holds for the recurrence operator since it coincides with addition.

Theorem 5.4. *Addition and the recurrence operator are decreasing function both on erasure annotations and erasure contexts, that is, $p + q \leq p$, $p + q \leq q$ and similarly for contexts (and analogous for $p \circledast_r q$).*

Proof. For annotations, by case distinction or by theorem 4.4, noting that addition, the recurrence operation and meet coincide. For contexts, by induction on the context length using the corresponding property for annotations. □

5.2 Target Language

In order for erasure annotations to be useful, a compiler of sorts that removes erasable terms is needed. This will be the topic of the next section. Here, we introduce $\lambda^{\mathbb{N}\times\top}$ which is the target language that $\lambda_{0\omega}^{\Sigma\Pi\mathbb{U}\mathbb{N}}$ will be compiled to.

The target language for our erasure extraction is an untyped lambda calculus which, with the exception of types, includes most constructs of $\lambda_{0\omega}^{\Sigma\Pi\mathbb{U}\mathbb{N}}$. Terms are, as for $\lambda^{\Sigma\Pi\mathbb{U}\mathbb{N}}$, natural number indexed to ensure correct boundness of variables. The abstract syntax for terms, $\mathsf{Term}_n^{\mathbb{N}\times\top}$, is given in Figure 5.1, meta-variables v and w are used to range over $\lambda^{\mathbb{N}\times\top}$ terms. The binders are, disregarding the lack of types,

the same as for $\lambda_{\mathbb{M}}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$, that is, v in $\lambda\, v$, s in $\mathsf{natrec}\, z\, s\, m$ and w in $\mathsf{prodrec}\, v\, w$. In addition to natural numbers, products and unit, the syntax includes $\lightning$, representing an undefined value. This term is necessary as the erased value of types, of which there are no other proper representation in the target language and will also be used in the extraction function to represent erased variables. In addition to the similar syntax, weakenings and substitutions are defined analogously to $\lambda_{\mathbb{M}}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$.

$$\frac{}{x_i \in \mathsf{Term}_n^{\mathbb{N}\times\top}}\, i < n \qquad \frac{v \in \mathsf{Term}_{n+1}^{\mathbb{N}\times\top}}{\lambda\, v \in \mathsf{Term}_n^{\mathbb{N}\times\top}} \qquad \frac{v \in \mathsf{Term}_n^{\mathbb{N}\times\top} \quad w \in \mathsf{Term}_n^{\mathbb{N}\times\top}}{v\, w \in \mathsf{Term}_n^{\mathbb{N}\times\top}}$$

$$\frac{}{\mathsf{zero} \in \mathsf{Term}_n^{\mathbb{N}\times\top}} \qquad \frac{v \in \mathsf{Term}_n^{\mathbb{N}\times\top}}{\mathsf{suc}\, v \in \mathsf{Term}_n^{\mathbb{N}\times\top}} \qquad \frac{z \in \mathsf{Term}_n^{\mathbb{N}\times\top} \quad s \in \mathsf{Term}_{n+2}^{\mathbb{N}\times\top} \quad m \in \mathsf{Term}_n^{\mathbb{N}\times\top}}{\mathsf{natrec}\, z\, s\, m \in \mathsf{Term}_n^{\mathbb{N}\times\top}}$$

$$\frac{v \in \mathsf{Term}_n^{\mathbb{N}\times\top} \quad w \in \mathsf{Term}_n^{\mathbb{N}\times\top}}{(v, w) \in \mathsf{Term}_n^{\mathbb{N}\times\top}} \qquad \frac{v \in \mathsf{Term}_n^{\mathbb{N}\times\top}}{\mathsf{fst}\, v \in \mathsf{Term}_n^{\mathbb{N}\times\top}} \qquad \frac{v \in \mathsf{Term}_n^{\mathbb{N}\times\top}}{\mathsf{snd}\, v \in \mathsf{Term}_n^{\mathbb{N}\times\top}}$$

$$\frac{v \in \mathsf{Term}_n^{\mathbb{N}\times\top} \quad w \in \mathsf{Term}_{n+2}^{\mathbb{N}\times\top}}{\mathsf{prodrec}\, v\, w \in \mathsf{Term}_n^{\mathbb{N}\times\top}} \qquad \frac{}{\star \in \mathsf{Term}_n^{\mathbb{N}\times\top}} \qquad \frac{}{\lightning \in \mathsf{Term}_n^{\mathbb{N}\times\top}}$$

Figure 5.1: Abstract syntax of the target language.

Likewise, the semantics follow the call-by-value, weak-head reduction style of $\lambda_{\mathbb{M}}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$. The reduction rules, shown in Figure 5.2, thus have many similarities to those of $\lambda_{\mathbb{M}}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ while being untyped. Values are again terms in WHNF, which for closed terms are lambda abstractions, natural numbers, pairs, $\star$ and $\lightning$. We also define $t \longrightarrow^* u$ to be the reflexive, transitive closure of the reduction relation.

$$\frac{v \longrightarrow v'}{v\, w \longrightarrow v'\, w} \qquad \frac{}{(\lambda\, v)\, w \longrightarrow v[w]} \qquad \frac{v \longrightarrow v'}{\mathsf{fst}\, v \longrightarrow \mathsf{fst}\, v'} \qquad \frac{v \longrightarrow v'}{\mathsf{snd}\, v \longrightarrow \mathsf{snd}\, v'}$$

$$\frac{}{\mathsf{fst}\, (v, w) \longrightarrow v} \qquad \frac{}{\mathsf{snd}\, (v, w) \longrightarrow w} \qquad \frac{v \longrightarrow v'}{\mathsf{prodrec}\, v\, w \longrightarrow \mathsf{prodrec}\, v'\, w}$$

$$\frac{}{\mathsf{prodrec}\, (v, v')\, w \longrightarrow w[v, v']} \qquad \frac{v \longrightarrow v'}{\mathsf{natrec}\, z\, s\, v \longrightarrow \mathsf{natrec}\, z\, s\, v'}$$

$$\frac{}{\mathsf{natrec}\, z\, s\, \mathsf{zero} \longrightarrow z} \qquad \frac{}{\mathsf{natrec}\, z\, s\, (\mathsf{suc}\, v) \longrightarrow s[\mathsf{natrec}\, z\, s\, v, v]}$$

Figure 5.2: Small-step reduction rules for $\lambda^{\mathbb{N}\times\top}$.

As for $\lambda_{\mathbb{M}}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$, the reduction relation is deterministic and reduces all terms to WHNF:

Theorem 5.5. *If $t \longrightarrow u$ with t in WHNF, then $t = u$.*

Proof. By induction on the reduction relation. □

Theorem 5.6. *If $t \longrightarrow^* u$ and $t \longrightarrow^* u'$ with u and u' in WHNF, then $u = u'$.*

Proof. By induction on the reduction relation. □

5.3 Program Extraction

With the target language in place, we can now define our compilation scheme. It takes the form of a program extraction function taking terms of $\lambda_{0\omega}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ to terms of $\lambda^{\mathbb{N}\times\top}$.

Definition 5.7 (Program extraction). The program extraction function $\_^\bullet : \mathsf{Term}_n^{0\omega} \to \mathsf{Term}_n^{\mathbb{N}\times\top}$ is defined recursively on the source language terms as follows:

$$
\begin{aligned}
\mathsf{U}^\bullet &:= \lightning & (t, u)^\bullet &:= (t^\bullet, u^\bullet)\\
\mathbb{N}^\bullet &:= \lightning & (\mathsf{fst}\, t)^\bullet &:= \mathsf{fst}\, t^\bullet\\
\top^\bullet &:= \lightning & (\mathsf{snd}\, t)^\bullet &:= \mathsf{snd}\, t^\bullet\\
\bot^\bullet &:= \lightning & \mathsf{zero}^\bullet &:= \mathsf{zero}\\
(\Pi_p^q\, F\, G)^\bullet &:= \lightning & (\mathsf{suc}\, t)^\bullet &:= \mathsf{suc}\, t^\bullet\\
(\Sigma^q\, F\, G)^\bullet &:= \lightning & (\mathsf{natrec}_p^r\, A\, z\, s\, n)^\bullet &:= \mathsf{natrec}\, z^\bullet\, s^\bullet\, n^\bullet\\
x_i^\bullet &:= x_i & \star^\bullet &:= \star\\
(\lambda^p\, t)^\bullet &:= \lambda\, t^\bullet & (\mathsf{emptyrec}_p\, A\, t)^\bullet &:= \lightning\\
(t\ {}^0u)^\bullet &:= t^\bullet\ \lightning & (\mathsf{prodrec}_0\, A\, t\, u)^\bullet &:= u^\bullet[\lightning, \lightning]\\
(t\ {}^\omega u)^\bullet &:= t^\bullet\ u^\bullet & (\mathsf{prodrec}_\omega\, A\, t\, u)^\bullet &:= \mathsf{prodrec}\, t^\bullet\, u^\bullet
\end{aligned}
$$

The similar structure between $\lambda_{0\omega}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ and $\lambda^{\mathbb{N}\times\top}$ makes the definition mostly straightforward. Since the target language is untyped, any type is necessarily extracted to $\lightning$. For most other terms, extraction preserves the structure of the term and simply recurses down over the sub-terms. More interesting are the places at which erasure occurs and this scheme no longer applies. For applications, a computationally irrelevant applied argument is simply replaced with $\lightning$ and the extraction continues over the function body. The other case in which erasure takes place is for the product recursor. Here, $\lightning$ is similarly used by substituting it for both components of the (irrelevant) product. Intuitively, the soundness of these erasure schemes is justified by theorem 5.3.

Also noteworthy are the places where erasure *does not* take place. These are the natural number recursor and the lambda abstractions, neither of which make use of their respective annotations. In the case of natrec, a similar erasure technique to the one used by the product recursor would appear feasible at first glance. The problem with this method is that a distinction between the zero and successor branches cannot be made without evaluating n. This means that the natrec structure necessarily

needs to be kept which makes any erasing attempt seemingly impossible. For the lambda abstractions, the structure of the term is also necessary to keep, in this case for the term to stay a value in both languages. We will briefly return to this in Section 7.2.

To conclude this chapter, we take a look at a small example to see how the erasure system is used. Probably the smallest illuminating example is the lambda term $\lambda x.\lambda y.y$ (with named variables) which uses y once and does not use x at all. Translating this term to $\lambda_{0\omega}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ we thus annotate the first lambda abstraction with 0 and the second with ω and get $\lambda^0\,\lambda^\omega\,x_0$. This term on its own is, as just discussed, a value so all that happens when it is extracted is that the annotations are removed, $(\lambda^0\,\lambda^\omega\,x_0)^\bullet = \lambda\,\lambda\,x_0$. If we apply some arguments to this extraction we can see the value of erasure. For instance, $((\lambda^0\,\lambda^\omega\,x_0)\,{}^0(\lambda^\omega\,x_0))\,{}^\omega\mathsf{zero}$, where we first apply the identity function $\lambda^\omega\,x_0$ and then zero, reduces to zero. Extracting this term, we get $(((\lambda^0\,\lambda^\omega\,x_0)\,{}^0(\lambda^\omega\,x_0))\,{}^\omega\mathsf{zero})^\bullet = ((\lambda\,\lambda\,x_0)\,\lightning)\,\mathsf{zero}$. The first, unused, argument was replaced with $\lightning$, the erasure annotations were removed and everything else was untouched. The gain in doing this was that the unused argument $\lambda^\omega\,x_0$ was replaced with the smaller argument $\lightning$, resulting in a smaller term. In this case, the gain was quite small, but for a sufficiently large unused argument, the improvement achieved by the erasure system would become significant.

6. Soundness

The goal of this chapter is to prove the extraction function of the previous section to be sound with regards to the semantics of $\lambda_{0\omega}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$. The method of choice is through a logical relation, relating terms of $\lambda_{0\omega}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ to terms of $\lambda^{\mathbb{N}\times\top}$ which we will define in Section 6.3. This relation will be based on the logical relation employed by Abel et al. to prove decidability of conversion for $\lambda^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ [8]. We thus first give a somewhat simplified introduction to the parts of this relation which we will use in Sections 6.1 and 6.2.

6.1 A Logical Relation for Reducibility

What we refer to as the logical relation used by Abel et al. [8] actually consists of several relations which can be divided into two groups: *reducibility* and *validity*. The general definition scheme for these relations is through induction-recursion in which one relation is defined inductively, and the following recursively on the derivation of the first. Another commonality of the relations is that they are indexed by a *type level*, ℓ which for $\lambda_{\mathbb{M}}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ ranges over 0 and 1 with 0 being the level of small types and 1 the level of large types. The relations are defined inductively over ℓ with the definition for large types generally being equal to that for small types. The exception to this is the universe U which unlike all other types can only be seen as a large type.

The presentation we give here for the reducibility relations and in the next section for the validity relations is somewhat simplified on two accounts. Primarily, parts of the relation which concern type or term conversion are omitted as they will not be relevant in the definition of our logical relation. Secondly, we limit the presentation to the simplified case of closed terms. The reason for this is that our soundness statement (and our logical relation) is restricted to closed terms. We again refer the interested reader to Abel et al. [8] for the full details. With slight modifications to account for the changed syntax, all proofs in this and the following section are attributed to this work.

The first group of the logical relation is that of reducible types and terms[1]. The general structure of the definition is similar to that of well-formed types and terms from Section 3.2 in that the definitions are mutually dependent but we again separate them here for clarity. The judgements take the forms $\epsilon \Vdash_\ell A$ and $\epsilon \Vdash_\ell t : A/\mathscr{A}$ for reducible type A and reducible term t of type A in type level ℓ respectively. In the latter judgement, $\mathscr{A}$ is the derivation that A is a reducible type that the judgement

[1]The full definition additionally includes reduciblity of type and term equality.

$$\langle \mathsf{U} \rangle \frac{}{\epsilon \Vdash_\ell \mathsf{U}}\, \ell' < \ell \qquad \langle \mathbb{N} \rangle \frac{\epsilon \vdash A \quad \epsilon \vdash A \longrightarrow^* \mathbb{N}}{\epsilon \Vdash_\ell A} \qquad \langle \top \rangle \frac{\epsilon \vdash A \quad \epsilon \vdash A \longrightarrow^* \top}{\epsilon \Vdash_\ell A}$$

$$\langle \bot \rangle \frac{\epsilon \vdash A \quad \epsilon \vdash A \longrightarrow^* \bot}{\epsilon \Vdash_\ell A} \qquad \langle \Pi \rangle \frac{\epsilon \vdash A \qquad \epsilon \vdash F \qquad \epsilon, F \vdash G \qquad \mathscr{F} : \epsilon \Vdash_\ell F \quad \epsilon \vdash A \longrightarrow^* \Pi_p^q\, F\, G \quad \forall a.\, \epsilon \Vdash_\ell a : F/\mathscr{F} \Rightarrow \epsilon \Vdash_\ell G[a]}{\epsilon \Vdash_\ell A}$$

$$\langle \Sigma \rangle \frac{\epsilon \vdash A \qquad \epsilon \vdash F \qquad \epsilon, F \vdash G \qquad \mathscr{F} : \epsilon \Vdash_\ell F \quad \epsilon \vdash A \longrightarrow^* \Sigma^q\, F\, G \quad \forall a.\, \epsilon \Vdash_\ell a : F/\mathscr{F} \Rightarrow \epsilon \Vdash_\ell G[a]}{\epsilon \Vdash_\ell A} \qquad \langle \mathrm{emb} \rangle \frac{\epsilon \Vdash_{\ell'} A}{\epsilon \Vdash_\ell A}\, \ell' < \ell$$

Figure 6.1: Definition of reducible types.

is defined recursively on. We write $\mathscr{A} : \epsilon \Vdash_\ell A$ to indicate that $\mathscr{A}$ is a derivation of $\epsilon \Vdash_\ell A$ and similarly for the other judgements.

Reducible types are defined inductively according to Figure 6.1, we use the labels on the left to refer to each specific derivation in the definition of the other judgements. The universe type, U is always a reducible type of type level 1, the lack of premise comes from there being no terms that reduce to U. The type level side condition could alternatively be written $l = 1$, constraining the derivation to large types. The following three rules state that any well-formed type that reduces to either $\mathbb{N}$, $\top$ or $\bot$ is a reducible type. The cases corresponding to Π and Σ types additionally require that F is a reducible type and that $G[a]$ is a reducible type[2] whenever a is a reducible term of type F. Finally, any small type can also be considered a large type, by which we allow any derivation for small types to be embedded to large types.

We now define reducible terms $\epsilon \Vdash_\ell t : A/\mathscr{A}$ by recursion on $\mathscr{A}$ as follows:

- If $\mathscr{A} = \langle \mathsf{U} \rangle$ then $\epsilon \Vdash_\ell t : U/\mathscr{A}$ holds iff there exists t' in WHNF such that $\epsilon \vdash t \longrightarrow t' : U$.
- If $\mathscr{A} = \langle \mathbb{N} \rangle$ then $\epsilon \Vdash_\ell t : A/\mathscr{A}$ holds iff there exists t' in WHNF such that $\epsilon \vdash t \longrightarrow t' : \mathbb{N}$ and
  - $t' = \mathsf{zero}$ or
  - $t' = \mathsf{suc}\, u$ and $\epsilon \Vdash_\ell u : A/\mathscr{A}$ holds.
- If $\mathscr{A} = \langle \top \rangle$ then $\epsilon \Vdash_\ell t : A/\mathscr{A}$ holds iff there exists t' in WHNF such that $\epsilon \vdash t \longrightarrow t' : \top$.

[2]The full definition requires these properties to hold under any weakening applied to F and G. This has been omitted because only closed terms are considered.

- If $\mathscr{A} = \langle \bot \rangle$ then $\epsilon \Vdash_\ell t : A/\mathscr{A}$ does not hold[3].
- If $\mathscr{A} = \langle \Pi \rangle$ then there are derivations $\mathscr{F} : \epsilon \Vdash_\ell F$ and[4] $\mathscr{G} : \forall a.\ \epsilon \Vdash_\ell a : F/\mathscr{F} \Rightarrow \epsilon \Vdash_\ell G[a]$. We then define $\epsilon \Vdash_\ell t : A/\mathscr{A}$ to hold iff there exists t' in WHNF such that $\epsilon \vdash t \longrightarrow t' : \Pi^q_p\, F\, G$ and[5] $\forall a.\ \epsilon \Vdash_\ell a : F/\mathscr{F} \Rightarrow \epsilon \Vdash_\ell t'\, {}^p a : G[a]/\mathscr{G}(a)$.
- If $\mathscr{A} = \langle \Sigma \rangle$ then there are derivations $\mathscr{F} : \epsilon \Vdash_\ell F$ and $\mathscr{G} : \forall a.\ \epsilon \Vdash_\ell a : F/\mathscr{F} \Rightarrow \epsilon \Vdash_\ell G[a]$. We then define $\epsilon \Vdash_\ell t : A/\mathscr{A}$ to hold iff there exists t' in WHNF such that $\epsilon \vdash t \longrightarrow t' : \Sigma^q\, F\, G$ and $\forall a.\ \epsilon \Vdash_\ell a : F/\mathscr{F} \Rightarrow \epsilon \Vdash_\ell \mathsf{fst}\, t' : F/\mathscr{F} \wedge \epsilon \Vdash_\ell \mathsf{snd}\, t' : G[a]/\mathscr{G}(a)$.
- If $\mathscr{A} = \langle \mathrm{emb} \rangle$ then there is a derivation $\mathscr{A}' : \epsilon \Vdash_{\ell'} A$ and $\epsilon \Vdash_\ell t : U/\mathscr{A}$ is defined to hold iff $\epsilon \Vdash_{\ell'} t : A/\mathscr{A}'$ holds.

Before we move on to the remaining parts of the logical relation we give some properties of the reducibility relations. First, the escape lemma which states that reducibility entails well-formedness.

Theorem 6.1 (Escape). *If $\mathscr{A} : \epsilon \Vdash_\ell A$ then $\epsilon \vdash A$. If additionally, $\epsilon \Vdash_\ell t : A/\mathscr{A}$ then $\epsilon \vdash t : A$.*

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

The second main property is *irrelevance* which refers to the actual derivation of reducibility being irrelevant.

Theorem 6.2 (Irrelevance). *If $\mathscr{A} : \epsilon \Vdash_\ell A$ and $\mathscr{A}' : \epsilon \Vdash_{\ell'} A$ and $\epsilon \Vdash_\ell t : A/\mathscr{A}$ then $\epsilon \Vdash_{\ell'} t : A/\mathscr{A}'$.*

Proof. By mutual induction on $\mathscr{A}$ and $\mathscr{A}'$ using theorem 3.2. □

6.2 A Logical Relation for Validity

Though the escape lemma shows that reducible types and terms are well-formed, it is not yet possible to show the converse (this property is the fundamental lemma of the logical relation). In order to show this, the reducibility relations need to be generalised to hold under substitutions. This extended property forms the logical relation for validity of which there are four judgments which we will consider. These

[3] The requirements for term reducibility for this case reduces to impossibility for closed terms since there are no closed terms of the empty type.

[4] Here, $\mathscr{G}$ represents a (possibly) infinite number of derivations for each reducible term a. We write $\mathscr{G}(a)$ for the specific derivation using a, which implicitly is required to be reducible.

[5] Again, the full definition requires the following property (and the analogous one for $\langle \Sigma \rangle$) to hold under any weakening.

are valid contexts, $\Vdash^v \Gamma$, valid substitutions[6], $\epsilon \Vdash^s \sigma : \Gamma/\mathscr{T}$, valid types $\Gamma \Vdash^v_\ell A/\mathscr{T}$, and valid terms $\Gamma \Vdash^v_\ell t : A/\mathscr{T}/\mathscr{A}$. The first three all depend on each other, requiring them to be defined together, but are as usual separated here. The induction-recursion scheme is used also here with context validity being defined inductively and substitution validity defined by recursion on that derivation.

Starting with valid contexts, the inductive definition is given in Figure 6.2. This definition simply states that the empty context is valid and that a valid context can be extended with valid types. Similarly to the definition of reducible types, the labels on the left are used to refer to a specific derivation.

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

Figure 6.2: Definition of validity of contexts.

Validity of substitutions, $\epsilon \Vdash^s \sigma : \Gamma/\mathscr{T}$ is then defined by recursion on the context validity derivation as follows:

- If $\mathscr{T} = \langle\epsilon\rangle$ then $\epsilon \Vdash^s \sigma : \epsilon/\mathscr{T}$ holds.
- If $\mathscr{T} = \langle\text{cons}\rangle$ then there are derivations $\mathscr{T} : \Vdash^v \Gamma$ and $\mathscr{A} : \Gamma \Vdash^v_\ell A/\mathscr{T}$ and we define $\epsilon \Vdash^s \sigma : \epsilon/\mathscr{T}$ to hold iff $\mathscr{S} : \epsilon \Vdash^s \mathsf{tail}\,\sigma : \Gamma/\mathscr{T}$ and $\epsilon \Vdash_\ell \mathsf{head}\,\sigma : A[\mathsf{tail}\,\sigma]/\mathscr{A}(\mathsf{tail}\,\sigma)$ hold.

The definition is similar to the validity of contexts. Empty substitutions are always valid and non-empty substitutions are valid if the head is a reducible term and the tail is a valid substitution.

Validity of types is defined such that $\Gamma \Vdash^v_\ell A/\mathscr{T}$ holds iff $\forall\sigma.\ \epsilon \Vdash^s \sigma : \Gamma/\mathscr{T} \Rightarrow \epsilon \Vdash_\ell A[\sigma]$ and validity for terms $\Gamma \Vdash^v_\ell t : A/\mathscr{T}/\mathscr{A}$ similarly holds iff $\forall\sigma.\ \epsilon \Vdash^s \sigma : \Gamma/\mathscr{T} \Rightarrow \epsilon \Vdash_\ell t[\sigma] : A[\sigma]/\mathscr{A}(\sigma)$. Both cases simply state that a type or term is valid iff it is reducible when applied to any valid substitution. This allows the fundamental theorem of the logical relation to be proven.

Theorem 6.3 (Fundamental lemma for validity and reducibility).

- *If* $\Gamma \vdash A$ *then* $\exists\mathscr{T}.\ \Gamma \Vdash^v_1 A/\mathscr{T}$.
- *If* $\Gamma \vdash t : A$ *then* $\exists\mathscr{T}, \mathscr{A}.\ \Gamma \Vdash^v_1 t : A/\mathscr{T}/\mathscr{A}$.
- *If* $\epsilon \vdash A$ *then* $\epsilon \Vdash_1 A$.

[6]We only consider closing substitutions, that is $\sigma \in \mathsf{Subst}^n_0$.

- *If* $\epsilon \vdash t : A$ *then* $\exists \mathscr{A}.\ \epsilon \Vdash_1 t : A/\mathscr{A}$.

Proof. Validity is shown by induction on the typing derivation. Reducibility follows from the validity case applied to the identity substitution. □

The validity judgements also have similar irrelevance properties to the reducibility judgements.

Theorem 6.4 (Irrelevance). *If* $\mathscr{T} : \Vdash^v \Gamma$ *and* $\mathscr{T}' : \Vdash^v \Gamma$ *and* $\mathscr{A} : \Gamma \Vdash^v_\ell A/\mathscr{T}$ *and* $\mathscr{A}' : \Gamma \Vdash^v_\ell A/\mathscr{T}'$ *the following statements hold:*

- *If* $\Gamma \Vdash^v_\ell A/\mathscr{T}$ *then* $\Gamma \Vdash^v_\ell A/\mathscr{T}'$.
- *If* $\Gamma \Vdash^v_\ell t : A/\mathscr{T}/\mathscr{A}$ *then* $\Gamma \Vdash^v_\ell t : A/\mathscr{T}'/\mathscr{A}'$.

Proof. By induction on the context validity judgements using irrelevance for the reducibility relations. □

Having proven the fundamental lemma, Abel et al. use the logical relation to show various properties of the type system, concluding in showing that type and term conversion are decidable. For our purposes, the main property of interest is syntactic validity.

Theorem 6.5 (Syntactic validity).

- *If* $\Gamma \vdash t : A$ *then* $\Gamma \vdash A$.
- *If* $\Gamma \vdash A \longrightarrow^* B$ *then* $\Gamma \vdash A$ *and* $\Gamma \vdash B$.
- *If* $\Gamma \vdash t \longrightarrow^* u : A$ *then* $\Gamma \vdash t : A$ *and* $\Gamma \vdash u : A$.

Proof. By the fundamental theorem and the escape lemma. □

6.3 A Logical Relation for Erasure

Using the logical relations for reducibility and validity, we can now move on to define our logical relation for erasure. Like the previous relations, we will require a collection of relations which we will collectively refer to as the logical relation. If distinctions between the other relations need to be made, we refer to this one as the logical relation for erasure. The structure of the logical relation is similar to those of the previous sections, being defined using the induction-recursion scheme and being indexed by a type level ℓ. There is, however, one major difference between the relation for erasure and the relations for reducibility and validity. While the latter two only cover single terms, the relation for erasure relates terms between $\lambda_0^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ to

terms of $\lambda^{\mathbb{N}\times\top}$. The intent is for every term t to be related to its erasure $t^\bullet$ (this is the fundamental lemma of the logical relation).

The main part of the logical relation, $t \mathrel{®}_\ell v : A/\mathscr{A}$ relates a closed term t in $\lambda_\omega^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ of some reducible type A to a closed term v in $\lambda^{\mathbb{N}\times\top}$. The definition is done by recursion on the derivation that A is reducible:

- If $\mathscr{A} = \langle \mathsf{U} \rangle$ then $t \mathrel{®}_\ell v : A/\mathscr{A}$ holds iff $\epsilon \vdash t : \mathsf{U}$.
- If $\mathscr{A} = \langle \mathbb{N} \rangle$ then $t \mathrel{®}_\ell v : A/\mathscr{A}$ holds iff either
  - $\epsilon \vdash t \longrightarrow^* \mathsf{zero} : \mathbb{N}$ and $v \longrightarrow^* \mathsf{zero}$ or
  - $\epsilon \vdash t \longrightarrow^* \mathsf{suc}\, t' : \mathbb{N}$, $v \longrightarrow^* \mathsf{suc}\, v'$ and $t' \mathrel{®}_\ell v' : A/\mathscr{A}$.
- If $\mathscr{A} = \langle \top \rangle$ then $t \mathrel{®}_\ell v : A/\mathscr{A}$ holds iff $\epsilon \vdash t : \top$ and $v \longrightarrow^* \star$.
- If $\mathscr{A} = \langle \bot \rangle$ then $t \mathrel{®}_\ell v : A/\mathscr{A}$ does not hold.
- If $\mathscr{A} = \langle \Pi \rangle$, $\epsilon \vdash A \longrightarrow^* \Pi_p^q\, F\, G$ holds and there are derivations $\mathscr{F} : \epsilon \Vdash_\ell F$ and $\mathscr{G} : \forall a.\ \epsilon \Vdash_\ell a : F/\mathscr{F} \Rightarrow \epsilon \Vdash_\ell G[a]$. We then define $t \mathrel{®}_\ell v : A/\mathscr{A}$ to hold iff either
  - $p = 0$ and $\forall a.\ \epsilon \Vdash_\ell a : F/\mathscr{F} \Rightarrow t\, {}^0a \mathrel{®}_\ell v\, \lightning : G[a]/\mathscr{G}(a)$ holds, or
  - $p = \omega$ and $\forall a, w.\ \epsilon \Vdash_\ell a : F/\mathscr{F} \Rightarrow a \mathrel{®}_\ell w : F/\mathscr{F} \Rightarrow t\, {}^\omega a \mathrel{®}_\ell v\, w : G[a]/\mathscr{G}(a)$ holds.
- If $\mathscr{A} = \langle \Sigma \rangle$, there are derivations $\mathscr{F} : \epsilon \Vdash_\ell F$ and $\mathscr{G} : \forall a.\ \epsilon \Vdash_\ell a : F/\mathscr{F} \Rightarrow \epsilon \Vdash_\ell G[a]$. We then define $t \mathrel{®}_\ell v : A/\mathscr{A}$ to hold iff $\exists t_1, t_2, v_1, v_2.\ \epsilon \vdash t \longrightarrow^* (t_1, t_2) : \Sigma^q\, F\, G \wedge v \longrightarrow^* (v_1, v_2) \wedge \Big(\epsilon \Vdash_\ell t_1 : F/\mathscr{F} \Rightarrow t_1 \mathrel{®}_\ell v_1 : F/\mathscr{F} \wedge t_2 \mathrel{®}_\ell v_2 : G[t_1]/\mathscr{G}(t_1)\Big)$ holds.
- If $\mathscr{A} = \langle \mathrm{emb} \rangle$ then there is a derivation $\mathscr{A}' : \epsilon \Vdash_{\ell'} A$ and $t \mathrel{®}_\ell v : A/\mathscr{A}$ is defined to hold iff $t \mathrel{®}_{\ell'} v : A/\mathscr{A}'$ holds.

For the base types, the main idea is that terms of both languages should reduce to corresponding values of the right type. In the universe case, we require t to be of type U which ensures that t is a (small) type. The natural number case is defined inductively, similarly to the natural numbers themselves. In the base case, both t and v reduce to zero while in the induction case they both reduce to the successor of some related terms on which the induction can proceed. The consequence is that t and v represent the same natural number. The unit case is similar to the universe case in that we simply require v to reduce to $\star$, the canonical value of the unit type. The last base type, the empty type, is treated as impossibility. The reason for this is, as was the case for reducibility of terms, that there are no closed terms of the

empty type.

For computationally relevant Π-types, t and v represent functions and should be related when applied to related (and in the case of t, reducible) arguments. For irrelevant Π-types, the relatedness restriction is dropped and we instead apply undefined to v. In the case of Σ-types, t and v represent pairs whose both components should be related. For embeddings, we simply induct and apply the definition to the embedded reducibility derivation.

An important property of the logical relation is that relatedness is preserved by reduction. This is true both for the $\lambda_0^{\Sigma\Pi\mathbb{U}_\mathbb{N}}$-terms and the $\lambda^{\mathbb{N}\times\top}$-terms, regardless of whether the reduction is followed backwards or forwards.

Theorem 6.6. *If* $\mathscr{A} : \epsilon \Vdash_\ell A$ *and* $t \mathrel{®}_\ell v : A/\mathscr{A}$ *then*

- *If* $\epsilon \vdash t \longrightarrow t' : A$ *then* $t' \mathrel{®}_\ell v : A/\mathscr{A}$.
- *If* $v \longrightarrow v'$ *then* $t \mathrel{®}_\ell v' : A/\mathscr{A}$.
- *If* $\epsilon \vdash t' \longrightarrow t : A$ *then* $t' \mathrel{®}_\ell v : A/\mathscr{A}$.
- *If* $v' \longrightarrow v$ *then* $t \mathrel{®}_\ell v' : A/\mathscr{A}$.
- *If* $\epsilon \vdash t \longrightarrow^* t' : A$ *and* $v \longrightarrow^* v'$ *then* $t' \mathrel{®}_\ell v' : A/\mathscr{A}$.
- *If* $\epsilon \vdash t' \longrightarrow^* t : A$ *and* $v' \longrightarrow^* v$ *then* $t' \mathrel{®}_\ell v' : A/\mathscr{A}$.

Proof. The first four cases are shown by induction on $\mathscr{A}$ using syntactic validity and *theorem* 3.2. The remaining two by induction on the reflexive transitive closure of the reduction relations, using the first four properties. □

As for the relations for reducibility and validity, we also have an irrelevance property for the derivation of the reducibility of A.

Theorem 6.7 (Irrelevance). *If* $\mathscr{A} : \epsilon \Vdash_\ell A$ *and* $\mathscr{A}' : \epsilon \Vdash_{\ell'} A$ *and* $t \mathrel{®}_\ell v : A/\mathscr{A}$ *then* $t \mathrel{®}_{\ell'} v : A/\mathscr{A}'$.

Proof. By mutual induction on $\mathscr{A}$ and $\mathscr{A}'$. □

At this point, we would like to show the fundamental lemma, $t \mathrel{®}_1 t^\bullet : A/\mathscr{A}$, for any well-typed term t but attempting to do so quickly runs us into a problem. For the lambda case ($p = \omega$), we need to show $t\,{}^\omega a \mathrel{®}_1 t^\bullet\, w : G[a]/\mathscr{G}(a)$ given $\epsilon \Vdash_1 a : F/\mathscr{F}$ and $a \mathrel{®}_1 w : F/\mathscr{F}$. By theorem 6.6 it is sufficient to show $t[a] \mathrel{®}_1 t^\bullet[w] : G[a]/\mathscr{G}(a)$ but the induction hypothesis is too weak to show this. This is the same problem that caused Abel et al. to introduce the validity relation, ensuring that the logical

relation holds under substitutions. We proceed in the same manner, starting with $\sigma \mathrel{®}_\ell \sigma' : \Gamma \triangleleft \gamma / \mathscr{T} / \mathscr{S}$ relating valid (closing) substitutions of $\lambda_\omega^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ with (closing) substitutions of $\lambda^{\mathbb{N}\times\top}$. The contexts Γ and γ keep track of the type and resource usage for each term in σ. The relation is defined by recursion on the validity of Γ as follows:

- If $\mathscr{T} = \langle\epsilon\rangle$ then $\sigma \mathrel{®}_\ell \sigma' : \epsilon \triangleleft \epsilon / \mathscr{T} / \mathscr{S}$ holds.
- If $\mathscr{T} = \langle\mathrm{cons}\rangle$ then there derivations $\mathscr{T}' : \Vdash^v \Gamma$ and $\mathscr{A} : \Gamma \Vdash^v_{\ell'} A / \mathscr{T}'$ and $\mathscr{S}' : \epsilon \Vdash^s \mathsf{tail}\,\sigma : \Gamma / \mathscr{T}'$ and $\mathscr{S} : \epsilon \Vdash^s \sigma : \Gamma, A / \mathscr{T}$ (note that $\mathscr{T} : \Vdash^v \Gamma, A$ in this case). We then define $\sigma \mathrel{®}_\ell \sigma' : \Gamma, A \triangleleft \gamma, p / \mathscr{T} / \mathscr{S}$ to hold iff $\mathsf{tail}\,\sigma \mathrel{®}_\ell \mathsf{tail}\,\sigma' : \Gamma \triangleleft \gamma / \mathscr{T}' / \mathscr{S}'$ and either
  - $p = 0$ or
  - $p = \omega$ and $\mathsf{head}\,\sigma \mathrel{®}_{\ell'} \mathsf{head}\,\sigma' : A[\mathsf{tail}\,\sigma] / \mathscr{A}(\mathsf{tail}\,\sigma)$ holds.

The idea is to treat substitutions as lists of terms and require the substitutions to be pointwise related for all computationally relevant variables but require nothing of erasable variables. The motivation for this is theorem 5.3. Since an erasable variable cannot occur, it does not matter what terms are being substituted for such variables.

Using this, we then define *erasure validity*, $\Gamma \triangleright \gamma \Vdash_\ell t : A / \mathscr{T} / \mathscr{A}$, which is defined to hold iff $\forall \sigma, \sigma'.\ \mathscr{S} : \epsilon \Vdash^s \sigma : \Gamma / \mathscr{T} \Rightarrow \sigma \mathrel{®}_\ell \sigma' : \Gamma \triangleleft \gamma / \mathscr{T} / \mathscr{S} \Rightarrow t[\sigma] \mathrel{®}_\ell t^\bullet[\sigma'] : A[\sigma] / \mathscr{A}(\sigma)$. Similarly to the definitions of valid types and terms, an open term t of a valid type A is valid under erasure if it is related to erasure under any related substitutions σ and σ'.

With this, it is possible to show the fundamental lemma, but we delay the proof a little bit longer and first show some other properties of our relations which will simplify the proof. First, as usual, we have irrelevance.

Theorem 6.8 (Irrelevance). *If $\mathscr{T} : \Vdash^v \Gamma$ and $\mathscr{T}' : \Vdash^v \Gamma$ then*

- *If $\mathscr{S} : \epsilon \Vdash^s \sigma : \Gamma / \mathscr{T}$ and $\mathscr{S}' : \epsilon \Vdash^s \sigma : \Gamma / \mathscr{T}'$ and $\sigma \mathrel{®}_\ell \sigma' : \Gamma \triangleleft \gamma / \mathscr{T} / \mathscr{S}$ then $\sigma \mathrel{®}_\ell \sigma' : \Gamma \triangleleft \gamma / \mathscr{T}' / \mathscr{S}'$.*
- *If $\mathscr{A} : \Gamma \Vdash^v_\ell A / \mathscr{T}$ and $\mathscr{A}' : \Gamma \Vdash^v_{\ell'} A / \mathscr{T}'$ and $\gamma \triangleright \Gamma \Vdash_t \ell : A / \mathscr{T} / \mathscr{A}$ then $\gamma \triangleright \Gamma \Vdash_{\ell'} t : A / \mathscr{T}' / \mathscr{A}'$.*

Proof. For substitutions, by mutual induction on $\mathscr{T}$, $\mathscr{T}'$ using theorem 6.7. For erasure validity, using irrelevance of substitutions and theorem 6.8. □

Then, a property analogous to the subsumption rule in the usage relation that allows erasable content to be treated as computationally relevant.

Theorem 6.9 (Subsumption). *If* $\mathscr{T} : \Gamma$ *and* $\gamma \leq \delta$ *then*

- *If* $\mathscr{S} : \epsilon \Vdash^s \sigma : \Gamma/\mathscr{T}$ *and* $\sigma \mathrel{®}_\ell \sigma' : \Gamma \triangleleft \gamma/\mathscr{T}/\mathscr{S}$ *then* $\sigma \mathrel{®}_\ell \sigma' : \Gamma \triangleleft \delta/\mathscr{T}/\mathscr{S}$.
- *If* $\mathscr{A} : \Gamma \Vdash^v_\ell A/\mathscr{T}$ *and* $\delta \triangleright \Gamma \Vdash_\ell t : A/\mathscr{T}/\mathscr{A}$ *then* $\gamma \triangleright \Gamma \Vdash_\ell t : A/\mathscr{T}/\mathscr{A}$.

Proof. For subsumption of related substitutions, by induction on $\mathscr{T}$ and case distinction on the head of γ, δ. Subsumption of erasure validity follows directly from subsumption of substitutions. □

A subtle difference between the two subsumption properties is the flipped order between γ and δ. For related substitutions, the theorem allows us to move to a larger context whereas erasure validity allows us to move to smaller contexts. The reason for this difference is the negative occurrence of $\sigma \mathrel{®}_\ell \sigma' : \Gamma \triangleleft \gamma/\mathscr{T}/\mathscr{S}$ in the definition of erasure validity.

With these properties, we are ready to show the fundamental lemma, starting with a lemma for the variable case.

Theorem 6.10 (Fundamental lemma for variables). *If* $\mathscr{T} : \Vdash^v \Gamma$ *and* $x_i : A \in \Gamma$ *and* $\gamma(x_i) = \omega$ *and* $\mathscr{S} : \epsilon \Vdash^s \sigma : \Gamma/\mathscr{T}$ *and* $\sigma \mathrel{®}_\ell \sigma' : \Gamma \triangleleft \gamma/\mathscr{T}/\mathscr{S}$ *then* $\exists \mathscr{A}.\ \sigma x_i \mathrel{®}_1 \sigma' x_i : A/\mathscr{A}(\sigma)$.

Proof. By mutual induction on $\mathscr{T}$ and $x_i : A \in \Gamma$ using theorems 5.3, 6.4 and 6.7. □

Theorem 6.11 (Fundamental lemma for erasure). *If* $\Gamma \vdash t : A$ *and* $\gamma \triangleright t$ *then* $\exists \mathscr{T}, \mathscr{A}.\ \gamma \triangleright \Gamma \Vdash_1 t : A/\mathscr{T}/\mathscr{A}$.

Proof. By induction on the typing derivation using the inversion lemma for the usage relation, fundamental lemma for variables and theorems 5.4 and 6.1 to 6.9. The most interesting cases are binder terms for which the proof strategy is as explained above. □

6.4 Soundness

With no less than three sets of logical relations defined, we are finally ready to show the main result of the erasure extension, namely that the extraction function is sound. As have already been mentioned, we limit the soundness result to closed programs since free variables prevent terms to be fully evaluated. We further limit ourselves to programs that output natural numbers, that is terms t of type $\mathbb{N}$.

The desired property is the following: If $\epsilon \vdash t \longrightarrow^* \mathsf{suc}^n\ \mathsf{zero} : \mathbb{N}$ and $\epsilon \triangleright t$ then $t^\bullet \longrightarrow^* \mathsf{suc}^n\ \mathsf{zero}$, where $\mathsf{suc}^0\ \mathsf{zero} = \mathsf{zero}$ and $\mathsf{suc}^{n+1}\ \mathsf{zero} = \mathsf{suc}\,(\mathsf{suc}^n\ \mathsf{zero})$. Un-

fortunately, this property does not appear to be provable in our system due to the semantics only reducing terms to WHNF. Instead, we have to settle for the next best thing, that t and $t^\bullet$ both reduce to WHNF representing the same natural number.

To formulate this property properly, we define the concept of weak head representations of natural numbers as inductive predicates on terms of $\lambda_{\mathfrak{w}}^{\Sigma\Pi\mathsf{U}\mathbb{N}}$ and $\lambda^{\mathbb{N}\times\top}$ in Figure 6.3. For both languages, the natural number 0 is represented by any term which reduces to zero and $n+1$ is represented by any term which reduces to the successor of a term that represents the number n.

$$\frac{\epsilon \vdash t \longrightarrow^* \mathsf{zero} : \mathbb{N}}{\mathsf{WH}\mathbb{N}_0\ t} \qquad \frac{\epsilon \vdash t \longrightarrow^* \mathsf{suc}\, t' : \mathbb{N} \quad \mathsf{WH}\mathbb{N}_n\ t'}{\mathsf{WH}\mathbb{N}_{n+1}\ t}$$

$$\frac{v \longrightarrow^* \mathsf{zero}}{\mathsf{WH}\mathbb{N}'_0\ v} \qquad \frac{v \longrightarrow^* \mathsf{suc}\, v' \quad \mathsf{WH}\mathbb{N}'_n\ v'}{\mathsf{WH}\mathbb{N}'_{n+1}\ v}$$

Figure 6.3: Weak head representations of natural numbers.

Under a stronger reduction relation, this corresponds to reduction to natural numbers in canonical form as shown by the following theorem.

Theorem 6.12. *If* $\epsilon \vdash t \longrightarrow^* t' : \mathbb{N} \Rightarrow \epsilon \vdash \mathsf{suc}\, t \longrightarrow^* \mathsf{suc}\, t' : \mathbb{N}$ *is added to the reduction rules of Figure 3.10 and* $\mathsf{WH}\mathbb{N}_n\ t$ *then* $\epsilon \vdash t \longrightarrow^* \mathsf{suc}^n\ \mathsf{zero} : \mathbb{N}$. *Likewise, if* $v \longrightarrow^* v' \Rightarrow \mathsf{suc}\, v \longrightarrow \mathsf{suc}\, v'$ *is added to the reduction rules of Figure 5.2 and* $\mathsf{WH}\mathbb{N}'_n\ v$ *then* $v \longrightarrow^* \mathsf{suc}^n\ \mathsf{zero}$.

Proof. By induction on $\mathsf{WH}\mathbb{N}_n\ t$ and $\mathsf{WH}\mathbb{N}'_n\ v$ respectively using transitivity of the reduction relation. □

Convinced that $\mathsf{WH}\mathbb{N}_n\ t$ and $\mathsf{WH}\mathbb{N}'_n\ v$ are satisfying representations of natural numbers, we now show soundness of natural numbers with regards to this representation via a quick lemma.

Theorem 6.13. *If* $t \mathrel{®_\ell} v : \mathbb{N}/\langle\mathbb{N}\rangle$ *and* $\mathsf{WH}\mathbb{N}_n\ t$ *then* $\mathsf{WH}\mathbb{N}'_n\ t^\bullet$.

Proof. By induction on $\mathsf{WH}\mathbb{N}_n\ t$ using theorem 3.2. □

Theorem 6.14 (Soundness). *If* $\epsilon \vdash t : \mathbb{N}$ *and* $\epsilon \triangleright t$ *and* $\mathsf{WH}\mathbb{N}_n\ t$ *then* $\mathsf{WH}\mathbb{N}'_n\ t^\bullet$.

Proof. By theorem 6.13 using the fundamental lemma for erasures and theorem 6.7. □

7. Discussion

In this chapter we return to the discussion we started in Section 4.2.7 about the usage rules for the projections and natural recursor and provide some alternative approaches. We then take a step back and look at some related work and possible areas for future work in this area.

7.1 Alternative Design Choices for the Usage Relation

The usage relation, for the most part, is relatively straightforward. The two main exceptions are the projections and the natural number recursor for which the subject reduction theorem put strict requirements on the usage rules defined in Figure 4.5. We discussed our design choices for these terms in Section 4.2.7, but the chosen ruleset is not the only possible way to treat these terms. In this section, we will discuss some alternative design choices that were considered to ensure that subject reduction holds.

7.1.1 Projections

The restriction that no resources are allowed for the use of projections is quite strict and severely decreases their usefulness. Recall from Section 4.2.7, if $\gamma \triangleright t$, $\delta \triangleright u$, we would need $\delta \leq \mathbf{0}$ for $\mathsf{fst}\,(t, u)$ to be well-resourced. As suggested in Section 4.2.7, letting the usage rule enforce this is infeasible since it would need access to both components of the pair, thus only allowing pairs to be in WHNF. Not only is this a large limitation on the usefulness of pairs, but it also makes the use of projections essentially redundant.

Another way to enforce this inequality is to change the definition of the modality ringoid to ensure that 0 is the greatest element. While this would indeed make the projections more useful, it is a large intrusion on the generality of the system. For instance, although the constraint would hold for the erasure modality, it does not hold for many typical use-cases such as that of linear types [1]. Yet another possibility is to extend the system with further annotations, allowing a more fine-grained resource treatment. Such an extension would most likely use annotations for the products themselves, similar to the treatment of erasure for Σ-types by Atkey [10] as well as annotations on the projections. We leave the details of formulating such a system as future work.

7.1.2 Natural Number Recursion

Without considering the details, the inclusion of the recurrence operator appears to be a quite unnatural addition to the definition of the modality ringoid. Indeed, unlike the other three operations, it is strictly speaking not needed to form the system and its use in the rule for natrec could be replaced by a side condition ensuring that the recurrence inequality holds. This solution has two main problems. Firstly, this breaks the usage inference function, making it impossible to infer valid contexts in general. Secondly, the definition of the recurrence operator explicitly ensures that any possible recurrence inequality *has* a solution. Losing that property means that it could be possible, in some modalities, for natrec-terms to be ill-resourced though one would expect them not to. In our approach, these situations are avoided by disallowing the modality to be used (since it does not satisfy our definition). There is a tradeoff to be made with regards to either allowing more modalities to be used on one hand and ensuring that the system behaves as expected on the other where we decided to side with the latter alternative.

Another approach that avoids introducing the recurrence operator, is based on the work of Brunel et al. [13]. Their quantitative calculus includes both a fixpoint operation as well as case splitting on natural numbers which can be combined to express the behaviour of our natrec. In that formulation, the resource contexts for both case branches are required to be equal which in our case would mean $\gamma = \delta$ and the conclusion of the usage rule becomes $r^*(\gamma + p\eta) \triangleright \mathsf{natrec}_p^r\, A\, z\, s\, n$. This replaces the need for the $\circledast_r$-operation with the unary star-operation[1] satisfying $r^* = 1 + rr^*$. While this formulation is thought to be sound it appears to be too restrictive on which contexts are allowed, again making some terms which one would expect to be well-resourced be ill-resourced.

In a similar vein to restricting 0 to be the greatest element of the modality ringoid, another solution would be to require the existence of a least element. The least context would then always be a solution to the recurrence inequality allowing it to be used in the conclusion of the usage rule. Much like requiring that 0 is the greatest element, this limits the generality of the system and additionally does not allow for a particularly precise treatment of the usage resource of natrec.

7.2 Related Work

The main inspiration for the modality extension comes from the general view of modalities in types introduced by Abel and Bernardy [1] on which most of our definition of modalities rests. A similar use of a ring-like structure for quantitive typing was proposed by Brunel et al. [13]. Their framework was similarly built on a semiring structure equipped with a bounded partial order but unlike Abel and Bernardy, considered recursion in the form of fixpoint iteration using the boundness restriction. As discussed in the previous section, this approach was considered for

[1] This is the characteristic operation accossiated with a Star-semiring. In the context of regular expressions and finite automata, it is also known as the Kleene star.

the recursion present in this work but was ultimately considered to not be suitable.

Several other uses of ring-like structures to achieve quantitive types also exist but are typically restricted to specific use cases. Wood and Atkey use a three-element semiring, equipped with a partial order to account for linearity and erasure [12]. Although their setting is restricted to linear types, and the studied language differs from ours, most usage rules coincide with ours. In the same work, Wood and Atkey also introduce the use of matrices to keep track of modality contexts after substitutions which we too have adopted. A similar approach was used by McBride to simultaneously keep track of both linearity and erasability [2]. This work was later extended by Atkey to include, among other things, Σ-types [10], the formulation of which much of our treatment of Σ-types is inspired.

Examples of erasure systems in a dependently typed setting include the already mentioned work by McBride where programs are extracted to the untyped lambda calculus [2]. For erasure, his three-element semiring behaves essentially identically to our erasure modality but unlike our work, extraction is not performed using a function, but rather through a relation between terms of the source and target languages. The works of Mishra-Linger and Sheard [5] and Barras and Bernardo [14] skip the semiring and simply annotate content as either *runtime* or *compile-time*, corresponding to computationally relevant and erasable content respectively. The extraction function approach used in these works is largely similar to ours, but erasable lambda abstractions are extracted to just the body. This approach allows for more erasable content to be removed at the cost of value preservation. Tejiščák uses a similar annotation scheme but additionally allows missing annotations to be inferred [6]. Using this approach, a programmer only needs to annotate parts of a program to gain the advantage of erasure.

7.3 Future Work

One advantage of using tools like Agda for formalisation is that it makes it easy to build on and extend previous work. In this regard, there are several areas suitable for future work.

Although the language has grown considerably since the Agda formalisation was originally published by Abel et al. [8], the language still lacks some of the common types found in MLTT. Extending the formalisation with these missing types, most notably identity and W-types, is thus a clear candidate for extending this work. Of course, at this point, extending the language includes more than just updating the language definition since both the formulation of decidability of conversion and the formulation of erasures need to be updated to account for the larger language.

A more direct extension of this work involves the modality extension. We have already suggested that a more satisfying treatment of Σ-types could be achieved but that more work is needed to figure out the details. Likewise, alternate approaches to recursion could be a possible area of study. This appears to be particularly relevant

if W-types are introduced as this likely involves essentially arbitrary primitive recursion schemes which would require a more general treatment of the resource usage for recursive terms.

Another option is to instantiate $\lambda_{\mathbb{M}}^{\Sigma\Pi U\mathbb{N}}$ to some specific modality and study its properties in a way similar to our exploration of erasures. Possibilities include well-studied principles such as linear types to more exotic choices from the endless possibilities of modality structures.

Finally, our use of the logical relation for erasures was limited to showing a basic soundness property. In a similar vein to how the logical relation for reducibility was used, not only to show decidability of conversion but also a number of other properties, the logical relation can likely be used to discover further properties of the erasure system. The most immediate direction would be to device soundness properties that are not limited to natural numbers but the logical relation can likely be used in other ways as well.

References

[1] A. Abel and J. Bernardy, "A unified view of modalities in type systems," *Proc. ACM Program. Lang.*, vol. 4, no. ICFP, 90:1–90:28, 2020. DOI: 10.1145/3408972.

[2] C. McBride, "I got plenty o'nuttin'," in *A List of Successes That Can Change the World*, Springer, 2016, pp. 207–233. DOI: 10.1007/978-3-319-30936-1_12.

[3] P. Martin-Löf, *Intuitionistic type theory: Notes by Giovanni Sambin*, ser. Studies in Proof Theory. Bibliopolis, 1984, vol. 1, pp. iv+91, ISBN: 88-7088-105-9.

[4] E. Brady, C. McBride, and J. McKinna, "Inductive families need not store their indices," in *Types for Proofs and Programs*, S. Berardi, M. Coppo, and F. Damiani, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 115–129, ISBN: 978-3-540-24849-1. DOI: 10.1007/978-3-540-24849-1_8.

[5] N. Mishra-Linger and T. Sheard, "Erasure and polymorphism in pure type systems," in *International Conference on Foundations of Software Science and Computational Structures*, Springer, 2008, pp. 350–364. DOI: 10.1007/978-3-540-78499-9_25.

[6] M. Tejiščák, "A dependently typed calculus with pattern matching and erasure inference," *Proc. ACM Program. Lang.*, vol. 4, no. ICFP, 2020-08. DOI: 10.1145/3408973.

[7] AgdaTeam, *The Agda Wiki*, https://wiki.portal.chalmers.se/agda/pmwiki.php, 2021.

[8] A. Abel, J. Öhman, and A. Vezzosi, "Decidability of conversion for type theory in type theory," *Proc. ACM Program. Lang.*, vol. 2, no. POPL, 2017-12. DOI: 10.1145/3158111.

[9] N. G. De Bruijn, "Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem," in *Indagationes Mathematicae (Proceedings)*, North-Holland, vol. 75, 1972, pp. 381–392.

[10] R. Atkey, "Syntax and semantics of quantitative type theory," in *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, A. Dawar and E. Grädel, Eds., ACM, 2018, pp. 56–65. DOI: 10.1145/3209108.3209189.

[11] A. Abel, "Resourceful dependent types," in *Presentation at 24th International Conference on Types for Proofs and Programs (TYPES 2018), Braga, Portugal*, abstract, 2018. [Online]. Available: http://www.cse.chalmers.se/~abela/types18.pdf.

[12] J. Wood and R. Atkey, "A linear algebra approach to linear metatheory," *CoRR*, vol. abs/2005.02247, 2020. arXiv: 2005.02247.

[13] A. Brunel, M. Gaboardi, D. Mazza, and S. Zdancewic, "A core quantitative coeffect calculus," in *Programming Languages and Systems. ESOP 2014*, Z. Shao, Ed., ser. Lecture Notes in Computer Science, vol. 8410, Springer, 2014, pp. 351–370. DOI: 10.1007/978-3-642-54833-8_19.

[14] B. Barras and B. Bernardo, "The implicit calculus of constructions as a programming language with dependent types," in *Foundations of Software Science and Computational Structures*, R. Amadio, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 365–379, ISBN: 978-3-540-78499-9. DOI: 10.1007/978-3-540-78499-9_26.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY