The Syntax and Semantics of Quantitative Type Theory

ROBERT ATKEY, University of Strathclyde

Type Theory offers a tantalising promise: that we can program and reason within a single unified system. However, this promise slips away when we try to produce *efficient* programs. Type Theory offers little control over the intensional aspect of programs: how are computational resources used, and when can they be reused. Tracking resource usage via types has a long history, starting with Girard's Linear Logic and culminating with recent work in contextual effects, coeffects, and quantitative type theories. However, there is conflict with full *dependent* Type Theory when accounting for the difference between usages in types and terms. Recently, McBride has proposed a system that resolves this conflict by treating usage in types as a "zero" usage, so that it doesn't affect the usage in terms. This leads to a simple expressive system, which we have named "Quantitative Type Theory" (QTT).

McBride presented a syntax and typing rules for the system, as well as an erasure property that exploits the difference between "not used" and "used", but does not do anything with the finer usage information. In this paper, we present present a semantic interpretation of a variant of McBride's system, where we fully exploit the usage information. We interpret terms simultaneously as having extensional ("compile-time") content and intensional ("runtime") content. In our example models, extensional content is set-theoretic functions, representing the "compile-time" or "type-level" content of a type-theoretic construction. Intensional content is given by realisers for the extensional content. We use Abramsky et al.'s Linear Combinatory Algebras as realisers, yield a large range of potential models from Geometry of Interaction, graph models, and syntactic models. Read constructively, our models provide a resource sensitive compilation method for QTT.

To rigorously define the structure required for models of QTT, we introduce the concept of a "Quantitative Category with Families", a generalisation of the standard "Category with Families" class of models of Type Theory, and show that this class of models soundly interprets Quantitative Type Theory.

CCS Concepts: • **Software and its engineering → General programming languages**; • **Social and professional topics** → *History of programming languages*;

Additional Key Words and Phrases: keyword1, keyword2, keyword3

ACM Reference format:
Robert Atkey. 2017. The Syntax and Semantics of Quantitative Type Theory. *Proc. ACM Program. Lang.* 1, 1, Article 1 (January 2017), 28 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Dependent Type Theory promises to relax the distinction between "programming" and "verification" by combining both in a single system. Implementations of Type Theory such as Agda [The Agda Team 2017] and Idris [Brady 2013] combine programming and proving into a single system. Indeed, Idris is explicitly intended to be used as a practical programming language as well as a lightweight verification platform. Coq [The Coq development team 2017] is primarily intended as a proof assistant, but also provides a powerful program extraction facility.

However, there are difficulties when attempting to actually use Type Theory as a programming language. Type Theory uses data for two distinct, but overlapping, purposes: *i)* extensionally, as information to be used in the formation of types, for example, the type $\mathsf{Fin}(n)$ of naturals bounded by n depends on the natural number n when type checking, but not a runtime; and *ii)* intensionally,

2017. 2475-1421/2017/1-ART1 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

as computational information that is manipulated by programs. (We use extensional and intensional in this paper to refer to these two different uses of data. Their use in equality in Type Theory is unrelated.) An example is illustrated by the type of the *cons* operation for length indexed vectors of elements of some type S:

$$(::) : (n : \mathsf{nat}) \to S \to \mathsf{vec}\ n\ S \to \mathsf{vec}\ (\mathsf{succ}\ n)\ S$$

A naive implementation of length indexed vectors will store, for every element: a value s; a tail v; *and* a natural number n recording the length of v. If a unary representation of natural numbers is used, then this can easily lead to a runtime representation of vectors that consumes memory space quadratic in the length of the list! The problem of working out what can be erased at runtime has been tackled in different ways. Coq uses a sort Prop of computationally irrelevant propositions that can be erased. Brady et al. [2003] considered alternative implementations of inductive types that take data from their indices instead of storing it. There are Type Theories that explicitly erase data proposed by Pfenning [2001], Miquel [2001], and Mishra-Linger and Sheard [2008], and recently an implementation of erasure inference in Idris[1].

Related to the problem of distinguishing between extensional and intensional use of data is distinguishing between different kinds of intensional use of data. Such information can be used to generate more efficient code, or to ensure that programs only use computational resources in restricted ways (allowing, for example, in place update of memory). Girard [1987]'s Linear Logic initiated a body of research into type systems that track how programs use their input. Initially, this recorded zero, single, or multiple uses (made explicit by Mogensen [1997]), but recent work in coeffects and quantitative types by Petricek et al. [2014], Brunel et al. [2014], and Ghica and Smith [2014] refined this to track usage via semirings. However, extending these systems to *dependent* Type Theory is not straightforward due to the conflict between extensional and intensional uses.

McBride [2016] has recently overcome this conflict by combining the work on erasability and quantitative types to produce a system that smoothly allows an expressive type theory that tracks refined information about variable usage. His insight is to use the 0 of the semiring to represent information that is erased at runtime, but is *still available* for use in types (i.e., extensionally). McBride presented a syntax and typing rules for the system, as well as an erasure property that exploits the difference between "not used" and "used", but does not do anything with the finer usage information available. In this paper, we present present semantic interpretations of a variant of McBride's system that fully exploits the usage information.

Contributions.

(1) In Section 2.1 and Section 3, we provide a reformulation of McBride's system as a declarative Martin-Löf style Type Theory with dependent function types, booleans, and a universe of small types. We name this family of systems "Quantitative Type Theory" (QTT). We also fix a bug in McBride's original system that prevented substitution being admissible (Section 3.2.4).
(2) In Section 4, we define *Quantitative Categories with Families* (QCwFs), a sound class of models for QTT. This codifies the structure of models of QTT independently of the syntax, and therefore simplifying the creation of new concrete models.
(3) In Section 2.2 and Section 6, we provide concrete realisability models of QTT, where the realisers are taken from Linear Combinatory Algebras (LCAs). LCAs are a flexible and general model of resource sensitive computation, originally arising in Abramsky et al. [2002]'s analysis of Girard's Geometry of Interaction. Thus, the usage information recorded in QTT typing derivations is exploited to construct resource sensitive realisations of QTT terms.

[1] http://idris.readthedocs.io/en/latest/reference/erasure.html

2 QUANTITATIVE TYPE THEORY, SYNTAX AND SEMANTICS

In this long introductory section, we describe the syntax and semantics of Quantitative Type Theory at a high level, providing pointers into the full technical development in later sections.

2.1 Type Theory with Usage Accounting

Combining type dependency with linear typing is not straightforward. In this section, we motivate McBride's system and present the formulation of it that we will use in this paper.

2.1.1 Dependency and Accountancy. In Martin-Löf Type Theory, the term judgement has the following stereotypical form:

$$x_1 : S_1, x_2 : S_2, \ldots, x_n : S_n \vdash M : T$$

From Type Theory's mixed computational/logical point of view, the context $x_1 : S_1, x_2 : S_2, \ldots, x_n : S_n$ has two uses. Computationally, it describes the names used to access resources which *may* be used in the term M to construct a value of type T. Logically, it describes the names used to refer to the extensional meanings used to form the types $S_2, \ldots, S_n$ and T.

For example, consider the judgement:

$$n : \mathsf{Nat}, x : \mathsf{Fin}(n) \vdash x : \mathsf{Fin}(n) \tag{1}$$

where Nat is the type of natural numbers, and $\mathsf{Fin}(n)$ is the type of numbers less than n. The variables n and x play two different roles: x is used as a reference to some computational data that is transferred from the input to the output; n is used logically to define the type of x. The fact that n is not used computationally is not explicitly recorded.

Linear Logic [Girard 1987] uses the presence or absence of variables in a context to explicitly record which variables are used computationally. An intuitionistic linear typing judgement,

$$x_1 : X_1, \ldots, x_n : X_n \vdash M : Z$$

indicates that the term M *must* use the computational resources named by $x_1, \ldots, x_n$ each precisely once. This resource sensitive reading enables a more refined analysis of the computational processes that M can describe, allowing for implementations of M that make fewer demands on their runtime environments. For example, not requiring a garbage collector [Baker 1992], or not preserving old versions of data structures [Wadler 1990].

A conflict arises when we want to combine Linear Logic's resource accounting via variables with type dependency. Linear typing uses the presence or absence of a variable to indicate whether or not it is used by the term. If a variable is not used in a term, it does not appear in the context, and so it is not available for use in a type. Thus, naively considered, Judgement 1 is not linear: both because n is not used in the term, and because it is used twice in types.

In Linear Logic, the reading of presence a variables in context as a intensional or "computational" use takes primacy, and this conflicts with Type Theory's reading as a variable as a reference to an extensional or "contemplative" piece of information to be used in when forming types.

To resolve this conflict, several authors have used formulations of Linear Logic that distinguish between unrestricted ("intuitionistic") variables and restricted ("linear") variables. In the simply typed setting, this originates in the work of Barber [1996], where judgements have two contexts, here separated by a vertical bar:

$$x_1 : S_1, \ldots, x_m : S_m \mid y_1 : X_1, \ldots, y_n : X_n \vdash M : Y$$

The variables $x_1, \ldots$ may be used without restriction, zero, one, or many times. The variables $y_1, \ldots$ must be used exactly once. Thus, the judgement tracks information about the two different kinds of usage. Cervesato and Pfenning [2002], and later Krishnaswami et al. [2015] and Vákár

[2015] adapted this idea to dependent types. Exploiting the fact that variables in the unrestricted context act as they do in normal Type Theory, there is no problem in allowing types to depend on the variables $x_1, \ldots$. Thus, assuming that we wish to treat references to elements of $\mathsf{Fin}(n)$ linearly, our Judgement 1 can be reformulated as:

$$n : \mathsf{Nat} \mid x : \mathsf{Fin}(n) \vdash x : \mathsf{Fin}(n)$$

The variables in the unrestricted context now inherit the ambiguous status of variables from standard Type Theory. They can be used purely contemplatively in types and they can be used for computation. The difference is not recorded in the typing judgement. A further restriction is that types cannot depend on linear variables. There is no way that we can contemplate the extensional content of a linear resource for the purposes of typing. Thus, we cannot form a judgement such as:

$$n : \mathsf{Nat} \mid x : \mathsf{Fin}(n) \vdash (x, \mathsf{refl}(x)) : (y : \mathsf{Fin}(n)) \times (x \equiv y)$$

where we are trying to state that the value we return has the same extensional content as the input by pairing the computational representation of the value with proof of equality.

Despite allowing some type dependency, the systems that rely on split contexts still adhere to Linear Logic's discipline of using presence or absence in a context to track how a variable is used. As we have seen, this interferes with type dependency. It also restricts what kinds of "use" we can express: either a variable is used or it is not. To lift this latter restriction, systems that annotate variables of the context with information about how they are used have been proposed by Brunel et al. [2014], Ghica and Smith [2014], and Petricek et al. [2014] (the general idea of marking variable bindings rather than their types was originally called "discharged assumptions" by Terui [2001]). In all these systems, information about how variables are used is recorded using a commutative semiring. Semirings are a natural structure to use: addition is used to sum up multiple uses of a variable, and multiplication is used to account for nested use. In the notation we will use below, a stereotypical judgement in these systems is

$$x_1 \overset{\rho_1}{:} S_1, \ldots, x_n \overset{\rho_n}{:} S_n \vdash M : T \tag{2}$$

where the $\rho_1, \ldots, \rho_n$ are elements of the semiring indicating how the corresponding variable is used. In these systems, the zero of the semiring is used to indicate that a variable is not used at all. Therefore, $x \overset{0}{:} S$ amounts to a complicated way of stating that x might as well not be present.

However, when we move to dependent types, 0 usage variables have a useful meaning. McBride [2016] reads the usage annotations $\rho_1, \ldots, \rho_n$ as purely indications of computational usage, so a variable with usage 0 can still be used in the formation of types. The properties of semrings means that 0 usage is ideal for tracking use in types: we always have $0 + \rho = \rho$, so combining a computational use with a use in a type retains the original usage; and $0\rho = 0$, so nesting an apparently computational use within a type treats the whole usage as noncomputational.

Term typing judgements now have the form:

$$x_1 \overset{\rho_1}{:} S_1, \ldots, x_n \overset{\rho_n}{:} S_n \vdash M \overset{\sigma}{:} T \tag{3}$$

where the difference from Judgement 2 is that the output is annotated with a usage σ, where σ is restricted to either be the 0 or the 1 of the semiring. This additional annotation means that we can construct terms that are to be only used in types. McBride allowed arbitrary usages ρ on the final colon. However, this leads to a system that does not admit substitution as we show in Section 3.2.4.

In the following sections we introduce our formulation of McBride's system, extending it with a type of booleans. McBride presented a bidirectional system suited for implementation, and did not distinguish between proper types and terms representing types. In this paper, we are interested in a semantical investigation of the theory and so switch to a system with an explicit separation of types

$+$	0	1	ω
0	0	1	ω
1	1	ω	ω
ω	ω	ω	ω

$\cdot$	0	1	ω
0	0	0	0
1	0	1	ω
ω	0	ω	ω

Fig. 1. Addition and Multiplication tables for the $\{0, 1, \omega\}$ semiring

and representations of types, and with a "declarative" presentation of the rules. Our presentation is laid out as follows: in Section 2.1.2 we fix our notation for semirings; in Section 2.1.3 we describe the presyntax of the system; in Section 2.1.4 we detail the judgement forms; and in Section 2.1.5, Section 2.1.6, and Section 2.1.7, we describe how depdendent function types, booleans, and universes are formulated in a type theory with usage annotations.

2.1.2 Commutative Semirings for Resource Annotation. A *commutative semiring* R is a structure consisting of a carrier set R, binary operations $(+) : R \times R \to R$, $(\cdot) : R \times R \to R$ and constants $1, 0 \in R$, satisfying the following equations:

$$\rho + (\pi + \epsilon) = (\rho + \pi) + \epsilon \qquad \rho + 0 = \rho \qquad \rho + \pi = \pi + \rho \qquad \rho(\pi\epsilon) = (\rho\pi)\epsilon \qquad \rho 1 = \rho$$

$$\rho\pi = \pi\rho \qquad \rho(\pi + \epsilon) = \rho\pi + \rho\epsilon \qquad \rho 0 = 0$$

McBride's leading example of a semiring for tracking usage has the carrier set $\{0, 1, \omega\}$, indicating zero, singleton, and multiple usage respectively. The latter ω usage is analogous to the unrestricted exponential ! in Linear Logic. The addition and multiplication tables for this semiring are in Figure 1.

Other useful semirings include: *i)* the $\{\mathsf{erased}, \mathsf{present}\}$ semiring, which tracks which data is erased at runtime; *ii)* the natural number semiring $\mathbb{N}$, which tracks number of uses precisely; and *iii)* the tropical semiring with carrier $\mathbb{N} \cup \{\infty\}$, where "addition" is the min operation, and multiplication is addition extended with $\infty + x = \infty$. This latter semiring could be used to model staged computation, where ∞ is the stage where types live.

2.1.3 Presyntax. Quantitative Type Theory (QTT) is defined over the following syntactic categories of usages, ρ, π, precontexts Γ, pretypes S, T, R, and preterms M, N, O:

Usages	ρ, π	$\in$	R
Precontexts	Γ	$::=$	$\diamond \mid \Gamma, x \overset{\rho}{:} S$
Pretypes	S, T, U	$::=$	$(x \overset{\pi}{:} S) \to T \mid \mathsf{Bool} \mid \mathsf{El}(M) \mid \mathsf{Set}$
Preterms	M, N, O	$::=$	$x \mid \lambda x \overset{\pi}{:} S.M^T \mid \mathsf{App}_{(x \overset{\pi}{:} S)T}(M, N)$
			$\mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{ElimBool}_{(z)T}(M_t, M_f, N)$
			$\mid \mathsf{Bool} \mid (x \overset{\pi}{:} M) \to N$

In the following sections, we describe the typing rules that identify the contexts, types, and terms from the precontexts, pretypes, and preterms. The symbol $\diamond$ denotes the empty precontext, which we omit when writing precontexts with more than zero elements. Readers experienced with type theory may note that these preterms contain more type information than is usually necessary (e.g., the result type T in a λ-abstraction). This will be used when we interpret the syntax in our categorical models in Section 5.

Precontexts contain usage annotations, ρ, on each of the constituent variables. We manipulate these annotations by scaling and addition operations. Scaling of a precontext, $\pi\Gamma$ is defined by

recursion on the structure of Γ:

$$\begin{array}{lcl} \pi(\diamond) & = & \diamond \\ \pi(\Gamma, x \overset{\rho}{:} S) & = & \pi\Gamma, x \overset{\pi\rho}{:} S \end{array}$$

Note that scaling is shallow in the sense that usage annotations in pretypes S, if any, are not affected. By the semiring laws, context zero-ing, 0Γ, sets all the annotations in the context to 0. For any precontext Γ, we use 0Γ as the canonical representation of the extensional content of that context.

Precontext addition $\Gamma_1 + \Gamma_2$ is only defined when $0\Gamma_1 = 0\Gamma_2$, and is also defined by recursion:

$$\begin{array}{lcl} \diamond + \diamond & = & \diamond \\ (\Gamma_1, x \overset{\rho_1}{:} S) + (\Gamma_2, x \overset{\rho_2}{:} S) & = & (\Gamma_1 + \Gamma_2), x \overset{\rho_1+\rho_2}{:} S \end{array}$$

These cases are exhaustive by the requirement that $0\Gamma_1 = 0\Gamma_2$.

2.1.4 Contexts, Types, Terms. Contexts are identified within the precontexts by the judgement $\Gamma \vdash$, defined by the following rules:

$$\frac{}{\diamond \vdash}\ \textsc{Ctxt-Emp} \qquad \frac{\Gamma \vdash \qquad 0\Gamma \vdash S}{\Gamma, x \overset{\rho}{:} S \vdash}\ \textsc{Ctxt-Ext}$$

where the judgement $0\Gamma \vdash S$ indicates that S is well formed as a type in the context 0Γ. We will see the type formation rules for each type former below. The rule Ctxt-Emp builds the empty context. The rule Ctxt-Ext extends a context Γ with a extra variable x of type S, with usage annotation ρ. The annotation ρ on the typing colon indicates that we are building a context representing environments that provide for ρ-"many" uses of x.

As we shall see below, all type formation rules yield judgements where all the usage annotations in Γ are equal to 0. Thus type formation requires only extensional information.

As indicated in Section 2.1.1 (Judgement 3) term judgements in QTT have the following form:

$$\Gamma \vdash M \overset{\sigma}{:} S$$

where $\sigma \in \{0, 1\}$. The binary choice of σ effectively splits the theory into two halves. When $\sigma = 0$, we are indicating that we are constructing a piece of extensional information that will have no computational content. By Lemma 3.3 (below) when $\sigma = 0$ we will necessarily have that all the usage annotations in the context are 0 too: we cannot use computational ("runtime") information to construct extensional information. When $\sigma = 1$, we are indicating we are constructing some computationally relevant data.

The two basic rules for terms are for variables and conversion between equal types:

$$\frac{\vdash 0\Gamma, x \overset{\sigma}{:} S, 0\Gamma'}{0\Gamma, x \overset{\sigma}{:} S, 0\Gamma' \vdash x \overset{\sigma}{:} S}\ \textsc{Tm-Var} \qquad \frac{\Gamma \vdash M \overset{\sigma}{:} S \qquad 0\Gamma \vdash S \equiv T}{\Gamma \vdash M \overset{\sigma}{:} T}\ \textsc{Tm-Conv}$$

The variable rule, Tm-Var, selects an individual variable from the context. In keeping with our intuition for usage annotations, all variables that are not selected are marked with 0 usage. The conversion rule, Tm-Conv, is almost identical to that in standard MLTT, except that the type equality judgement $0\Gamma \vdash S \equiv T$ explicitly stipulates that types are judged equal in a context with no intensional resources.

2.1.5 The Dependent Function Type. Dependent function types $(x \overset{\pi}{:} S) \to T$ in QTT record how the function will use its argument via the π annotation. The formation rule is:

$$\frac{0\Gamma \vdash S \qquad 0\Gamma, x \overset{0}{:} S \vdash T}{0\Gamma \vdash (x \overset{\pi}{:} S) \to T}\ \textsc{Ty-Pi}$$

The usage annotation π is not used when judging that T is a type. As for all type well formedness judgements, it is judged in a context of 0 usage. The usage annotation π is used in the introduction and elimination rules of this type to track how the abstracted variable x is used, and how to multiply the resources required for the argument, respectively.

$$\frac{\Gamma, x \overset{\sigma\pi}{:} S \vdash M \overset{\sigma}{:} T}{\Gamma \vdash \lambda x \overset{\pi}{:} S.M^T \overset{\sigma}{:} (x \overset{\pi}{:} S) \to T}\ \textsc{Tm-Lam}$$

$$\frac{\Gamma_1 \vdash M \overset{\sigma}{:} (x \overset{\pi}{:} S) \to T \qquad \Gamma_2 \vdash N \overset{\sigma}{:} S \qquad 0\Gamma_1 = 0\Gamma_2}{\Gamma_1 + \pi\Gamma_2 \vdash \mathrm{App}_{(x \overset{\pi}{:} S)T}(M, N) \overset{\sigma}{:} T[N/x]}\ \textsc{Tm-App}$$

Forgetting the resource annotations, these are the standard introduction and elimination rules for dependent function types (remembering that the side condition $0\Gamma_1 = 0\Gamma_2$ means that Γ_1 and Γ_2 have the same types for the same variables). In the introduction rule, we require that the abstracted variable x has usage $\sigma\pi$ – multiplication by σ is used to enforce the zero-needs-nothing property of the system. In the elimination rule, we sum up the usage requirements of the function and its argument, scaling the argument's requirements by the amount required by the function itself.

2.1.6 The Boolean Type. The type Bool is an example of a type of data that can be available at runtime. The rule for Bool states that it is a type in any environment with no resources:

$$\frac{0\Gamma \vdash}{0\Gamma \vdash \mathrm{Bool}}\ \textsc{Ty-Bool}$$

The introduction rules for booleans:

$$\frac{0\Gamma \vdash}{0\Gamma \vdash \mathrm{true} \overset{\sigma}{:} \mathrm{Bool}}\ \textsc{Tm-Bool-True} \qquad \frac{0\Gamma \vdash}{0\Gamma \vdash \mathrm{false} \overset{\sigma}{:} \mathrm{Bool}}\ \textsc{Tm-Bool-False}$$

indicate two pieces of intensional information. First, the 0 usage on the context Γ indicates that no resources are required to construct the constants true and false. Second, the result usage annotation σ is unrestricted, indicating that we are free to choose whether this is a boolean value that is purely extensional, or one that can be used computationally.

The elimination rule for booleans is:

$$\frac{0\Gamma_1, z \overset{0}{:} \mathrm{Bool} \vdash T \qquad \Gamma_1 \vdash M_t \overset{\sigma}{:} T[\mathrm{true}/z] \qquad \Gamma_1 \vdash M_f \overset{\sigma}{:} T[\mathrm{false}/z] \qquad \Gamma_2 \vdash N \overset{\sigma}{:} \mathrm{Bool} \qquad 0\Gamma_1 = 0\Gamma_2}{\Gamma_1 + \Gamma_2 \vdash \mathrm{ElimBool}_{(z)T}(M_t, M_f, N) \overset{\sigma}{:} T[N/z]}\ \textsc{Tm-Bool-Elim}$$

We ensure that the result type T is wellformed, with 0 usage. Then the two branches M_t and M_f for the true and false case respectively must be typed with the same usage annotations Γ_1. This follows the same style as for the "additive" connectives in Linear Logic. The actual boolean to be eliminated must be constructed from other resources Γ_2, but this must have the same extensional content ($0\Gamma_1 = 0\Gamma_2$).

2.1.7 Universe: The Type of Small Types. The type of small types, Set, is an example of a type whose elements have no useful computational presence, but do have an interesting extensional interpretation. Extensionally, they represent small types (here, booleans and functions with small domain and codomain). Intensionally, no type checking or type driven computation is performed at runtime, so inhabitants of Set do not need to be represented.

The type formation rule for Set is similar to the one for Bool:

$$\frac{0\Gamma \vdash}{0\Gamma \vdash \mathsf{Set}}\ \textsc{Ty-Set}$$

The introduction rules for small sets are only available in the subsystem where $\sigma = 0$, indicating that we can never construct a Set value with runtime presence.

$$\frac{0\Gamma \vdash}{0\Gamma \vdash \mathsf{Bool} \overset{0}{:} \mathsf{Set}}\ \textsc{Tm-Set-Bool} \qquad \frac{0\Gamma \vdash M \overset{0}{:} \mathsf{Set} \qquad 0\Gamma, x \overset{0}{:} \mathsf{El}(M) \vdash N \overset{0}{:} \mathsf{Set}}{0\Gamma \vdash (x \overset{\pi}{:} M) \to N \overset{0}{:} \mathsf{Set}}\ \textsc{Tm-Set-Pi}$$

Elements of Set are turned into proper types via the "decoder" $\mathsf{El}(-)$:

$$\frac{0\Gamma \vdash M \overset{0}{:} \mathsf{Set}}{0\Gamma \vdash \mathsf{El}(M)}\ \textsc{Ty-El}$$

This rules indicates that the use of a code for a type in a type does not require any computational information, as we might expect.

It would also be possible to consider a system where elements of Set do have a computational presence, and can be pattern matched on at runtime. In QTT this would be indicated by altering the introduction rules to allow $\sigma = 1$, as well as adding the required elimination constructs.

2.2 Models via Linear Combinatory Algebras

In our description of QTT, we have distinguished between the extensional and intensional (or "computational") content of terms. We make this distinction formal by constructing realisability models of QTT. Computational content will be represented in the model by realisers from a model of linear computation. Our chosen model of linear computation is Abramsky et al. [2002]'s *Linear Combinatory Algebras* (LCAs).

2.2.1 Linear Combinatory Algebras. A *Linear Combinatory Algebra* (LCA) consists of a set $\mathcal{A}$, a binary operation $\cdot : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ (called *application*, and written left associatively), a unary operation $! : \mathcal{A} \to \mathcal{A}$ and eight distinguished elements of $\mathcal{A}$: $B, C, I, K, W, D, \delta, F$, satisfying the following equations:

$$\begin{array}{lcllcl} & & & K \cdot x \cdot !y & = & x \\ B \cdot x \cdot y \cdot z & = & x \cdot (y \cdot z) & W \cdot x \cdot !y & = & x \cdot !y \cdot !y \\ C \cdot x \cdot y \cdot z & = & x \cdot z \cdot y & D \cdot !x & = & x \\ I \cdot x & = & x & \delta \cdot !x & = & !!x \\ & & & F \cdot !x \cdot !y & = & !(x \cdot y) \end{array}$$

LCAs are the untyped counterpart of a Hilbert style axiomatisation of the $(\multimap, !)$ fragment of Linear Logic, analogous to the S and K combinators for minimal implicational logic. We think of the elements of $\mathcal{A}$ as "computational widgets" that can be applied to one another via application. This model is more refined than the SK model in that each widget is assumed to be *linear* in that application only uses its argument once. Since this would be very restrictive, for each widget x, we assume that there is another widget $!x$ that allows multiple uses. The B, C, I combinators

form a basis for linear implication, $\multimap$, allowing composition (B), exchange (C), and identity (I). The remaining combinators handle the exponential !, allowing discarding of !'d resources (K), duplication of such (W), dereliction (D), duplication (δ), and functoriality (F). Importantly, LCAs are *combinatorily complete*: given an expression M with variables and the LCA combinators that contains a variable x exactly once and not under a !, there is an expression $\lambda^* x.M$ that does not contain x, such that $(\lambda^* x.M) \cdot N = M[N/x]$. Likewise, for arbitrary occurences of x, there is an expression $\lambda^! x.M$ which does not contain x, such that $(\lambda^! x.M) \cdot !N = M[N/x]$. The combinatory completeness of LCAs was originally observed by Simpson [2005].

Abramsky et al. [2002]'s motivation for LCAs was their derivation from *Geometry of Interaction* GoI situations, arising from Girard's interpretation of Linear Logic proofs as operators on a Hilbert space [Girard 1989]. We sketch LCAs arising from GoI situations in Section 6.2, as well as LCAs built from graph models of untyped computation, and the syntax of the untyped linear λ-calculus. For now, we assume we have a fixed LCA $\mathcal{A}$, and that we are modelling the version of the system with the $\{0, 1, \omega\}$ semiring. We now sketch the interpretation of QTT using an LCA. The remaining details of the interpretation will be presented in Section 6.1, where we will able to structure our presentation as a Quantitative Category with Families (Section 4).

2.2.2 Modelling Contexts as Assemblies. The essence of our models will be the pairing of extensional information with intensional, or computational, information. For modelling contexts and simultaneous substitutions between them, we will make use of the standard notion of the category of *Assemblies* over a model of computation (see, e.g., Longley and Normann [2015], §3.3.6). Hoshino [2007] has already investigated assemblies over LCAs in the simply typed setting. In this paper we extend this analysis to dependent types. An assembly Γ is a pair of a set $|\Gamma|$ of extensional meanings, and a binary relation $\models_\Gamma \subseteq \mathcal{A} \times |\Gamma|$ such that for all $\gamma \in \Gamma$, there is at least one $a \in \mathcal{A}$ such that $a \models_\Gamma \gamma$. The idea is that the relation $a \models_\Gamma \gamma$ indicates that the LCA element a is an implementation of the extensional meaning γ. A morphism between assemblies $(|\Gamma|, \models_\Gamma)$ and $(|\Gamma'|, \models_{\Gamma'})$ is a function $f : |\Gamma| \to |\Gamma'|$ that is realisable: there exists an $a_f \in \mathcal{A}$ such that for all $\gamma \in |\Gamma|$ and a_γ, if $a_\gamma \models_\Gamma \gamma$, then $a_f \cdot a_\gamma \models_{\Gamma'} f(\gamma)$. These definitions yield a category, using the I of the LCA to realise identities, and B to realise composition.

The requirement that every extensional meaning $\gamma \in |\Gamma|$ have a realiser may seem odd, considering that we have stated above that we wish to allow for zero usage members of contexts. To see how this is resolved, we look at the interpretation of a context of the form:

$$x_1 \overset{0}{:} S_1, x_2 \overset{1}{:} S_2, x_3 \overset{\omega}{:} S_3$$

Contexts are interpreted as left nested stacks of pairs. The extensional meaning of an inhabitant of this context will have the form $(((*, s_1), s_2), s_3)$, where s_1, s_2, s_3 are members of the extensional interpretations of the types S_1, S_2, S_3, respectively.

Computationally, contexts are also represented as left nested stacks of pairs. In an LCA, pairs $[a, b]$ can be represented by their Church-encoding $\lambda^* p.p \cdot a \cdot b$. (This was used by Hoshino [2007] to construct symmetric monoidal structure on the category of assemblies over an LCA.) The empty pair is represented as the identity combinator I. A typical realiser for the extensional value $(((*, s_1), s_2), s_3)$ will look like:

$$[[[I, I], a_2], !a_3]$$

The empty context has been realised by the identity combinator I; the zero use variable x_1 has also been realised by I; the single use variable x_2 has been realised by some combinator a_2; and the multiple use variable x_3 has been realised by some !'d combinator a_3. The use of I means that we can discard the first element of the context at runtime, and the use of ! means that we can duplicate the third element as needed via the W combinator.

2.2.3 Modelling Types as Families of Assemblies. The formation of types has no computational content, hence no realisers, but types do describe how their own extensional inhabitants are realised. Thus we interpret types as families of assemblies indexed over a set, not an assembly. Given a set Δ of extensional interpretations of the context, we model a type in the context Δ as an assembly $(|S(\delta)|, \Vdash_{S(\delta)})$ for each $\delta \in \Delta$.

2.2.4 Modelling Terms. The interpretation of a term judgement $\Gamma \vdash M \overset{\sigma}{:} T$ depends on the value of σ. When $\sigma = 0$, we only use the extensional component of the context Γ and type T, and model M as a set theoretic function $M : \forall \gamma \in |\Gamma|.\ |S(\gamma)|$. In cases that we need a realiser for M, we can realise it as the identity combinator I, following our realising of erased members of the context by I (the actual interpretation is a little more complex due to the difference between I and pairings of Is, but this is the essential content).

When $\sigma = 1$, we again require an extensional interpretation $M : \forall \gamma \in |\Gamma|.\ |S(\gamma)|$, but now stipulate that it must be realised by some element of the LCA, in the same way that substitutions between contexts are realised. Thus, terms intended to have intensional content in QTT are assigned such intensional content represented as elements of the LCA $\mathcal{A}$.

2.3 Codifying the Structure of Models: Quantitative Categories with Families

To connect the syntax and the LCA-realisability models, we codify the structure of models of QTT in a structure we have named *Quantitative Categories with Families* (QCwFs). Our definition extends the standard *Categories with Families* (CwF) models of Type Theory [Dybjer 1996; Hofmann 1997] with the intensional information in the usage annotations. We present the definition of QCwF in full in Section 4. Here, we sketch QCwFs, and relate them to the realisability models over LCAs.

The essence of a QCwF is to have two categories $\mathcal{C}$ and $\mathcal{L}$, where $\mathcal{C}$ consists of extensional interpretations, and $\mathcal{L}$ consists of extensional interpretations augmented with intensional information. These categories are related by a pair of functors:

$$\mathcal{C} \underset{J}{\overset{U}{\rightleftarrows}} \mathcal{L}$$

such that $U \circ J = \mathrm{Id}$. The functor $U : \mathcal{L} \to \mathcal{C}$ forgets the intensional information, and $J : \mathcal{C} \to \mathcal{L}$ augments an extensional interpretation with trivial intensional information. We require that the functor U is faithful: equality only depends on extensional information. In the LCA-realisability model, the category $\mathcal{L}$ is the category of assemblies described above, and $\mathcal{S}$ is the category of sets and functions. The functor U forgets the realisability information, and J augments a set Δ with the information that I is the sole realiser for every extensional element.

QCwFs require that the category $\mathcal{C}$ have the structure of a Category with Families for interpreting the extensional (0-use) subsystem of QTT: for each each $\Delta \in \mathcal{C}$ there is a set of semantic types $\mathrm{Ty}(\Delta)$; for each Δ and $S \in \mathrm{Ty}(\Delta)$ there is a set of semantic terms $\mathrm{Tm}(\Delta, S)$; and for each Δ and $S \in \mathrm{Ty}(\Delta)$ there is an object $\Delta.S \in \mathcal{C}$, representing context extension, with comprehension structure. We recall the full definition of a CwF in Section 4.1. In the LCA-realisability model, we do not use the standard CwF structure on the category of Sets because we also require that the interpretation of types includes realisability information. Therefore $\mathrm{Ty}(\Delta)$ consists of Δ-indexed families of assemblies, as described above (Section 2.2.3). The semantic terms at the extensional level, however, ignore the realisability information.

The remaining structure in a QCwF refines the structure of the CwF on $\mathcal{C}$ with intensional information. To model the syntactic operations of scaling and addition of contexts (Section 2.1.3), we will require functors $\pi(-) : \mathcal{L} \to \mathcal{L}$ for each π in the semiring, and a functor $(+) : \mathcal{L} \times_{\mathcal{C}} \mathcal{L} \to \mathcal{L}$.

The category $\mathcal{L} \times_C \mathcal{L}$ is the category of pairs of objects of $\mathcal{L}$ that map to the same object in $\mathcal{C}$ via U (i.e., the pullback of U along itself). The restricted domain of semantic context addition models the restriction $0\Gamma_1 = 0\Gamma_2$ on precontext addition. In the LCA-realisability model, scaling will be interpreted as setting all the realisers to be I, doing nothing, or using !, for scaling by $0, 1, \omega$, respectively. Addition of contexts will be interpreted by pairing.

Since type formation does not require any intensional information, there is no collection of types specifically for every object $\Gamma \in \mathcal{L}$. Instead, we use the types of the CwF structure on $\mathcal{C}$, via U: $\mathrm{Ty}(U\Gamma)$. Context extension with usage annotations now takes $\Gamma \in \mathcal{L}$, $S \in \mathrm{Ty}(U\Gamma)$, and a semiring element ρ to bulid a new object $\Gamma.\rho S \in \mathcal{L}$. To reflect the essential extensional content of context extension, this will satisfy $U(\Gamma.\rho S) = U\Gamma.S$.

To model term judgements, we require a collection of semantic intensional terms $\mathrm{Tm}^{\mathcal{L}}(\Gamma, \rho S)$ for every $\Gamma \in \mathcal{L}$, ρ a semiring element, and $S \in \mathrm{Ty}(U\Gamma)$. Unlike in the syntax, we allow arbitrary usages ρ of semantic terms. This makes the interpretation of substitution easier. As for contexts, we require that there is a forgetful operation $U : \mathrm{Tm}^{\mathcal{L}}(\Gamma, \rho S) \to \mathrm{Tm}(U\Gamma, S)$ that forgets the intensional information associated with terms (i.e., forgets the existence of a realiser in the LCA model).

Finally, there are usage annotated variants of the comprehension structure that operate on intensional terms and intensional contexts, tracked by the corresponding structure in $\mathcal{C}$. We detail this structure in our full definition in Definition 4.2.

There are two immediate results relating CwFs and QCwFs. First, by construction, every QCwF contains a CwF: the extensional content of the QCwF. Secondly, every CwF trivially gives a QCwF, with U as the identity functor, which completely ignores the usage annotations. This will be Proposition 4.3, and guarantees that we at least have consistent models for QTT.

2.4 The Remainder of the Paper

In the next section, we complete our description of QTT by giving the remaining type and term equality rules, and some metatheoretical results. In Section 4, we begin our semantical investigation of QTT by giving the definition of Quantitative Categories with Families, and stating some basic results about them. We show how to to interpret QTT in a QCwF in Section 5, stating the soundness result for our interpretation. In Section 6, we fill in the remaining details of LCA-realisability model of QTT, and discuss further instances of QCwFs for interpreting QTT that have different notions of computational content of terms. Section 7 discusses related work. Finally, Section 8 concludes with a discussion of the contributions of this paper, and ideas for future development of this work.

3 METATHEORY OF QUANTITATIVE TYPE THEORY

Section 2.1 has already introduced the context, type, and term judgements and rules for QTT. We now complete the presentation by giving the type and term equality rules, and state the admissibility of reresourcing, weakening, and substitution rules.

3.1 Type and Term Judgemental Equality

We gave all of the context, type, and term rules in Section 2.1. The remaining rules of QTT concern the type and term equality judgements $\Gamma \vdash S \equiv T$ and $\Gamma \vdash M \equiv N \overset{\sigma}{:} S$. The rules for the former are given in Figure 2, and the latter in Figure 3. The type equality rules are standard, stipulating that equality is an equivalence relation and a congruence. The final two rules define the behaviour of the universe decoder $\mathsf{El}(-)$ on the codes for boolean and dependent function types. As for the type judgement $\Gamma \vdash S$, the type equality rules maintain the invariant that the context always has the 0 usage annotation on each member.

$$\frac{0\Gamma \vdash S}{0\Gamma \vdash S \equiv S}\ \textsc{Ty-Eq-Refl} \qquad \frac{0\Gamma \vdash S \equiv T}{0\Gamma \vdash T \equiv S}\ \textsc{Ty-Eq-Symm} \qquad \frac{0\Gamma \vdash S \equiv T \qquad 0\Gamma \vdash T \equiv U}{0\Gamma \vdash S \equiv U}\ \textsc{Ty-Eq-Tran}$$

$$\frac{0\Gamma \vdash S \equiv S' \qquad 0\Gamma, x \overset{0}{:} S \vdash T \equiv T'}{0\Gamma \vdash (x \overset{\rho}{:} S) \to T \equiv (x \overset{\rho}{:} S') \to T'}\ \textsc{Ty-Eq-Pi-Cong} \qquad \frac{0\Gamma \vdash M \equiv N \overset{0}{:} \mathsf{Set}}{0\Gamma \vdash \mathsf{El}(M) \equiv \mathsf{El}(N)}\ \textsc{Ty-Eq-El-Cong}$$

$$\frac{0\Gamma \vdash M \overset{0}{:} \mathsf{Set} \qquad 0\Gamma, x \overset{0}{:} \mathsf{El}(M) \vdash N \overset{0}{:} \mathsf{Set}}{0\Gamma \vdash \mathsf{El}((x \overset{\pi}{:} M) \to N) \equiv (x \overset{\pi}{:} \mathsf{El}(M)) \to \mathsf{El}(N)}\ \textsc{Ty-Eq-El-Pi}$$

$$\frac{0\Gamma \vdash}{0\Gamma \vdash \mathsf{El}(\mathsf{Bool}) \equiv \mathsf{Bool}}\ \textsc{Ty-Eq-El-Bool}$$

Fig. 2. Type Equality $\Gamma \vdash S \equiv T$

Term equality always relates terms at a type and a usage σ. The usage annotation is required to maintain the invariant that if $\Gamma \vdash M \equiv N \overset{\sigma}{:} S$, then $\Gamma \vdash M \overset{\sigma}{:} S$ and likewise for N, However, by Lemma 3.2, this additional annotation does not affect the equality of terms. As for types, the term equality rules are standard: term equality is an equivalence relation and a congruence, and includes $\beta\eta$ for terms of function type and β for boolean elimination.

3.2 Admissible Rules

QTT enjoys weakening and substitution as admissible rules, as long as the usage annotations are correctly maintained. QTT also admits rules for manipulation of usage annotations.

3.2.1 Usage annotation Manipulation. As described in the previous section, usage annotations do not affect typing. Therefore, they do not affect the identification of contexts within the precontexts:

Lemma 3.1. *Usage annotations do not affect context well formedness:*

$$\frac{\Gamma_1 \vdash \qquad 0\Gamma_1 = 0\Gamma_2}{\Gamma_2 \vdash}$$

We can take a term or term equality and "extract" its extensional component by zeroing:

Lemma 3.2. *The following* 0*-ing rule is admissible:*

$$\frac{\Gamma \vdash M \overset{\sigma}{:} S}{0\Gamma \vdash M \overset{0}{:} S}\ \textit{Tm-Zero} \qquad \frac{\Gamma \vdash M \equiv N \overset{\rho}{:} S}{0\Gamma \vdash M \equiv N \overset{0}{:} S}\ \textit{Tm-Eq-Zero}$$

The following lemma describes how knowledge of the usage annotation on the result of a judgement can be used to infer knowledge about the usage annotations on the variables in the context. Lemma 3.3 states that a term of 0 usage needs no resources.

Lemma 3.3 (Zero needs nothing). *If* $\Gamma \vdash M \overset{0}{:} S$, *then* $0\Gamma = \Gamma$.

Note that the "other direction" of this lemma does not hold. If we have $0\Gamma \vdash M \overset{\sigma}{:} S$, then it is not necessarily the case that $\sigma = 0$. For example, the Tm-Bool-True rule produces judgements of arbitrary usage from no resources.

$$\frac{\Gamma \vdash M \overset{\sigma}{:} S}{\Gamma \vdash M \equiv M \overset{\sigma}{:} S}\ \textsc{Tm-Eq-Refl} \qquad \frac{\Gamma \vdash M \equiv N \overset{\sigma}{:} S}{\Gamma \vdash N \equiv M \overset{\sigma}{:} S}\ \textsc{Tm-Eq-Symm}$$

$$\frac{\Gamma \vdash M \equiv N \overset{\sigma}{:} S \qquad \Gamma \vdash N \equiv O \overset{\sigma}{:} S}{\Gamma \vdash M \equiv O \overset{\sigma}{:} S}\ \textsc{Tm-Eq-Tran}$$

$$\frac{\Gamma, x \overset{\sigma\pi}{:} S \vdash M \equiv M' \overset{\sigma}{:} T \qquad 0\Gamma \vdash S \equiv S' \qquad 0\Gamma, x \overset{0}{:} S \vdash T \equiv T'}{\Gamma \vdash \lambda x \overset{\pi}{:} S.M^T \equiv \lambda x \overset{\pi}{:} S'.M'^{T'} \overset{\sigma}{:} (x \overset{\pi}{:} S) \to T}\ \textsc{Tm-Eq-Lam-Cong}$$

$$\frac{\begin{array}{c}\Gamma_1 \vdash M \equiv M' \overset{\sigma}{:} (x \overset{\pi}{:} S) \to T \\ \Gamma_2 \vdash N \equiv N' \overset{\sigma}{:} S \qquad 0\Gamma_1 \vdash S \equiv S' \qquad 0\Gamma_1, x \overset{0}{:} S \vdash T \equiv T' \qquad 0\Gamma_1 = 0\Gamma_2\end{array}}{\Gamma_1 + \pi\Gamma_2 \vdash \mathrm{App}_{(x \overset{\pi}{:} S)T}(M, N) \equiv \mathrm{App}_{(x \overset{\pi}{:} S')T'}(M', N') \overset{\sigma}{:} T[N/x]}\ \textsc{Tm-Eq-App-Cong}$$

$$\frac{\Gamma_1, x \overset{\sigma\pi}{:} S \vdash M \overset{\sigma}{:} T \qquad \Gamma_2 \vdash N \overset{\sigma}{:} S}{\Gamma_1 + \pi\Gamma_2 \vdash \mathrm{App}_{(x \overset{\pi}{:} S)T}(\lambda x \overset{\pi}{:} S.M^T, N) \equiv M[N/x] \overset{\sigma}{:} T[N/x]}\ \textsc{Tm-Eq-Pi}\beta$$

$$\frac{\Gamma \vdash M \overset{\sigma}{:} (x \overset{\pi}{:} S) \to T}{\Gamma \vdash \lambda x \overset{\pi}{:} S.\mathrm{App}_{(x \overset{\pi}{:} S)T}(M, x)^T \equiv M \overset{\sigma}{:} (x \overset{\pi}{:} S) \to T}\ \textsc{Tm-Eq-Pi}\eta$$

$$\frac{\begin{array}{c}0\Gamma_1, z \overset{0}{:} \mathrm{Bool} \vdash T \equiv T' \qquad \Gamma_1 \vdash M_t \equiv M_t' \overset{\sigma}{:} T[\mathrm{true}/z] \\ \Gamma_1 \vdash M_f \equiv M_f' \overset{\sigma}{:} T[\mathrm{false}/z] \qquad \Gamma_2 \vdash N \equiv N' \overset{\sigma}{:} \mathrm{Bool} \qquad 0\Gamma_1 = 0\Gamma_2\end{array}}{\Gamma_1 + \Gamma_2 \vdash \mathrm{ElimBool}_{(z)T}(M_t, M_f, N) \equiv \mathrm{ElimBool}_{(z)T'}(M_t', M_f', N') \overset{\sigma}{:} T[N/z]}\ \textsc{Tm-Eq-ElimBool-Cong}$$

$$\frac{0\Gamma, z \overset{0}{:} \mathrm{Bool} \vdash T \qquad \Gamma \vdash M_t \overset{\sigma}{:} T[\mathrm{true}/z] \qquad \Gamma \vdash M_f \overset{\sigma}{:} T[\mathrm{false}/z]}{\Gamma \vdash \mathrm{ElimBool}_{(z)T}(M_t, M_f, \mathrm{true}) \equiv M_t \overset{\sigma}{:} T[\mathrm{true}/z]}\ \textsc{Tm-Eq-True}\beta$$

$$\frac{0\Gamma, z \overset{0}{:} \mathrm{Bool} \vdash T \qquad \Gamma \vdash M_t \overset{\sigma}{:} T[\mathrm{true}/z] \qquad \Gamma \vdash M_f \overset{\sigma}{:} T[\mathrm{false}/z]}{\Gamma \vdash \mathrm{ElimBool}_{(z)T}(M_t, M_f, \mathrm{false}) \equiv M_f \overset{\sigma}{:} T[\mathrm{false}/z]}\ \textsc{Tm-Eq-False}\beta$$

$$\frac{0\Gamma \vdash M \equiv M' \overset{0}{:} \mathrm{Set} \qquad 0\Gamma, x \overset{0}{:} \mathrm{El}(M) \vdash N \equiv N' \overset{0}{:} \mathrm{Set}}{0\Gamma \vdash (x \overset{\pi}{:} M) \to N \equiv (x \overset{\pi}{:} M') \to N' \overset{0}{:} \mathrm{Set}}\ \textsc{Tm-Eq-Set-Pi-Cong}$$

Fig. 3. Term Equality: $\Gamma \vdash M \equiv N \overset{\sigma}{:} S$

3.2.2 Weakening. Weakening allows the insertion of additional variables into the context of each judgement, provided that the inserted variable is marked with usage 0.

Lemma 3.4 (Weakening). *Weakening is admissible:*

$$\frac{\Gamma, \Gamma' \vdash \mathcal{J} \qquad 0\Gamma \vdash U}{\Gamma, x \overset{0}{:} U, \Gamma' \vdash \mathcal{J}}\ \textit{Weaken}$$

where $\mathcal{J}$ ranges over the judgements "is context", S, $S \equiv T$, $M \overset{\sigma}{:} S$ and $M \equiv N \overset{\sigma}{:} S$.

3.2.3 Substitution. Substitution (Lemma 3.5) allows for a variable $x \overset{\rho}{:} U$ to be replaced by a term of type U, as long as we add the resources it requires, multiplied by ρ, to the resources already present. The following lemma is proved by a mutual induction on the derivations of the first premise of each rule:

Lemma 3.5. *The following substitution rules are admissible:*

$$\frac{\Gamma_1, x \overset{\rho}{:} U, \Gamma' \vdash \qquad \Gamma_2 \vdash O \overset{\sigma}{:} U \qquad \rho\sigma = \rho \qquad 0\Gamma_1 = 0\Gamma_2}{(\Gamma_1 + \rho\Gamma_2), \Gamma'[O/x] \vdash} \textsc{Ctxt-Subst}$$

$$\frac{0\Gamma_1, x \overset{0}{:} U, 0\Gamma' \vdash S \qquad \Gamma_2 \vdash O \overset{0}{:} U \qquad 0\Gamma_1 = 0\Gamma_2}{0\Gamma_1, 0\Gamma'[O/x] \vdash S[O/x]} \textsc{Type-Subst}$$

$$\frac{0\Gamma_1, x \overset{0}{:} U, 0\Gamma' \vdash S \equiv T \qquad \Gamma_2 \vdash O \overset{0}{:} U \qquad 0\Gamma_1 = 0\Gamma_2}{0\Gamma_1, 0\Gamma'[O/x] \vdash S[O/x] \equiv T[O/x]} \textsc{Type-Eq-Subst}$$

$$\frac{\Gamma_1, x \overset{\rho}{:} U, \Gamma' \vdash M \overset{\sigma}{:} T \qquad \Gamma_2 \vdash O \overset{\sigma'}{:} U \qquad 0\Gamma_1 = 0\Gamma_2 \qquad \sigma'\rho = \rho}{(\Gamma_1 + \rho\Gamma_2), \Gamma'[O/x] \vdash M[O/x] \overset{\sigma}{:} T[O/x]} \textsc{Term-Subst}$$

$$\frac{\Gamma_1, x \overset{\rho}{:} U, \Gamma' \vdash M \equiv N \overset{\sigma}{:} T \qquad \Gamma_2 \vdash O \overset{\sigma'}{:} U \qquad 0\Gamma_1 = 0\Gamma_2 \qquad \sigma'\rho = \rho}{(\Gamma_1 + \rho\Gamma_2), \Gamma'[O/x] \vdash M[O/x] \equiv N[O/x] \overset{\sigma}{:} T[O/x]} \textsc{Term-Eq-Subst}$$

3.2.4 Inadmissibility of Substitution in McBride's system. McBride's original system allowed for term judgements of the form $\Gamma \vdash M \overset{\rho}{:} T$, where ρ is an arbitrary element of the semiring. We now show that such a formulation with the semiring $\{0, 1, \omega\}$ yields a system that is not closed under substitution. A proof attempt to show that substitution is admissible fails on the Tm-App because we need to take the term $\Gamma \vdash O \overset{\rho_1+\rho_2}{:} U$ that is being substituted in, and split it into some $\Gamma_1 \vdash O \overset{\rho_1}{:} U$ and $\Gamma_2 \vdash O \overset{\rho_2}{:} U$ such that $\Gamma_1 + \Gamma_2 = \Gamma$. However, this is not possible in general.

We illustrate the problem with the term:

$$f \overset{\omega}{:} (x \overset{1}{:} A) \to A \vdash \lambda x \overset{\omega}{:} A.fx \overset{\omega}{:} (x \overset{\omega}{:} A) \to A \tag{4}$$

This term effectively coerces some function f from a type that states it uses its argument once to a type that states that it uses its argument multiple times. This is possible even though McBride's system does not explicitly include weakening of usages.

The following derivation gives a context into which substitution of Judgement 4 fails.

$$\dfrac{\dfrac{}{y \overset{0}{:} A, g \overset{1}{:} (x \overset{\omega}{:} A) \to A \vdash g \overset{1}{:} (x \overset{\omega}{:} A) \to A} \qquad \dfrac{\dfrac{}{y \overset{0}{:} A, g \overset{\omega}{:} (x \overset{\omega}{:} A) \to A \vdash g \overset{\omega}{:} (x \overset{\omega}{:} A) \to A} \qquad \dfrac{}{y \overset{\omega}{:} A, y \overset{0}{:} (x \overset{\omega}{:} A) \to A \vdash y \overset{\omega}{:} A}}{y \overset{\omega}{:} A, g \overset{\omega}{:} (x \overset{\omega}{:} A) \to A \vdash gy \overset{\omega}{:} A}}{y \overset{\omega}{:} A, g \overset{\omega}{:} (x \overset{\omega}{:} A) \to A \vdash g(gy) \overset{1}{:} A}$$

If we attempt to substitute $\lambda x \overset{\omega}{:} A.fx$ for g, then to push the term into the two halves of the top level application, we need to split Judgement 4 according to the equation $\omega = 1 + \omega$. We already have a version of usage ω, but the following judgement is not derivable:

$$f \overset{1}{:} (x \overset{1}{:} A) \to A \nvdash \lambda x \overset{\omega}{:} A.fx \overset{1}{:} (x \overset{\omega}{:} A) \to A$$

Indeed, it is not possible to derive $y \overset{\omega}{:} A, f \overset{\omega}{:} (x \overset{1}{:} A) \to A \vdash f(fy) \overset{\omega}{:} A$ due to the mismatch between y's ω annotation, and f's 1 requirement. The implicit weakening in Judgement 4 is not accessible without an intervening λ-abstraction. This problem is fixed by our reformulation only allowing 0 or 1 usage on terms, which prohibits this kind of implicit "subusaging".

4 QUANTITATIVE CATEGORIES WITH FAMILIES

We propose Quantitative Categories with Families (QCwFs) as a canonical model for QTT that captures the intensional information recorded in QTT typing derivations. In this section and the next we define QCwFs and show how to soundly interpret QTT in them. This completes our sketched introduction to QCwFs from Section 2.3. In Section 6, we substantiate our claim that QCwFs usefully capture the usage information from QTT by defining concrete instances of QCwFs that use this information to assign interesting intensional content to QTT terms.

As we sketched in Section 2.3, a QCwF contains a Category with Families (CwF) to model the extensional content of contexts, types, and terms. Therefore, we first recall the definition of a CwF.

4.1 Categories with Families

A CwF [Dybjer 1996; Hofmann 1997] defines enough structure to interpret the contexts, types and terms of type theory, along with simultaneous substitutions and their action on types and terms. Judgemental equality on types and terms is modelled directly as equality in the model.

Definition 4.1. A *Category with Families* (CwF) structure $(\mathcal{C}, \mathrm{Ty}, \mathrm{Tm}, \top, -.-, \langle -, - \rangle)$ consists of:

(1) A category $\mathcal{C}$, whose objects are intended as the semantic interpretation of contexts and whose morphisms are the intepretation of simultaneous substitutions;
(2) For each object Δ in $\mathcal{C}$, a collection $\mathrm{Ty}(\Delta)$ of *semantic types*;
(3) For each object Δ in $\mathcal{C}$ and semantic type $S \in \mathrm{Ty}(\Delta)$, a collection $\mathrm{Tm}(\Delta, S)$ of *semantic terms*;
(4) For every morphism $f : \Delta \to \Delta'$ in $\mathcal{C}$, a function

$$-\{f\} : \mathrm{Ty}(\Delta') \to \mathrm{Ty}(\Delta)$$

for interpreting substitution in types, and for every $S \in \mathrm{Ty}(\Delta')$ a function

$$-\{f\} : \mathrm{Tm}(\Delta', S) \to \mathrm{Tm}(\Delta, S\{f\})$$

for interpreting substitution in terms, such that the following equations hold:

$$A\{\mathrm{id}_\Delta\} = A \qquad A\{f\}\{g\} = A\{f \circ g\} \qquad M\{\mathrm{id}_\Delta\} = M \qquad M\{f\}\{g\} = M\{f \circ g\}$$

(5) A chosen terminal object $\top$ in $\mathcal{C}$, for interpreting the empty context;
(6) For each object Δ in $\mathcal{C}$ and $S \in \mathrm{Ty}(\Gamma)$ an object $\Delta.S$ (called the *comprehension of* S) such that there is a bijection natural in Δ':

$$\mathcal{C}(\Delta', \Delta.S) \cong \{(f, M) \mid f : \Delta' \to \Delta, M \in \mathrm{Tm}(\Delta', S\{f\})\}$$

Given $f : \Delta' \to \Delta$ and $M \in \mathrm{Tm}(\Delta', S\{f\})$, we write $\langle f, M \rangle$ for the associated morphism $\Delta' \to \Delta.S$ in $\mathcal{C}$. Conversely, given a morphism $f : \Delta' \to \Delta.S$ in $\mathcal{C}$, we write $f^{\#1} : \Delta' \to \Delta$ and $f^{\#2} \in \mathrm{Tm}(\Delta', S\{f^{\#1}\})$ for the associated morphism and semantic term.

The following definitions derive projection and weakening structure from Definition 4.1, and also define the semantic counterpart of a simultaneous substitution of a term for a variable. We will provide usage annotated counterparts of these in our definition of QCwF below.

(1) For any Δ in $\mathcal{C}$ and $S \in \mathrm{Ty}(\Gamma)$, the *first projection* morphism $\mathrm{p}_{\Delta.S} : \Delta.S \to \Delta$ is defined as $\mathrm{p}_{\Delta.S} = \mathrm{id}^{\#1}_{\Delta.A}$. First projection is used to interpret weakening by discarding a single variable.

(2) For any Δ in $\mathcal{C}$ and $S \in \mathrm{Ty}(\Delta)$, the *second projection* $\mathsf{v}_{\Delta.S} \in \mathrm{Tm}(\Delta.S, S\{\mathsf{p}_{\Gamma.S}\})$ is defined as $\mathsf{v}_{\Delta.S} = \mathrm{id}^{\#2}_{\Delta.S}$. Second projection is used to interpret the selection from a context.
(3) For any Δ, Δ' in $\mathcal{C}$, $S \in \mathrm{Ty}(\Delta)$ and morphism $f : \Delta' \to \Delta$, the *weakening* of f by S, $\mathsf{wk}(f, S) : \Delta'.S\{f\} \to \Delta.S$ is defined as $\mathsf{wk}(f, S) = \langle f \circ \mathsf{p}_{\Delta'.S\{f\}}, \mathsf{v}_{\Delta'.S\{f\}} \rangle$. Weakening is used to lift semantic interpretations of simultaneous substitutions to extended contexts.
(4) For any Δ in $\mathcal{C}$, $S \in \mathrm{Ty}(\Delta)$ and $M \in \mathrm{Tm}(\Delta, S)$, we define the morphism $\overline{M} : \Delta \to \Delta.S$ as $\overline{M} = \langle \mathrm{id}_\Delta, M \rangle$. Morphisms $\overline{M}$ are used to interpret the substitution of a term for a variable.

4.2 Quantitative Categories with Families

Colouring in our sketch definition in Section 2.3, the definition of a *Quantitative Category with Families* incorporates a CwF, and layers over it facilities for representing terms and context morphisms with intensional information. We will require: *i)* the ability to scale and add semantic contexts, modelling the corresponding operations on precontexts; *ii)* the existence of semantic terms carrying intensional information ("resourced" terms), with a scaling operation; and *iii)* context extension with usage information, with the necessary projection, weakening and term subsitition operations. We provide some commentary and basic results after the definition.

Definition 4.2. For a semiring R, an *R-Quantitative Category with Families* (R-QCwF) consists of:

(1) A CwF (Definition 4.1) $(\mathcal{C}, \mathrm{Ty}, \mathrm{Tm}, \top, -.-, \langle -, - \rangle)$;
(2) A category $\mathcal{L}$, whose objects will be used to interpret contexts with resource annotations and whose morphisms will interpret simultaneous substitutions;
(3) Functors $U : \mathcal{L} \to \mathcal{C}$ and $J : \mathcal{C} \to \mathcal{L}$, such that U is faithful and $U \circ J = \mathrm{Id}_\mathcal{C}$;
(4) *(Addition)* Let $\mathcal{L} \times_\mathcal{C} \mathcal{L}$ stand for the pullback of $\mathcal{L} \xrightarrow{U} \mathcal{C} \xleftarrow{U} \mathcal{L}$. We require a bifunctor
$$(+) : \mathcal{L} \times_\mathcal{C} \mathcal{L} \to \mathcal{L}$$
such that $U(\Gamma_1 + \Gamma_2) = U\Gamma_1$ $(= U\Gamma_2)$, and with the following natural isomorphisms:
$$\mathsf{lunit} : \Gamma + J\Delta \cong \Gamma \qquad\qquad \mathsf{runit} : J\Delta + \Gamma \cong \Gamma$$
such that $U(\mathsf{lunit}) = U(\mathsf{runit}) = \mathrm{id}$.
(5) *(Scaling)* For $\rho \in R$, there is a functor $\rho(-) : \mathcal{L} \to \mathcal{L}$ such that $U(\rho(-)) = U(-)$, $0\Gamma = J(U\Gamma)$, and there are natural transformations $\mathsf{one} : \Gamma \to 1\Gamma$ and $\mathsf{dupzero} : J- \to \rho(J-)$, such that $U(\mathsf{one}) = U(\mathsf{dupzero}) = \mathrm{id}$.
(6) *(Resourced Terms)* For Γ in $\mathcal{L}$, $\rho \in R$, and $S \in \mathrm{Ty}(U\Gamma)$, there is a collection of semantic resourced terms $\mathrm{Tm}^\mathcal{L}(\Gamma, \rho\ S)$, with functions $U_{\Gamma.S} : \mathrm{Tm}^\mathcal{L}(\Gamma, \rho\ S) \to \mathrm{Tm}(U\Gamma, S)$ and $J_{\Delta.S} : \mathrm{Tm}(\Delta, S) \to \mathrm{Tm}^\mathcal{L}(J\Delta, 0\ S)$, such that $U_{J\Delta.S} \circ J_{\Delta.S} = \mathrm{Id}$, and $U_{\Gamma.S}$ is injective.
(7) *(Substitution in Resourced Terms)* For $f : \Gamma' \to \Gamma$, and $S \in \mathrm{Ty}(\Gamma)$, there is a function
$$-\{f\} : \mathrm{Tm}^\mathcal{L}(\Gamma, \rho\ S) \to \mathrm{Tm}^\mathcal{L}(\Gamma', \rho\ S\{f\})$$
such that $U(M\{f\}) = (UM)\{Uf\}$ and $J(M\{f\}) = (JM)\{Jf\}$.
(8) *(Scaling of Resourced Terms)* For each $\pi \in R$, there is a family of functions, natural in Γ:
$$\pi(-) : \mathrm{Tm}^\mathcal{L}(\Gamma, \rho\ S) \to \mathrm{Tm}^\mathcal{L}(\pi\Gamma, (\pi\rho)\ S)$$
such that $U(\pi M) = U(M)$, $0M = J(UM)$, and $1M = M\{\mathsf{one}\}$.
(9) *(Resourced context extension)* For Γ in $\mathcal{L}$, $\rho \in R$ and $S \in \mathrm{Ty}(U\Gamma)$, there is an object $\Gamma.\rho S$ in $\mathcal{L}$ such that $U(\Gamma.\rho S) = U\Gamma.S$ and there exist natural transformations:
$$\begin{array}{lcl} \mathsf{ext}_\pi & : & \pi\Gamma.(\pi\rho)S \to \pi(\Gamma.\rho S) \\ \mathsf{ext}_+ & : & (\Gamma_1 + \Gamma_2).(\rho_1 + \rho_2)S \to \Gamma_1.\rho_1 S + \Gamma_2.\rho_2 S \end{array}$$
such that $U(\mathsf{ext}_\pi) = \mathrm{id}$ and $U(\mathsf{ext}_+) = \mathrm{id}$.

(10) There exist resourced counterparts of the projection, weakening, and term substitution structure of the underlying CwF:
   (a) For Γ and $S \in \mathrm{Ty}(U\Gamma)$, there is a morphism $\mathsf{p}_{\Gamma.S} : \Gamma.0S \to \Gamma$ such that $U(\mathsf{p}_{\Gamma.S}) = \mathsf{p}_{U\Gamma.S}$;
   (b) For Γ, $S \in \mathrm{Ty}(U\Gamma)$, and $\rho \in R$, there is a semantic term $\mathsf{v}_{\Gamma.\rho S} \in \mathrm{Tm}(0\Gamma.\rho S, \rho\, S\{\mathsf{p}_{U\Gamma.S}\})$ such that $U(\mathsf{v}_{\Gamma.\rho S}) = \mathsf{v}_{U\Gamma.S}$;
   (c) For $f : \Gamma \to \Gamma'$, $\rho \in R$, and $S \in \mathrm{Ty}(U\Gamma')$ there is a morphism $\mathsf{wk}(f, \rho S) : \Gamma.\rho S\{Uf\} \to \Gamma'.\rho S$ such that $U(\mathsf{wk}(f, \rho S)) = \mathsf{wk}(Uf, S)$;
   (d) For $M \in \mathrm{Tm}(\Gamma_2, \rho S)$ and Γ_1 such that $U\Gamma_1 = U\Gamma_2$, there is a morphism $\overline{M} : \Gamma_1 + \Gamma_2 \to \Gamma_1.\rho S$ such that $U(\overline{M}) = \overline{UM}$.

The structure in the category $\mathcal{L}$ acts as an intensional refinement of the corresponding structure in the CwF $\mathcal{C}$. We have analogous constructs in $\mathcal{L}$: semantic contexts, substitutions, terms, and comprehension structure, but the "typing" of these constructs is more refined. For example, context extension in $\mathcal{C}$ is $\Gamma.S$, whereas context extension in $\mathcal{L}$ includes additional resource information: $\Gamma.\rho S$. Likewise, the morphism $\overline{M}$ derived from a semantic term M has type $\Gamma_1 + \Gamma_2 \to \Gamma_1.\rho S$, indicating how the intensional content of Γ_2 is "used up" in the production of ρS.

Readers familiar with categorical models of linear type theories or coeffect systems may find some aspects of Definition 4.2 odd. Specifically, there appears to be a shortage of the typical structural natural transformations and isomorphisms present in symmetric monoidal categories with (graded) exponential comonads. Moreover, there are no coherence axioms required for the structural morphisms that do exist.

We explain the apparent lack of structural morphisms (associativity, symmetry, etc.) by the way that CwFs, and consequently QCwFs, model contexts via comprehension: $\Delta.S$ and $\Gamma.\rho S$, and its interaction with the usage accounting. Comprehension builds up the interpretation of each context in a left-associative normalised fashion, allowing the type S to "see" the entire preceeding context. On top of this, we layer the structure required to manipulate the resource annotations via scaling and addition. When we need to shift perspective from contexts-as-collections-of-variables to contexts-as-carriers-of-resource, we use the distributivity morphisms ext_π and ext_+. (This will manifest as the splitting ($\mathsf{s}_{\Gamma_1,\Gamma_2}$) and distribution morphisms ($\mathsf{m}_{\pi,\Gamma}$) we will use in the interpretation below.) In the interpretation of linear type theories (see, for example, Barber [1996]), symmetric monoidal structure is used for both purposes: to combine the interpretations of multiple variables, and to combine the multiple resources. Thus we need associativity and symmetry to rearrange these two perspectives. Our definition of a QCwF contains precisely the morphisms we need.

The lack of coherence axioms (e.g., the pentagon diagram for associativity [Mac Lane 1998]) is easier to explain. We have required the functor U to be faithful, and the structural morphisms we have required all map to the identity via U. Therefore, if we have two structural morphisms in $\mathcal{L}$ with the same domain and codomain, the fact that they map to the same morphism in $\mathcal{C}$ implies by faithfulness that they are equal. This also extends to the rest of the structure on $\mathcal{L}$: in our proofs of soundness, we use equational reasoning in $\mathcal{C}$ and map it back to $\mathcal{L}$ via U. For example, we have required substitution operations for the resourced semantic terms in Definition 4.2, but not required that these preserve identities and composition. However, these preservation properties can be derived from the corresponding functoriality of the semantic terms in the underlying CwF and the injectivity of U on semantic terms.

We note the following consequence of our definition of QCwF, which states that we have a ready supply of "uninteresting" QCwFs built from CwFs that ignore the usage annotation information. This means that QTT has consistent models. We will consider instances of QCwFs with interesting interpretations of the intensional content of terms in Section 6.

Proposition 4.3. *Let R be any commutative semiring. Every CwF $(\mathcal{C}, \mathrm{Ty}, \mathrm{Tm}, 1, -.-, \langle -, - \rangle)$ yields an R-QCwF with $\mathcal{L} = \mathcal{C}$.*

4.3 Type Formers in a QCwF

Definition 4.2 only defines enough structure to interpret the construction of contexts and the Tm-Var and Tm-Conv rules. To interpret the rest of QTT, we need to define requirements on a QCwF for the three QTT type formers, dependent function types, booleans, and the universe, that we introduced in Section 2.1.

4.3.1 Dependent Function Types. As in the definition of QCwFs, we first define the requirements for the extensional interpretation of dependent function types, and then refine it with intensional information.

Definition 4.4. A CwF $\mathcal{C}$ *supports dependent products with usage information* if for all semantic contexts $\Delta \in \mathcal{C}$ and semantic types $S \in \mathrm{Ty}(\Delta)$, $T \in \mathrm{Ty}(\Delta.A)$, and usages $\pi \in R$, there exists a semantic type $\Pi\pi ST \in \mathrm{Ty}(\Delta)$, natural in Δ, such that there is a bijection $\Lambda : \mathrm{Tm}(\Delta.S, T) \cong \mathrm{Tm}(\Delta, \Pi\pi ST)$, natural in Δ.

For Δ, S and T as in Definition 4.4, and $M \in \mathrm{Tm}(\Delta, \Pi\pi ST)$ and $N \in \mathrm{Tm}(\Delta, S)$, we define extensional semantic application $App^{\Delta}_{S,T}(M, N) \in \mathrm{Tm}(\Delta, T\{\overline{N}\})$ as $App^{\Delta}_{S,T} = (\Lambda^{-1}(M))\{\overline{N}\}$.

Definition 4.5. A QCwF $(\mathcal{L}, \mathcal{C}, \dots)$ *supports dependent products with usage information* if the CwF $\mathcal{C}$ supports dependent products with usage information (Definition 4.4), and for each $\sigma \in \{0, 1\}$, there is an bijection $\Lambda^{\mathcal{L}} : \mathrm{Tm}^{\mathcal{L}}(\Gamma.(\sigma\pi)S, \sigma T) \cong \mathrm{Tm}^{\mathcal{L}}(\Gamma, \sigma\Pi\pi ST)$, natural in Γ such that $U \circ \Lambda_{\mathcal{L}} = \Lambda \circ U$ and $U \circ \Lambda^{-1}_{\mathcal{L}} = \Lambda^{-1} \circ U$.

Intensional application is also a derived operation. Given $M \in \mathrm{Tm}(\Gamma_1, \sigma(\Pi\pi ST))$ and $N \in \mathrm{Tm}(\Gamma_2, (\pi\sigma)S)$, such that $U\Gamma_1 = U\Gamma_2$, we define

$$App_{\Gamma_1,\Gamma_2,\sigma,\pi,S,T}(M, N) = \Lambda^{-1}_{\Gamma_1,\sigma,\pi,S,T}(M)\{\overline{N}\} \in \mathrm{Tm}(\Gamma_1 + \Gamma_2, \sigma T\{U\overline{N}\})$$

Note how the intensional application refines the typing of the extensional application defined above. We also have that $U(App(M, N)) = App(UM, UN)$, so the extensional content of an intensional application is the extensional application of the extensional content of the two components.

4.3.2 Boolean Type. As for function types, we define the requirements for interpreting boolean types by first defining the extensional requirements, and then defining the intensional interpretations that are related to the extensional ones via U.

Definition 4.6. A CwF $\mathcal{C}$ *supports a boolean type* if, for all $\Delta \in \mathcal{C}$, there is a semantic type $Bool_\Delta \in \mathrm{Ty}(\Delta)$, semantic terms $true, false \in \mathrm{Tm}(\Delta, Bool_\Delta)$, and, for each $S \in \mathrm{Ty}(\Delta.Bool_\Delta)$, a function on semantic terms:

$$ElimBool_A : \mathrm{Tm}(\Delta, S\{\overline{true}\}) \times \mathrm{Tm}(\Delta, S\{\overline{false}\}) \to \mathrm{Tm}(\Delta.Bool, S)$$

such that all the above are natural in Δ, and the following two equations hold:

$$ElimBool_A(M_t, M_f)\{\overline{true}\} = M_t \qquad ElimBool_A(M_t, M_f)\{\overline{false}\} = M_f$$

Definition 4.7. A QCwF $(\mathcal{C}, \mathcal{L}, \dots)$ *supports a boolean type* if the underlying CwF $\mathcal{C}$ supports a boolean type (Definition 4.6), there exist semantic terms $true \in \mathrm{Tm}^{\mathcal{L}}(J\Delta, \sigma\ Bool_\Delta)$ and $false \in \mathrm{Tm}^{\mathcal{L}}(J\Delta, \sigma\ Bool_\Delta)$, and there is a function:

$$ElimBool_A : \mathrm{Tm}^{\mathcal{L}}(\Gamma, \sigma\ A\{\overline{true}\}) \times \mathrm{Tm}^{\mathcal{L}}(\Gamma, \sigma\ A\{\overline{false}\}) \to \mathrm{Tm}^{\mathcal{L}}(\Gamma.\sigma Bool, \sigma\ A)$$

such that $U(true) = true$, $U(false) = false$, and $U(ElimBool_A(M_t, M_f)) = ElimBool_A(UM_t, UM_f)$.

4.3.3 Universe Type. The universe type in QTT has no run time representation. Every term judgement building elements of Set has the form $\Gamma \vdash M \overset{0}{:} \mathrm{Set}$. Therefore, we only need the appropriate structure in the extensional CwF $\mathcal{C}$ in order to interpret the universe type. The only difference from the standard requirements for non quantitative type theory is that we need a code for the resourced variant of dependent function types.

Definition 4.8. A CwF $\mathcal{C}$ *supports a universe closed under booleans and resourced dependent product* if: (a) for all Δ in $\mathcal{C}$ there exists a semantic type $Set_\Delta \in \mathrm{Ty}(\Delta)$, and a semantic type $El_\Delta \in \mathrm{Ty}(\Delta.Set_\Delta)$, both natural in Δ; (b) there exists a semantic term $Bool_\Delta \in \mathrm{Tm}(\Delta, Set_\Delta)$, and for all $\pi \in R$, $M \in \mathrm{Tm}(\Delta, Set_\Delta)$, $N \in \mathrm{Tm}(\Delta.El_\Delta\{\overline{M}\}, Set_{\Delta.El_\Delta\{\overline{M}\}})$, there exists a semantic term $\Pi\pi MN \in \mathrm{Tm}(\Delta, Set_\Delta)$, both natural in Δ; and (c) such that the following equations hold:

$$El_\Delta\{\overline{Bool_\Delta}\} = Bool_\Delta \qquad El_\Delta\{\overline{\Pi\pi MN}\} = \Pi\pi(El_\Delta\{\overline{M}\})(El_{\Delta.El_\Delta\{\overline{M}\}}\{\overline{N}\})$$

5 INTERPRETATION OF QUANTITATIVE TYPE THEORY

Rigorously defining the interpretation of a dependent type theory is complicated, due to the mutually recursive nature of their definition: contexts rely on types, which rely on terms; which rely on contexts. Therefore, we follow Hofmann [1997]'s presentation of the interpretation of Type Theory in CwF by first defining a partial interpretation of the presyntax, and then showing that this interpretation is well defined on well typed presyntax. In the following we use $s \simeq t$ to denote *Kleene equality* on partial data: if either of s or t is defined, then so is the other, and they are equal.

5.1 Partial Interpretation of the Presyntax

We interpret precontexts Γ as objects $[\![\Gamma]\!]$ of our category $\mathcal{L}$. Analogous to the syntactic result that well formedness of contexts does not depend on the resource annotations (Lemma 3.1), we also have that the interpretation of contexts does not depend on the resource annotations (Lemma 5.1). We also define, for every pair of precontexts Γ_1, Γ_2 such that $0\Gamma_1 = 0\Gamma_2$ a morphism $\mathsf{s}_{\Gamma_1,\Gamma_2} : [\![\Gamma_1 + \Gamma_2]\!] \to [\![\Gamma_1]\!] + [\![\Gamma_2]\!]$ and for every precontext Γ and semiring element $\pi \in R$, a morphism $\mathsf{m}_{\pi,\Gamma} : [\![\pi\Gamma]\!] \to \pi[\![\Gamma]\!]$. These morphisms will allow us to manipulate the intensional content of contexts when we interpret terms. In the interpretation of application, we will need to split and scale the contexts in order to use our derived intensional application combinator from Section 4.3.1. These morphisms will be defined whenever their domains and codomains are (Lemma 5.2).

We partially define the interpretations of types $[\![\Gamma; S]\!]$ as elements of $\mathrm{Ty}(U[\![\Gamma]\!])$ and terms $[\![\Gamma; \sigma M]\!]$ as elements of $\mathrm{Tm}([\![\Gamma]\!], \sigma, [\![\Gamma; S]\!])$. Our interpretation of terms is non standard in that as well as (partially) returning a semantic resourced qterm in our QCwF, it also returns the usage annotations required to type that term. These are packaged together as a pair $(t \mid \Gamma)$, where Γ is a precontext annotated with the usage annotations expected by the interpretation t. This is required so that we can select the right semantic combinators in the interpretation, so that we get the correct intensional interpretation of terms. This is analogous to a compiler's code generation phase making use of information on how variables are used by the program to produce efficient code. We often know what the usage annotations are going to be, because we are starting with a typed term, so we define the notation $[\![\lfloor\Gamma\rfloor; \sigma M]\!]$ to be t whenever $[\![0\Gamma; \sigma M]\!] \simeq (t \mid \Gamma)$.

The interpretation of QTT in a QCwF is defined by the following clauses by induction on the structure of the presyntax:

Contexts.

- $[\![\diamond]\!] = J\top$

- $\mathsf{s}_{\diamond,\diamond} = \left([\![\diamond + \diamond]\!] = [\![\diamond]\!] = J\top \overset{\mathsf{runit}^{-1}}{\longrightarrow} J\top + J\top = [\![\diamond]\!] + [\![\diamond]\!] \right)$
- $\mathsf{m}_{\pi,\diamond} = \left([\![\pi\diamond]\!] = [\![\diamond]\!] = J\top \overset{\mathsf{dupzero}}{\longrightarrow} \pi(J\top) = \pi[\![\diamond]\!] \right)$
- $[\![\Gamma, x \overset{\rho}{:} S]\!] = [\![\Gamma]\!].\rho[\![\Gamma; S]\!]$
- $\mathsf{s}_{(\Gamma_1, x \overset{\rho_1}{:} S), (\Gamma_2, x \overset{\rho_2}{:} S)}$ is the composite:

$$
\begin{array}{c}
[\![(\Gamma_1, x \overset{\rho_1}{:} S) + (\Gamma_2, x \overset{\rho_2}{:} S)]\!] \\
\simeq [\![(\Gamma_1 + \Gamma_2), x \overset{\rho_1+\rho_2}{:} S]\!] \\
\simeq [\![\Gamma_1 + \Gamma_2]\!].(\rho_1 + \rho_2)[\![\Gamma_1 + \Gamma_2; S]\!] \\
\Big\downarrow \mathsf{wk}(\mathsf{s}_{\Gamma_1,\Gamma_2}, (\rho_1+\rho_2)[\![\Gamma_1; S]\!]) \\
([\![\Gamma_1]\!] + [\![\Gamma_2]\!]).(\rho_1 + \rho_2)[\![\Gamma_1 + \Gamma_2; S]\!] \\
\Big\downarrow \mathsf{ext}_{+} \\
[\![\Gamma_1]\!].\rho_1[\![\Gamma_1; S]\!] + [\![\Gamma_2]\!].\rho_2[\![\Gamma_2; S]\!] \\
\simeq [\![\Gamma_1, x \overset{\rho_1}{:} S]\!] + [\![\Gamma_2, x \overset{\rho_2}{:} S]\!]
\end{array}
$$

- $\mathsf{m}_{\pi,(\Gamma, x \overset{\rho}{:} S)}$ is the composite:

$$
\begin{array}{c}
[\![\pi(\Gamma, x \overset{\rho}{:} S)]\!] \\
\simeq [\![\pi\Gamma, x \overset{\pi\rho}{:} S]\!] \\
\simeq [\![\pi\Gamma]\!].(\pi\rho)[\![\pi\Gamma; S]\!]
\end{array}
\quad \xrightarrow{\mathsf{wk}(\mathsf{m}_{\pi,\Gamma}, (\pi\rho)[\![\pi\Gamma;S]\!])} \pi[\![\Gamma]\!].(\pi\rho)[\![\Gamma; S]\!] \xrightarrow{\mathsf{ext}_{\pi,\rho}} \pi([\![\Gamma]\!].\rho[\![\Gamma; S]\!]) \simeq \pi[\![\Gamma, x \overset{\rho}{:} S]\!]
$$

Types. are interpreted as elements of $\mathsf{Ty}(U[\![\Gamma]\!])$, assuming that the interpretation of the context is defined. Each pretype is interpreted by the corresponding structure we have assumed in a QCwF. In the El case, note how the usage annotation 0 is fed into the interpretation of the term M, corresponding to the usage annotations in the typing rule.

- $[\![\Gamma; (x \overset{\pi}{:} S) \to T]\!] = \Pi\pi[\![\Gamma; S]\!][\![\Gamma, x \overset{0}{:} S; T]\!]$
- $[\![\Gamma; \mathsf{Bool}]\!] = \mathit{Bool}_{U[\![\Gamma]\!]}$
- $[\![\Gamma; \mathsf{Set}]\!] = \mathit{Set}_{U[\![\Gamma]\!]}$
- $[\![\Gamma; \mathsf{El}(M)]\!] = \mathit{El}_{U[\![\Gamma]\!]}\{\overline{U[\![\lfloor 0\Gamma \rfloor; 0\ M]\!]}\}$

Terms. The interpretation of terms returns a pair $(t \mid \Gamma)$ of a semantic term t and a context with usage annotations Γ. By definition, we will always have $0\Gamma = 0\Delta$, where Δ is the input context. The interpretation of terms is where the interpretation of QTT differs most from interpretation of standard type theory in a CwF. For precontexts and pretypes, tracking usage annotations is a bookkeeping task. For preterms, we must select the correct combinators that match the refined usage information. In the LCA-realisability model below, this will correspond to selecting the appropriate combinator for the usage annotations.

Variables are interpreted using the projection combinators v and p, stepping through the context to find the correct variable. The variables that are not selected are marked as 0 usage.

- $[\![\Delta, x : S; \sigma\ x]\!] = (\mathsf{v}_{[\![0\Delta]\!].\sigma[\![0\Delta;S]\!]}\{\mathsf{wk}(\mathsf{m}_{0,0\Delta}, \sigma[\![0\Delta; S]\!])\} \mid 0\Delta, x \overset{\sigma}{:} S)$
- $[\![\Delta, x : S, \Delta', y : T; \sigma\ x]\!] = \mathsf{let}\ (t \mid \Gamma, x \overset{\sigma}{:} S, \Gamma') = [\![\Delta, x : S, \Delta'; \sigma\ x]\!]\ \mathsf{in}$
$$(t\{\mathsf{p}_{[\![\Gamma, x \overset{\sigma}{:} S, \Gamma']\!].[\![\Gamma, x \overset{\sigma}{:} S, \Gamma'; T]\!]}\} \mid \Gamma, x \overset{\sigma}{:} S, \Gamma', y \overset{0}{:} T)$$

Introduction and elimination of dependent functions are interpreted using the Λ and derived App combinator, respectively. In the elimination case, we use the s and m morphisms defined above to split the interpretation of a pair of summed precontexts into a summed pair of interpretations. Semantically, this splits the intensional content supplied to the function application into the content for the function and the content for its argument. Note how we have to use the type information present in the presyntax, and the usage information provided by the interpretation of te subterms to define these interpretations.

- $[\![\Delta; \sigma\ \lambda x^{\pi}\!:\!S.M^T]\!] = \text{let } (t \mid \Gamma, x\ {}^{\sigma\pi}\!:\!\ S) = [\![\Delta, x : S; \sigma\ M]\!] \text{ in } (\Lambda_{[\![\Gamma]\!], \sigma, \pi, [\![\Gamma;S]\!], [\![\Gamma, x^{0}:S;T]\!]}(t) \mid \Gamma)$
- $[\![\Delta; \sigma\ \text{App}_{(x^{\pi}:S)T}(M, N)]\!] =$
  $\quad \text{let } (t_1 \mid \Gamma_1) = [\![\Delta; \sigma\ M]\!] \text{ and } (t_2 \mid \Gamma_2) = [\![\Delta; \sigma\ N]\!] \text{ in}$
  $\quad (\text{App}_{[\![\Gamma_1]\!], \pi[\![\Gamma_2]\!], \sigma, [\![\Gamma_1;S]\!], [\![\Gamma_1, x^{0}:S;T]\!]}(t_1, \pi t_2)\{([\![\Gamma_1]\!] + \mathsf{m}_{\pi,\Gamma_2}) \circ \mathsf{s}_{\Gamma_1,\pi\Gamma_2}\} \mid \Gamma_1 + \pi\Gamma_2)$
  where the morphism $([\![\Gamma_1]\!] + \mathsf{m}_{\pi,\Gamma_2}) \circ \mathsf{s}_{\Gamma_1,\pi\Gamma_2}$ has type $[\![\Gamma_1 + \pi\Gamma_2]\!] \to [\![\Gamma_1]\!] + \pi[\![\Gamma_2]\!]$, when it is defined.

Booleans are interpreted by the corresponding combinators we assumed to be present in a QCwF that supports a boolean type. Boolean elimination again uses the splitting morphism s to split intensional resources between the branches and the boolean being eliminated.

- $[\![\Delta; \sigma\ \text{true}]\!] = (\mathit{true}_{[\![0\Delta]\!]} \mid 0\Delta)$
- $[\![\Delta; \sigma\ \text{false}]\!] = (\mathit{false}_{[\![0\Delta]\!]} \mid 0\Delta)$
- $[\![\Delta; \sigma\ \text{ElimBool}_{(z)T}(M_t, M_f, N)]\!] = \text{let } (t_t \mid \Gamma_1) = [\![\Delta; \sigma\ M_t]\!]$
  $\text{and } (t_f \mid \Gamma_1) = [\![\Delta; \sigma\ M_f]\!]$
  $\text{and } (t \mid \Gamma_2) = [\![\Delta; \sigma\ N]\!]$
  $\text{in } \mathit{ElimBool}_{[\![0\Delta, z^{0}:\text{Bool};T]\!]}(t_t, t_f)\{\overline{N} \circ \mathsf{s}_{\Gamma_1,\Gamma_2}\}$

Finally, the preterms representing small types are interpreted by the corresponding structure in the CwF, transported to the resourced terms by the function J:

- $[\![\Delta; 0\ \text{Bool}]\!] = (\mathit{JBool}_{U[\![0\Delta]\!]} \mid 0\Delta)$
- $[\![\Delta; 0\ (x\ {}^{\pi}\!:\ M) \to N]\!] = \text{let } (t_1 \mid 0\Delta) = [\![\Delta; 0\ M]\!] \text{ and } (t_2 \mid 0\Delta) = [\![\Delta, x\ {}^{0}\!:\ \mathsf{El}(M); 0\ N]\!] \text{ in}$
  $(J\Pi\pi(Ut_1)(Ut_2) \mid 0\Delta)$

5.2 Soundness: Complete Interpretation of the Syntax

Proving soundness of the interpretation (Theorem 5.4, below) essentially involves a large mutual induction on the five judgement forms constituting QTT. However, there are several intermediary lemmas needed for the proof. In common with the interpretation of normal type theory, we need to show that weakening and substitution are soundly interpreted, and have the right effect. This is very similar to the case for the interpretation of normal type theory, as described by Hofmann [1997], so we omit it here[2]. Peculiar to QTT, though, is the interpretation of the usage annotation manipulation. The semantic counterpart of the fact that usage annotations do not affect context well formedness (Lemma 3.1) is the following:

Lemma 5.1. *Let Γ_1, Γ_2 be precontexts such that $0\Gamma_1 = 0\Gamma_2$. Then $[\![\Gamma_1]\!]$ is defined iff $[\![\Gamma_2]\!]$ is defined, and in this case $U[\![\Gamma_1]\!] = U[\![\Gamma_2]\!]$.*

This lemma is used in the soundness proof to transfer the interpretations of types between contexts that have the same extensional content.

[2]Please refer to the anonymous supplementary material for detais.

Moreover, the intensional content manipulation morphisms $\mathsf{m}_{\pi,\Gamma}$ and $\mathsf{s}_{\Gamma_1,\Gamma_2}$ that we defined as part of our interpretation are well defined whenever their domain and codomain are well defined, and they have no extensional content of their own (they both map to the identity via U):

Lemma 5.2. *The multiplication and splitting morphisms are well defined:*

(1) *Let Γ be a precontext. The expression $\mathsf{m}_{\pi,\Gamma}$ is defined iff $[\![\Gamma]\!]$ (equivalently $[\![\pi\Gamma]\!]$, by Lemma 5.1) is defined, and in this case is a morphism $[\![\pi\Gamma]\!] \to \pi[\![\Gamma]\!]$ such that $U\mathsf{m}_{\pi,\Gamma} = \mathrm{id}$;*
(2) *Let Γ_1, Γ_2 be precontexts such that $0\Gamma_1 = 0\Gamma_2$. The expression $\mathsf{s}_{\Gamma_1,\Gamma_2}$ is defined iff $[\![\Gamma_1]\!]$ (equivalently, $[\![\Gamma_2]\!]$ and $[\![\Gamma_1+\Gamma_2]\!]$, by Lemma 5.1) is defined, and in this case is a morphism $[\![\Gamma_1+\Gamma_2]\!] \to [\![\Gamma_1]\!]+[\![\Gamma_2]\!]$ such that $U\mathsf{s}_{\Gamma_1,\Gamma_2} = \mathrm{id}$.*

We also have a semantic analogue of Lemma 3.2, which allows zeroing out of terms. This is used when substituting terms for variables with 0 usage:

Lemma 5.3. *Let Γ be a precontext and M be a preterm. If $[\![\lfloor\Gamma\rfloor; \sigma\ M]\!]$ is defined, then so is $[\![\lfloor 0\Gamma\rfloor; 0\ M]\!]$, and is equal to $0[\![\lfloor\Gamma\rfloor; \rho\ M]\!]\{\mathsf{m}_{0,0\Gamma}\}$.*

Finally, our main soundness theorem is as follows, which shows that every judgement of QTT has a defined meaning in the QCwF structure. As in Hofmann [1997], this is proved by an induction on the derivations of the judgements involved, using the lemmas above as well as the results about the interpretation of weakening and subsitution.

Theorem 5.4. *The partial interpretation above is defined on well formed presyntax:*

(1) *If $\Gamma \vdash$, then $[\![\Gamma]\!]$ is an object of $\mathcal{L}$;*
(2) *If $\Gamma \vdash S$, then $[\![\Gamma; S]\!]$ is an element of $\mathrm{Ty}(U[\![\Gamma]\!])$;*
(3) *If $\Gamma \vdash S \equiv T$, then $[\![\Gamma; S]\!] = [\![\Gamma; T]\!]$;*
(4) *If $\Gamma \vdash M \overset{\sigma}{:} S$, then $[\![\lfloor\Gamma\rfloor; \sigma\ M]\!]$ is an element of $\mathrm{Tm}([\![\Gamma]\!], \sigma[\![\Gamma; S]\!])$;*
(5) *If $\Gamma \vdash M \equiv N \overset{\sigma}{:} S$, then $[\![\lfloor\Gamma\rfloor; \sigma\ M]\!] = [\![\lfloor\Gamma\rfloor; \sigma\ N]\!]$.*

6 QUANTITATIVE CATEGORIES WITH FAMILIES FROM LCAS

We now complete our definition of the realisability model of QTT over an LCA $\mathcal{A}$ that we started in Section 2.2. We then describe three concrete LCAs, based on Scott [1976]'s Graph Model of the untyped λ-calculus, Abramsky et al. [2002]'s Geometry of Interaction situations, and Simpson [2005]'s linear λ-calculus.

6.1 Realisability Models over LCAs

In section Section 2.2.2, we described the category of assemblies over an LCA. As stated above, we use this category as the category $\mathcal{L}$ of interpretations of contexts with intensional content and simultaneous substitutions between them. To complete the remaining obligations for the structure of a QCwF, we need to define how to scale and add objects and morphisms of this category, and how to define the context extension with usage annotations. This will provide a QCwF for the $\{0, 1, \omega\}$ semiring. We discuss models for other semirings below.

6.1.1 Scaling and Addition. Scaling of an assembly $\Gamma = (|\Gamma|, \Vdash_\Gamma)$ yields an assembly with $|\pi\Gamma| = |\Gamma|$ (i.e., scaling does not alter the extensional content), and the realisability relation is defined depending on the value of π. When $\pi = 0$, then $a \Vdash_{0\Gamma} \gamma$ iff $a = I$; when $\pi = 1$, then $a \Vdash_{1\Gamma} \gamma$ iff $a \Vdash_\Gamma \gamma$; and then $\pi = \omega$, then $a \Vdash_{\omega\Gamma} \gamma$ iff there exists a b such that $a = !b$ and $b \Vdash_\Gamma \gamma$. Scaling of morphisms does not affect the function component, but the realiser a_f becomes I when $\pi = 0$, stays the same when $\pi = 1$ and becomes $F \cdot !a_f$ when $\pi = \omega$.

Addition of assemblies $\Gamma_1 = (|\Gamma|, \models_{\Gamma_1})$ and $\Gamma_2 = (|\Gamma|, \models_{\Gamma_2})$ (note that they have the same underlying set) is defined as $\Gamma_1 + \Gamma_2 = (|\Gamma|, \models_{\Gamma_1+\Gamma_2})$, where $a \models_{\Gamma_1+\Gamma_2} \gamma$ iff there exist b, c such that $a = [b, c]$ and $b \models_{\Gamma_1} \gamma$ and $c \models_{\Gamma_2} \gamma$. Thus addition of intensional intepretations of contexts does not affect the extensional content, but does affect the computation resources available. The resources required for the constituent contexts are paired together.

6.1.2 Context Extension. Given an assembly $(|\Gamma|, \models_\Gamma)$ and a type $S \in \mathrm{Ty}(|\Gamma|)$, the extensional content of context extension is defined as $|\Gamma.\rho S| = \{(\gamma, s) \mid \gamma \in |\Gamma|, s \in |S(\gamma)|\}$. The realisability relation $\models_{\Gamma.\rho S}$ is defined by cases on ρ:

$$\begin{array}{lcl} a \models_{\Gamma.0S} (\gamma, s) & \Leftrightarrow & \exists b.\ a = [b, I] \wedge b \models_\Gamma \gamma \\ a \models_{\Gamma.1S} (\gamma, s) & \Leftrightarrow & \exists b, c.\ a = [b, c] \wedge b \models_\Gamma \gamma \wedge c \models_{S(\gamma)} s \\ a \models_{\Gamma.\omega S} (\gamma, s) & \Leftrightarrow & \exists b, c.\ a = [b, !c] \wedge b \models_\Gamma \gamma \wedge c \models_{S(\gamma)} s \end{array}$$

Thus, extension of a context by a 0 use variable results in a realisability relation that represents that spot by an erased dummy value. Likewise, extension by a multiple use variable results in a relation that represents that variable by a !'d realiser. Thus, we produce the interpretation of contexts we described in Section 2.2.2.

6.1.3 Type Formers. The interpretation of dependent function types in the LCA realisability model is very similar to the usual definition in realisability models [Hofmann 1997]: the extensional part of the type is the set of *realisable* functions. A small modification is required to handle the usage annotation on the input, analogous to the case-by-case definition we used for context extension above. The interpretation of the universe type assumes the existence of a Grothendieck universe of small sets in our ambient set theory, and interprets the type Set as the collection of small assemblies. Finally, the interpretation of booleans is standard except that we have to extend our definition of LCA with some additional structure to model resource sensitive booleans:

Definition 6.1. An LCA $\mathcal{A}$ has conditionals if there are elements $T, F \in \mathcal{A}$ and a function $E : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ such that $E(p, q) \cdot T = p$ and $E(p, q) \cdot F = q$.

6.1.4 Variants. The QCwF we have defined above models usages tracked by the zero, one, many semiring $\{0, 1, \omega\}$. The model can be adapted to other semirings.

Erasability. For the {erased, present} semiring, we can use the standard S, K, I combinators, using I to represent data that has been erased. This provides a model of runtime erasure.

Precise resource accounting. For the semiring of natural numbers, with the usual addition and multiplication, we can construct a model using just the B, C, I fragment of an LCA (with E, T, F to model booleans). For any realisability relation $\models_\Gamma$, we define the n-ary pairing of it as $a \models^n_\Gamma \gamma$ iff there exist $b_1, \ldots, b_n$ such that $a = [b_1, \ldots, b_n]$ and $b_i \models_\Gamma \gamma$. We then use this to define scaling of assemblies by natural numbers and context extension. This model represents data that is used n times as n copies of that data that, crucially, all have the same extensional meaning. This kind of resource refinement is not available in traditional linear systems since they do not distinguish between intensional use and extensional meaning in the way QTT does.

Arbitrary semirings. The definition of LCA can be altered to track usages in arbitrary semirings R. The exponential operation is now indexed by elements of the semiring $!_\pi : \mathcal{A} \to \mathcal{A}$, and we alter the combinators to take this structure into account:

Definition 6.2 (R-Linear Combinatory Algebra). Let R be a semiring. An *R-Linear Combinatory Algebra* (R-LCA) is a BCI-algebra $(\mathcal{A}, (\cdot), B, C, I)$ with function $!_\rho : \mathcal{A} \to \mathcal{A}$, for all $\rho \in R$ and

elements $K, W_{\pi\rho}, D, \delta_{\pi\rho}, F_\rho \in \mathcal{A}$ such that:

$$K \cdot x \cdot !_0 y = x \qquad W_{\pi\rho} \cdot x \cdot !_{\pi+\rho} y = x \cdot !_\pi y \cdot !_\rho y \qquad D \cdot !_1 x = x \qquad \delta_{\pi\rho} \cdot !_{\pi\rho} x = !_\pi !_\rho x$$

$$F_\rho \cdot !_\rho x \cdot !_\rho y = !_\rho (x \cdot y)$$

R-LCAs can be used to give QCwFs for arbitrary semirings R.

6.2 Some concrete LCAs

6.2.1 Graph Models. An example BCI-algebra is given by the restriction of Scott [1976]'s graph model of the untyped λ-calculus to linear application – application where at exactly one use of the input is allowed for every output. This example was originally given by Hoshino [2007]. Let A be the set inductively defined by the following rules:

$$\frac{}{\bullet \in A} \qquad \frac{a \in A \qquad b \in A}{\langle a, b\rangle \in A} \qquad \frac{n \in \mathbb{N}}{n \in A}$$

Then the power set of A, $\mathcal{P}(A)$ has the structure of an LCA. Define linear application as follows:

$$p \cdot q = \{b \mid \exists a.\ \langle a, b\rangle \in p, a \in q\}$$

The B, C, I combinators are defined as follows:

$$\begin{array}{lcl} B & = & \{\langle\langle b, c\rangle, \langle\langle a, b\rangle, \langle a, c\rangle\rangle\rangle \mid a, b, c \in A\} \\ C & = & \{\langle\langle b, \langle a, c\rangle\rangle, \langle a, \langle b, c\rangle\rangle\rangle \mid a, b, c \in A\} \\ I & = & \{\langle a, a\rangle \mid a \in A\} \end{array}$$

To define exponentiation, let us write $[a_1, \cdots, a_n]$ to stand for a representation of lists as right-nested tuples: $\langle a_1, \cdots \langle a_n, \bullet\rangle \cdots\rangle$. We (partially) define the append of two elements of A as usual:

$$\begin{array}{rcl} \langle a, a'\rangle +\!\!+ a'' & = & \langle a, a' +\!\!+ a''\rangle \\ \bullet +\!\!+ a'' & = & a'' \end{array}$$

where $n +\!\!+ a$ is undefined. Now we define

$$!p = \{[a_1, \cdots, a_n] \mid a_1, \ldots, a_n \in p\}$$

and

$$\begin{array}{lcl} K & = & \{\langle a, \langle [], a\rangle\rangle \mid a \in A\} \\ W & = & \{\langle\langle a_1, \langle a_2, b\rangle\rangle, \langle a_1 +\!\!+ a_2, b\rangle\rangle \mid a_1, a_2, b \in A\} \\ D & = & \{\langle [a], a\rangle \mid a \in A\} \\ \delta & = & \{\langle a_1 +\!\!+ \cdots +\!\!+ a_n, [a_1, \ldots, a_n]\rangle \mid a_1, \ldots, a_n \in A\} \\ F & = & \{\langle [\langle a_1, b_1\rangle, \ldots, \langle a_n, b_n\rangle], \langle [a_1, \ldots, a_n], [b_1, \ldots, b_n]\rangle\rangle \mid a_i, b_i \in A\} \end{array}$$

Proposition 6.3. *The structure $(\mathcal{P}(A), (\cdot), B, C, I, !, K, W, D, \delta, F)$ is an LCA.*

Proposition 6.4. *The LCA $(\mathcal{P}(A), (\cdot), B, C, I, \ldots)$ supports conditionals. There are elements $T, F \in \mathcal{P}(A)$ and a function $E : \mathcal{P}(A) \times \mathcal{P}(A) \to \mathcal{P}(A)$ such that $E(p, q) \cdot T = p$ and $E(p, q) \cdot F = q$.*

6.2.2 A Geometry of Interaction Model. As shown by Abramsky et al. [2002], LCAs can be derived from *Geometry of Interaction situations*. For lack of space, we do not describe their construction in detail. A simple example of an LCA derived from a GoI situation is given by the set of partial functions on the natural numbers. We think of these functions as memoryless computational units that take natural numbers as input and produce natural numbers as output (or fail). Abramsky et al. show how to define an application operation in terms of a feedback operation and multiplexing. Exponentials (!) are interpreted by $\mathbb{N}$-wide multiplexing. The interest in these models lies in the range of computational situations they can model. Abramsky et al. [2002] give functional, relational,

game, and stochastic versions, and there are applications actual hardware circuits [Ghica 2007] and effectful situations [Hoshino et al. 2014]. A small problem is that not all instances of GoI situations are able to model the boolean type. For example, the requirements of Definition 6.1 do not appear to be satisfiable in the partial injective functions GoI situation, though they are in the partial function and relational situations.

6.2.3 The Linear λ-Calculus. Simpson [2005] defined the untyped linear λ-calculus:

$$e \quad ::= \quad x \mid e_1 e_2 \mid \lambda x.e \mid \lambda^! x.e \mid !e$$

with the constraint that a linearly bound variable $\lambda x.e$ must appear precisely once, and not within a !. The collection of linear λ-terms, quotiented by β equality, is an LCA. The combinators are defined as the linear λ-terms directly implementing the required equation for each combinator. The untyped linear λ-calculus therefore gives a way of compiling QTT, via our interpretation in a QCwF, to this syntactic model of linear computation (provided it is extended with linear booleans as in Definition 6.1).

7 RELATED WORK

In our introductory section we have already described the main predecessors of this work. In the split context tradition, where intuitionistic and linear types are kept separate, the original work is Cervesato and Pfenning [2002]'s Linear Logical Framework, which Vákár [2015] has provided a categorical semantics for; Krishnaswami et al. [2015] also investigated a variant of the split context system, also based on Benton [1994]'s Linear Non-Linear Logic. It would be interesting to compare our categorical semantics with Vákár's indexed symmetric monoidal category semantics. We suspect that QCwFs are more specific in that every QCwF (augmented with some kind of $\otimes$-product type former) forms one of Vákár's indexed symmetric monoidal categories.

Another approach to linear dependent type theory has recently been proposed by Luo and Zhang [2016], where linear and intuitionistic variables are mixed within the same context, and specialised context merging operations are used to distinguish between extensional and intensional uses. We have not yet closely examined the relationship between their system and QTT.

McBride [2016]'s system, of which Quantitative Type Theory is a variant, crucially uses usage annotations on types. Such annotations have already appeared in the work of Petricek et al. [2014] and Brunel et al. [2014] on *coeffects*, where they are used to track information about the *context* in which a computation occurs. It would be interesting to compare the concrete models they give (in particular, Brunel et al. [2014] use a realisability model) to ours. Ghica and Smith [2014] particularly emphasises the quantitative nature of the annotations, and presents an application to timing information.

We have used Abramsky et al. [2002]'s notion of Linear Combinatory Algebra as our notion of intensional computational information to be associated with terms. Their Geometry of Interaction situations provide a large supply of LCAs, however not all of them have the necessary structure to interpret the kind of boolean type that we have included in our calculus, as we saw in Section 6.2.2. This appears to be related to the problem of interpretating the additive connectives of Linear Logic in Abramsky et al. [2002]'s axiomatic approach. Nevertheless, the connection with Geometry of Interaction style models is intriguing, and may help develop the theory of Quantitative Type Theory into areas such as reversible computation [Abramsky 2005] and hardware synthesis [Ghica 2007].

8 CONCLUSIONS AND FUTURE WORK

We have presented a reformulation of McBride [2016]'s system combining linear and dependent types that fixes the inadmissibility of substitution in that system. We have defined a sound class of

categorical models for the system, *Quantitative Categories with Families*, and shown that there exist instances of these models that interpret the precise intensional usage information non trivially, using Linear Combinatory Algebras. This work forms a basis on which to further explore the extensions and applications of Quantitative Type Theory, its implementation, and additional models.

More Type Formers. In this work, we have extend McBride's original system to include boolean types and an explicit universe type. However, there are many more type formers that we could consider. From simply typed Linear Logic, there are the multiplicative ($\otimes$) and additive ($\times$) variants of product types, which ought to have dependent version. Internalisation of usage annotations via an exponential type $!_\pi S$ should also be investigated. More interesting are recursive types such as the natural numbers, and equality types. Recursive types allow arbitrary iteration, which needs to be accounted for in usage annotations. Equality types are interesting because they emphasise the difference between the extensional and intensional content of terms.

Application: Implicit computational complexity. A major theoretical application of linear types has been to implicit computational complexity, where complexity classes have been captured in terms of specific linear type systems [Lago 2011]. QTT offers the intriguing possibility of a type system where *terms* are restricted to polynomial time computation (for instance), but arbitrary time computation is permitted in types. The realisability models of implicit computational complexity presented by Lago and Hofmann [2011] seem to offer a good starting point for finding QCwFs that model restricted complexity computing.

Application: Staged Computation, Information Flow. The tropical semiring $(\mathbb{N} \cup \{\infty\}, \min, \infty, +, 0)$ appears to offer a way to type staged computation in QTT by marking each variable with the "stage" it will be available at in the future. The 0 stage represents "now", and ∞ represents the stage, unreachable at runtime, where types reside. It may be possible to construct a dependent version of Atkey and McBride [2013]'s system of clocks for guarded recursion and coprogramming. A generalisation of this semiring to arbitrary "security levels" may yield a dependent Type Theory for secure information flow.

Application: Imperative in-place update. Finally, we wish to apply QTT to imperative computation. Following Ahmed et al. [2007]'s *Linear Language with Locations*, we plan to investigate models of QTT that are realised by imperative programs that manipulate pointer data structures. Since we are embedded within a full dependent Type Theory, we have the possibility of combining imperative programming with reasoning and generic programming in a smooth way.

ACKNOWLEDGMENTS

REFERENCES

Samson Abramsky. 2005. A structural approach to reversible computation. *Theor. Comput. Sci.* 347, 3 (2005), 441–464. https://doi.org/10.1016/j.tcs.2005.07.002

Samson Abramsky, Esfandiar Haghverdi, and Philip J. Scott. 2002. Geometry of Interaction and Linear Combinatory Algebras. *Mathematical Structures in Computer Science* 12, 5 (2002), 625–665. https://doi.org/10.1017/S0960129502003730

Amal Ahmed, Matthew Fluet, and Greg Morrisett. 2007. L^3: A Linear Language with Locations. *Fundam. Inform.* 77, 4 (2007), 397–449. http://content.iospress.com/articles/fundamenta-informaticae/fi77-4-06

Robert Atkey and Conor McBride. 2013. Productive coprogramming with guarded recursion. In *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, Greg Morrisett and Tarmo Uustalu (Eds.). ACM, 197–208. https://doi.org/10.1145/2500365.2500597

Henry G. Baker. 1992. Lively Linear Lisp–'Look Ma, No Garbage!'. *ACM Sigplan notices* 27, 8 (1992), 89–98.

Andrew Barber. 1996. *Dual Intuitionistic Linear Logic.* Technical Report ECS-LFCS-96-347. LFCS, University of Edinburgh.

P. N. Benton. 1994. A Mixed Linear and Non-Linear Logic: Proofs, Terms and Models (Extended Abstract). In *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers (Lecture Notes*

in Computer Science), Leszek Pacholski and Jerzy Tiuryn (Eds.), Vol. 933. Springer, 121–135. https://doi.org/10.1007/BFb0022251

Edwin Brady. 2013. Idris, a general-purpose dependently typed programming language: Design and implementation. *J. Funct. Program.* 23, 5 (2013), 552–593. https://doi.org/10.1017/S095679681300018X

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

Aloïs Brunel, Marco Gaboardi, Damiano Mazza, and Steve Zdancewic. 2014. A Core Quantitative Coeffect Calculus. In *Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*. 351–370. https://doi.org/10.1007/978-3-642-54833-8_19

Iliano Cervesato and Frank Pfenning. 2002. A Linear Logical Framework. *Inf. Comput.* 179, 1 (2002), 19–75. https://doi.org/10.1006/inco.2001.2951

Peter Dybjer. 1996. Internal Type Theory. In *Types for Proofs and Programs, International Workshop TYPES'95, Torino, Italy, June 5-8, 1995, Selected Papers (Lecture Notes in Computer Science)*, Stefano Berardi and Mario Coppo (Eds.), Vol. 1158. Springer, 120–134.

Dan R. Ghica. 2007. Geometry of synthesis: a structured approach to VLSI design. In *POPL*. ACM, 363–375.

Dan R. Ghica and Alex I. Smith. 2014. Bounded Linear Types in a Resource Semiring. In *Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*. 331–350. https://doi.org/10.1007/978-3-642-54833-8_18

Jean-Yves Girard. 1987. Linear Logic. *Theoretical Computer Science* 50 (1987), 1–101.

Jean-Yves Girard. 1989. Geometry of Interaction I: Interpretation of System F. In *Logic Colloqium '88*, R. Ferro *et al* (Ed.). North Holland.

Martin Hofmann. 1997. Syntax and Semantics of Dependent Types. In *Semantics and Logics of Computation*. Cambridge University Press, 79–130.

Naohiko Hoshino. 2007. Linear Realizability. In *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*. 420–434. https://doi.org/10.1007/978-3-540-74915-8_32

Naohiko Hoshino, Koko Muroya, and Ichiro Hasuo. 2014. Memoryful geometry of interaction: from coalgebraic components to algebraic effects. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, Thomas A. Henzinger and Dale Miller (Eds.). ACM, 52:1–52:10. https://doi.org/10.1145/2603088.2603124

Neelakantan R. Krishnaswami, Pierre Pradic, and Nick Benton. 2015. Integrating Linear and Dependent Types. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*. 17–30. https://doi.org/10.1145/2676726.2676969

Ugo Dal Lago. 2011. A Short Introduction to Implicit Computational Complexity. In *Lectures on Logic and Computation - ESSLLI 2010 Copenhagen, Denmark, August 2010, ESSLLI 2011, Ljubljana, Slovenia, August 2011, Selected Lecture Notes (Lecture Notes in Computer Science)*, Nick Bezhanishvili and Valentin Goranko (Eds.), Vol. 7388. Springer, 89–109. https://doi.org/10.1007/978-3-642-31485-8_3

Ugo Dal Lago and Martin Hofmann. 2011. Realizability models and implicit complexity. *Theor. Comput. Sci.* 412, 20 (2011), 2029–2047. https://doi.org/10.1016/j.tcs.2010.12.025

John Longley and Dag Normann. 2015. *Higher-Order Computability*. Springer. https://doi.org/10.1007/978-3-662-47992-6

Z. Luo and Y. Zhang. 2016. A Linear Dependent Type Theory. In *TYPES 2016*.

Saunders Mac Lane. 1998. *Categories for the Working Mathematician* (2nd ed.). Number 5 in Graduate Texts in Mathematics. Springer-Verlag.

The Coq development team. 2017. *The Coq proof assistant reference manual*. LogiCal Project. http://coq.inria.fr Version 8.6.

Conor McBride. 2016. I Got Plenty o' Nuttin'. In *A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*. 207–233. https://doi.org/10.1007/978-3-319-30936-1_12

Alexandre Miquel. 2001. The Implicit Calculus of Constructions. In *TLCA*. 344–359. https://doi.org/10.1007/3-540-45413-6_27

Nathan Mishra-Linger and Tim Sheard. 2008. Erasure and Polymorphism in Pure Type Systems. In *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29 - April 6, 2008. Proceedings (Lecture Notes in Computer Science)*, Roberto M. Amadio (Ed.), Vol. 4962. Springer, 350–364. https://doi.org/10.1007/978-3-540-78499-9_25

Torben Æ. Mogensen. 1997. Types for 0, 1 or Many Uses. In *Implementation of Functional Languages, 9th International Workshop, IFL'97, St. Andrews, Scotland, UK, September 10-12, 1997, Selected Papers (Lecture Notes in Computer Science)*, Chris Clack, Kevin Hammond, and Antony J. T. Davie (Eds.), Vol. 1467. Springer, 112–122. https://doi.org/10.1007/BFb0055427

Tomas Petricek, Dominic A. Orchard, and Alan Mycroft. 2014. Coeffects: a calculus of context-dependent computation. In *Proceedings of the 19th ACM SIGPLAN international conference on Functional programming, Gothenburg, Sweden, September 1-3, 2014*, Johan Jeuring and Manuel M. T. Chakravarty (Eds.). ACM, 123–135. https://doi.org/10.1145/2628136.2628160

Frank Pfenning. 2001. Intensionality, Extensionality, and Proof Irrelevance in Modal Type Theory. In *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*. IEEE Computer Society, 221–230. https://doi.org/10.1109/LICS.2001.932499

Dana S. Scott. 1976. Data Types as Lattices. *SIAM J. Comput.* 5, 3 (1976), 522–587. https://doi.org/10.1137/0205037

Alex K. Simpson. 2005. Reduction in a Linear Lambda-Calculus with Applications to Operational Semantics. In *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings (Lecture Notes in Computer Science)*, Jürgen Giesl (Ed.), Vol. 3467. Springer, 219–234. https://doi.org/10.1007/978-3-540-32033-3_17

Kazushige Terui. 2001. Light Affine Calculus and Polytime Strong Normalization. In *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*. 209–220. https://doi.org/10.1109/LICS.2001.932498

The Agda Team. 2017. (2017). http://wiki.portal.chalmers.se/agda.

Matthijs Vákár. 2015. A Categorical Semantics for Linear Logical Frameworks. In *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings (Lecture Notes in Computer Science)*, Andrew M. Pitts (Ed.), Vol. 9034. Springer, 102–116. https://doi.org/10.1007/978-3-662-46678-0_7

Philip Wadler. 1990. Linear types can change the world!. In *Programming Concepts and Methods*, M. Broy and C. Jones (Eds.). North Holland, Amsterdam.