Syntax and Semantics of Quantitative Type Theory

Robert Atkey
University of Strathclyde
robert.atkey@strath.ac.uk

Abstract

We present Quantitative Type Theory, a Type Theory that records usage information for each variable in a judgement, based on a previous system by McBride. The usage information is used to give a realizability semantics using a variant of Linear Combinatory Algebras, refining the usual realizability semantics of Type Theory by accurately tracking resource behaviour. We define the semantics in terms of Quantitative Categories with Families, a novel extension of Categories with Families for modelling resource sensitive type theories.

CCS Concepts • **Theory of computation → Linear logic**; **Type theory**;

Keywords Type Theory, Linear Logic

ACM Reference Format:
Robert Atkey. 2018. Syntax and Semantics of Quantitative Type Theory. In *LICS '18: 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, July 9–12, 2018, Oxford, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3209108.3209189

1 Introduction

Dependent Type Theory promises to combine "programming" and "verification" by combining both in a single system. The implementations Agda [34] and Idris [7] advertise themselves as a "dependently typed functional programming language" and a "general purpose pure functional programming language with dependent types", respectively. Coq [24] is primarily intended as a proof assistant, but also provides a program extraction facility.

However, when trying to actually use Type Theory as a general purpose programming language, type dependency appears to actively encourage inefficient code. This is caused by Type Theory's use of variables for two purposes: 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 as computational information that is manipulated by programs at runtime. An example is illustrated by the type of the *cons* operation for length indexed vectors of Ss:

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

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!

Related to the problem of distinguishing between type formation and computational use of data is distinguishing between different kinds of computational 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). Linear Logic [14] initiated a body of research into such systems. Initially, this recorded zero, single, or multiple uses (made explicit by Mogensen [28]), but recent work in coeffects and quantitative types by Petricek *et al.* [29], Brunel *et al.*[8], and Ghica and Smith [13] refined this to track usage via semirings. However, extending these systems to *dependent* Type Theory is not straightforward due to the conflict between type formation and computational uses.

McBride [25] has recently proposed a resolution to this conflict by combining the work on erasability and quantitative types. 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 fix and extend McBride's system, and present semantic interpretations that fully exploit the usage information.

Contributions

1. Section 2 reformulates McBride's system as *Quantitative Type Theory* (QTT) to add dependent tensor products and booleans, and to fix a bug in the original system that caused substitution to be inadmissible.
2. Section 3 presents *Quantitative Categories with Families*, a novel class of categorical models for interpreting QTT.
3. Section 4 presents several concrete realisability models of QTT as instances of QCwFs. These demonstrate that QTT allows resource sensitive interpretation of terms. Read constructively, these interpretations yield an efficient extraction mechanism with precise control over resource usage.

2 Quantitative Type Theory

Combining dependency with linearity is not straightforward. We motivate McBride's solution and our formulation of it.

2.1 Dependency and Accountancy

In Martin-Löf Type Theory, the term judgement has the 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. It describes the names used for resources which may be used by M to construct a resource of type T. It also describes the names used to refer to the extensional meanings represented by those resources used to

ACM ISBN 978-1-4503-5583-4/18/07…$15.00
https://doi.org/10.1145/3209108.3209189

form the types $S_2, \ldots, S_n$ and T. This dual usage is illustrated in 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 a resource that is transferred from the input to the output; and n is used to define the type of x. The fact that n is not used in the computation being described is not explicitly recorded.

Linear Logic [14] uses presence or absence in a context to explicitly record used resources. A linear typing judgement,

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

indicates that the term M uses the resources named by the x_i each precisely once.

Linear Logic's resource accounting via presence conflicts with type dependency. If a variable's referenced resources are not used computationally, it must not appear in the context and so is not available for use in type formation. Judgement 1 is thus not linear: both because n is not used in the term, and because it is used twice in types.

To resolve this conflict, several authors have used formulations of Linear Logic that distinguish between unrestricted ("intuitionistic") variables and restricted ("linear") variables. For simple types, this originates in Barber' work [5], where judgements have two contexts 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_i may be used without restriction, zero, one, or many times. The variables y_i must be used exactly once. Thus, the judgement tracks information about the two different kinds of usage. Cervesato and Pfenning [9], and later Krishnaswami *et al.* [18] and Vákár [35] adapted this 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_i. If we wish to treat elements of $\mathsf{Fin}(n)$ as non-duplicable resources, 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, for example:

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

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.

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. This 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.* [8], Ghica and Smith [13], and Petricek *et al.* [29] (the general idea of marking variable bindings rather than their types was originally called "discharged assumptions" by Terui [33]). In all these systems, information about how variables are used is recorded using a 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 in this paper, a judgement in these systems looks like:

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

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: $x \overset{0}{:} S$ is a complicated way of stating that x may as well be absent.

However, when we move to dependent types, 0 usage variables have a useful meaning. McBride [25] reads the usage annotations ρ_i as indications of computational usage, so a variable with usage 0 indicates that it has no "run-time" presence, but may still be used in the formation of types. The properties of semirings 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. Thus McBride's system incorporates both erasure (see also Miquel [26] and Mishra-Linger and Sheard [27]) with linearity.

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 annotation means that we can construct terms that are only to be used in the formation of types. McBride allowed arbitrary usages ρ on the final colon. However, this yields a system that does not admit substitution as we show in Section 2.3.

We now introduce our formulation of McBride's system, extending it with a data type of booleans, and dependent tensor product types. McBride presented a bidirectional system suited for implementation, and did not distinguish between proper types and terms representing types. We are interested in a semantical investigation of the theory and so switch to a system with an explicit separation of types and representations of types, and with a "declarative" presentation of the rules.

2.1.1 Semirings for Usage Accounting

A *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 $1, 0 \in R$, such that $(R, +, 0)$ is a commutative monoid, $(R, \cdot, 1)$ is a monoid, $(+)$ and $(\cdot)$ distribute, and $0\rho = \rho 0 = 0$, for all $\rho \in R$. We also require that the semiring R is *positive*, meaning that $\rho + \pi = 0$ implies $\rho = 0$ and $\pi = 0$, and has the zero-product property: $\rho\pi = 0$ implies $\rho = 0$ or $\pi = 0$. These latter two properties are required for the admissibility of substitution.

Suitable semirings include the zero-one-many semiring $\{0, 1, \omega\}$, where $\rho + \omega = \omega$ and $\omega \cdot \omega = \omega$; the natural numbers with addition and multiplication; and the boolean semiring $\{0, 1\}$, which we can read as "erased" and "present".

2.1.2 Presyntax

Quantitative Type Theory (QTT) is defined over the syntactic categories of usages, ρ, π, precontexts Γ, pretypes S, T, R, and preterms

M, N, O:

$$\begin{array}{lll}
\rho, \pi & \in & R \\
\Gamma & ::= & \diamond \mid \Gamma, x \overset{\rho}{:} S \\
S, T, U & ::= & (x \overset{\pi}{:} S) \to T \mid (x \overset{\pi}{:} S) \otimes T \mid I \mid \mathsf{Bool} \mid \mathsf{El}(M) \mid \mathsf{Set} \\
M, N, O & ::= & x \mid \lambda x \overset{\pi}{:} S.M^T \mid \mathrm{App}_{(x \overset{\pi}{:} S)T}(M, N) \\
 & & \mid (M, N)_{x \overset{\pi}{:} S.T} \mid * \mid \mathrm{fst}_{x \overset{\pi}{:} S.T}(M) \mid \mathrm{snd}_{x \overset{\pi}{:} S.T}(M) \\
 & & \mid \mathrm{let}_{x \overset{\pi}{:} S.T}(x, y) = M \text{ in } N \mid \mathrm{let}_I * = M \text{ in } N \\
 & & \mid \mathrm{true} \mid \mathrm{false} \mid \mathrm{ElimBool}_{(z)T}(M_t, M_f, N) \\
 & & \mid \mathrm{Bool} \mid (x \overset{\pi}{:} M) \to N
\end{array}$$

The rules that identify the contexts, types, and terms from the precontexts, pretypes, and preterms are described below. The symbol $\diamond$ denotes the empty precontext, which we omit when writing precontexts with more than zero elements. 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 3.

Precontexts contain usage annotations, ρ, on each of the constituent variables. Scaling, $\pi\Gamma$, is defined by:

$$\pi(\diamond) = \diamond \qquad\qquad \pi(\Gamma, x \overset{\rho}{:} S) = \pi\Gamma, x \overset{\pi\rho}{:} S$$

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 to 0.

Precontext addition $\Gamma_1 + \Gamma_2$ is only defined when $0\Gamma_1 = 0\Gamma_2$:

$$\diamond + \diamond = \diamond \qquad (\Gamma_1, x \overset{\rho_1}{:} S) + (\Gamma_2, x \overset{\rho_2}{:} S) = (\Gamma_1 + \Gamma_2), x \overset{\rho_1+\rho_2}{:} S$$

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

2.1.3 Contexts, Types, Terms

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

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

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 Emp builds the empty context. The rule Ext extends a context Γ with a extra variable x of type S, with usage annotation ρ. The annotation ρ indicates building of 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 no computational resources.

Term judgements in QTT have the 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 term with no need for computational content. By Lemma 2.3 (below) when $\sigma = 0$ we will necessarily have that all the usage annotations in the context are 0 too. When $\sigma = 1$, we are indicating we are constructing computationally relevant data. The two halves are similar to Pfenning's ":" and "÷" judgements for terms and irrelevant proofs respectively [30].

The rules for variables and type conversion are:

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

The variable rule, 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, and the selected variable is marked with the result usage σ. The conversion rule, 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 always judged equal in a context with no resources.

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

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

The usage annotation π is not used when judging that T is a type. As for all type well formedness judgements in QTT, 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.

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

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

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 variables with the same types). 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. By the last premise, the function argument N may be judged in the 0-use fragment of the system in the case when either we are already in that fragement, or the function will not use the argument ($\pi = 0$). It is this condition on usages that requires us to stipulate that the usage accounting semiring we use is positive and has the zero-product property.

Dependent Tensor Product Types We extend McBride's system with a dependent tensor product type $(x \overset{\pi}{:} S) \otimes T$, where the π annotation records how many times the first argument may be used, and a unit type I:

$$\frac{0\Gamma \vdash A \qquad 0\Gamma, x \overset{0}{:} S \vdash T}{0\Gamma \vdash (x \overset{\pi}{:} S) \otimes T}\ \otimes \qquad\qquad \frac{0\Gamma \vdash}{0\Gamma \vdash I}\ I$$

As for the dependent function type, type formation does not require any intensional resources. The introduction rules:

$$\frac{\begin{array}{c}\Gamma_1 \vdash M \overset{\sigma'}{:} S \qquad 0\Gamma_1 = 0\Gamma_2 \\ \Gamma_2 \vdash N \overset{\sigma}{:} T[M/x] \qquad \sigma' = 0 \Leftrightarrow (\pi = 0 \vee \sigma = 0)\end{array}}{\pi\Gamma_1 + \Gamma_2 \vdash (M, N)_{x \overset{\pi}{:} S.T} \overset{\sigma}{:} (x \overset{\pi}{:} S) \otimes T} \qquad \frac{0\Gamma \vdash}{0\Gamma \vdash * \overset{\sigma}{:} I}$$

Resource-wise, building a pair is similar to the App rule above. To build an element of the unit type requires no resources. When we are in the extensional portion of the theory ($\sigma = 0$), there are effectively no restrictions, and these are the normal introduction rules for dependent product and unit types.

The elimination rules are interesting because there are two sets, depending on whether we are in the erased ($\sigma = 0$) or present ($\sigma = 1$) part of the theory. Under erasure, we are free to use the normal first and second projection operators:

$$\frac{\Gamma \vdash M \overset{0}{:} (x \overset{\pi}{:} S) \otimes T}{\Gamma \vdash \mathrm{fst}_{x \overset{\pi}{:} S.T}(M) \overset{0}{:} A} \qquad \frac{\Gamma \vdash M \overset{0}{:} (x \overset{\pi}{:} S) \otimes T}{\Gamma \vdash \mathrm{snd}_{x \overset{\pi}{:} S.T}(M) \overset{0}{:} T[\mathrm{fst}(M)/x]}$$

We also assume the usual $\beta\eta$ equality rules in the erased fragment. There is no erased eliminator for the I type; in the absence of intensional information, the I type acts like the "unit" type in a normal type theory, and so has no eliminator. We have an η-rule for I in the $\sigma = 0$ fragment stating that $*$ is the only inhabitant.

When we take into account the resource content of objects, we are more restricted in how we can eliminate objects of tensor product and unit type. Just as for elimination of $\otimes$ in intuitionistic linear types theories, we must use a pattern matching construct to ensure that both parts of the product are used. Likewise, we must explicitly eliminate elements of the unit type:

$$\frac{\begin{array}{c}0\Gamma_1, z \overset{0}{:} (x \overset{\pi}{:} S) \otimes T \vdash U \qquad \Gamma_1 \vdash M \overset{\sigma}{:} (x \overset{\pi}{:} S) \otimes T \\ \Gamma_2, x \overset{\sigma\pi}{:} S, y \overset{\sigma}{:} T \vdash N \overset{\sigma}{:} U[(x,y)/z] \qquad 0\Gamma_1 = 0\Gamma_2\end{array}}{\Gamma_1 + \Gamma_2 \vdash \mathrm{let}_{x \overset{\pi}{:} S.T}\ (x,y) = M \text{ in } N \overset{\sigma}{:} U[M/z]}$$

$$\frac{0\Gamma_1, x \overset{0}{:} I \vdash U \qquad \Gamma_1 \vdash M \overset{\sigma}{:} I \qquad \Gamma_2 \vdash N \overset{\sigma}{:} U[*/x] \qquad 0\Gamma_1 = 0\Gamma_2}{\Gamma_1 + \Gamma_2 \vdash \mathrm{let}_I * = M \text{ in } N \overset{\sigma}{:} U[M/x]}$$

These constructs are subject to the usual $\beta\eta$ and commuting conversion rules [5]. These constructs are permitted in the $\sigma = 0$ and $\sigma = 1$ fragments, but in the $\sigma = 0$ fragment they are shorthand for fst and snd:

$$\frac{\begin{array}{c}0\Gamma, z \overset{0}{:} (x \overset{\pi}{:} S) \otimes T \vdash U \\ 0\Gamma \vdash M \overset{0}{:} (x \overset{\pi}{:} S) \otimes T \qquad 0\Gamma, x \overset{0}{:} S, y \overset{0}{:} T \vdash N \overset{0}{:} U[(x,y)/z]\end{array}}{0\Gamma \vdash \mathrm{let}_{x \overset{\pi}{:} S.T}\ (x,y) = M \text{ in } N \equiv N[\mathrm{fst}(M)/x, \mathrm{snd}(M)/y] \overset{0}{:} U[M/z]}$$

By combining $\otimes$ and I, we can derive an exponential type $!_\rho A$ for QTT as $!_\rho A = (x \overset{\rho}{:} A) \otimes I$.

Boolean Type The type Bool is an example of a type of data that can be available at runtime.

$$\frac{0\Gamma \vdash}{0\Gamma \vdash \mathsf{Bool}}\ \textsc{Bool} \qquad \frac{0\Gamma \vdash}{0\Gamma \vdash \mathrm{true}, \mathrm{false} \overset{\sigma}{:} \mathsf{Bool}}\ \text{B-T,F}$$

The introduction rules indicate two pieces of 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 erased or present.

The elimination rule for booleans is:

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

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 typing content ($0\Gamma_1 = 0\Gamma_2$).

Universe: The Type of Small Types The type of small types, Set, is an example of a type whose elements occupy no resource, but do have a role in forming types. The type formation rule for Set is similar to the one for Bool:

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

The introduction rules for Set are only available in the $\sigma = 0$ fragment; we can never construct a Set value with runtime presence. For instance, the terms representing dependent function types:

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

The other introduction rules are similar, but are omitted for space.

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

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

This rules indicates that the use of a code for a type in a type does not require any computational information.

Using Set with our other type formers, we can construct additive product and sum types. For example, $S \times T = (b \overset{1}{:} \mathsf{Bool}) \to \mathrm{ElimBool}_{(\_)\mathsf{Set}}(S, T, b)$.

This concludes our presentation of the context, type and term formation rules of QTT. The remaining rules of QTT concern the type and term equality judgements $\Gamma \vdash S \equiv T$ and $\Gamma \vdash M \equiv N \overset{\sigma}{:} S$, but are omitted for space.

2.2 Syntactic Properties of QTT

Usage annotation manipulation The following shows how usage annotations and the judgements of QTT interact.

Lemma 2.1. *Usage annotations do not affect context well formedness:* $\Gamma_1 \vdash$ *and* $0\Gamma_1 = 0\Gamma_2$ *implies* $\Gamma_2 \vdash$.

Lemma 2.2. *The following* 0*-ing rules are admissible:*

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

These rules state that we can take any term and produce its "resource free" counterpart in the $\sigma = 0$ fragment. Semantically, this will be modelled by the U functor/functions in our QCwFs in Definition 3.2. Note that it is not possible to go the other way: there is no way to reconstruct the resource usage, and some constructs such as fst and snd only live in the $\sigma = 0$ fragment.

Lemma 2.3 (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.

Weakening Weakening allows the insertion of variables into contexts, provided they are 0 usage.

Lemma 2.4. *Weakening is admissible:*

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

where $\mathcal{J}$ *is "is context",* S, $S \equiv T$, $M \overset{\sigma}{:} S$, *or* $M \equiv N \overset{\sigma}{:} S$.

Substitution Substitution (Lemma 2.5) allows for a variable $x \overset{\rho}{:} S$ to be replaced by a term of type S, as long as, in the case of the term judgements, we add the resources it requires multiplied by ρ to the resources already present.

Lemma 2.5. *The following substitution rules are admissible:*

$$\dfrac{\Gamma, x \overset{\rho}{:} S, \Gamma' \vdash \qquad 0\Gamma \vdash N \overset{0}{:} S}{\Gamma, \Gamma'[N/x] \vdash}$$

$$\dfrac{0\Gamma_1, x \overset{0}{:} S, 0\Gamma' \vdash T \qquad \Gamma_2 \vdash N \overset{0}{:} S \qquad 0\Gamma_1 = 0\Gamma_2}{0\Gamma_1, 0\Gamma'[N/x] \vdash T[N/x]}$$

$$\dfrac{0\Gamma_1, x \overset{0}{:} S, 0\Gamma' \vdash T \equiv T' \qquad \Gamma_2 \vdash N \overset{0}{:} S \qquad 0\Gamma_1 = 0\Gamma_2}{0\Gamma_1, 0\Gamma'[N/x] \vdash T[N/x] \equiv T'[N/x]}$$

$$\dfrac{\begin{array}{c}\Gamma_1, x \overset{\rho}{:} S, \Gamma' \vdash M \overset{\sigma}{:} T \\ \Gamma_2 \vdash N \overset{\sigma'}{:} S \qquad 0\Gamma_1 = 0\Gamma_2 \qquad \sigma' = 0 \Leftrightarrow \rho = 0\end{array}}{(\Gamma_1 + \rho\Gamma_2), \Gamma'[N/x] \vdash M[N/x] \overset{\sigma}{:} T[N/x]}$$

$$\dfrac{\begin{array}{c}\Gamma_1, x \overset{\rho}{:} S, \Gamma' \vdash M \equiv M' \overset{\sigma}{:} T \\ \Gamma_2 \vdash N \overset{\sigma'}{:} S \qquad 0\Gamma_1 = 0\Gamma_2 \qquad \sigma' = 0 \Leftrightarrow \rho = 0\end{array}}{(\Gamma_1 + \rho\Gamma_2), \Gamma'[N/x] \vdash M[N/x] \equiv M'[N/x] \overset{\sigma}{:} T[N/x]}$$

2.3 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 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 $\Gamma_1 + \Gamma_2 = \Gamma$. This is not possible in general. We illustrate the problem with:

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

This term coerces some function f from a type that states it uses its argument once to a type where it uses its argument multiple times. This is despite McBride's system not explicitly allowing subusaging.

The following derivation gives a context into which substitution of Judgement 4 fails. Let D_1 and D_2 stand for uses of the Var rule to derive $y \overset{0}{:} A, g \overset{1}{:} (x \overset{\omega}{:} A) \to A \vdash g \overset{1}{:} (x \overset{\omega}{:} A) \to A$ and $y \overset{\omega}{:} A, g \overset{0}{:} (x \overset{\omega}{:} A) \to A \vdash y \overset{\omega}{:} A$.

$$\dfrac{D_1 \qquad \dfrac{\dfrac{}{y \overset{0}{:} A, g \overset{\omega}{:} (x \overset{\omega}{:} A) \to A \vdash g \overset{\omega}{:} (x \overset{\omega}{:} A) \to A} \qquad D_2}{y \overset{\omega}{:} A, g \overset{\omega}{:} (x \overset{\omega}{:} A) \to A \vdash g y \overset{\omega}{:} A}}{y \overset{\omega}{:} A, g \overset{\omega}{:} (x \overset{\omega}{:} A) \to A \vdash g(gy) \overset{1}{:} A}$$

If we attempt to substitute $\lambda x \overset{\omega}{:} A. f x$ 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:

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

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 only allowing 0 or 1 usage on terms, which prohibits this kind of implicit "subusaging". An alternative solution would be to allow subusaging. This is something we plan to investigate.

3 Quantitative Categories with Families

Giving a model for dependent Type Theory is a delicate affair due to the complex mutual definition of contexts, types and terms. We build on the existing notion of Categories with Families (CwFs) [11, 15] to define *Quantitative Categories with Families* (QCwFs) suitable for modelling QTT. Every QCwF contains a CwF, so we first recall their definition.

Definition 3.1. A *Category with Families* (CwF) is a six-tuple $(\mathcal{C}, \mathrm{Ty}, \mathrm{Tm}, \top, -.-, \langle -, - \rangle)$, where:

1. $\mathcal{C}$ is a category with a chosen terminal object $\top$;
2. For $\Delta \in \mathrm{Ob}\mathcal{C}$, a collection $\mathrm{Ty}(\Delta)$ of *semantic types*;
3. For $\Delta \in \mathrm{Ob}\mathcal{C}$ and $S \in \mathrm{Ty}(\Delta)$, a collection $\mathrm{Tm}(\Delta, S)$ of *semantic terms*;
4. For every $f : \Delta \to \Delta'$ in $\mathcal{C}$, a function $-\{f\} : \mathrm{Ty}(\Delta') \to \mathrm{Ty}(\Delta)$, and for $S \in \mathrm{Ty}(\Delta')$ a function $-\{f\} : \mathrm{Tm}(\Delta', S) \to \mathrm{Tm}(\Delta, S\{f\})$, such that both assignments preserve identities and composition.
5. For each object Δ in $\mathcal{C}$ and $S \in \mathrm{Ty}(\Delta)$ 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 from Definition 3.1, and also define the semantic counterpart of a substitution of a term for a variable. We will require usage annotated counterparts of these below.

1. For $\Delta \in \mathrm{Ob}\mathcal{C}$ and $S \in \mathrm{Ty}(\Delta)$, the *first projection* morphism $\mathrm{p}_{\Delta.S} : \Delta.S \to \Delta$ is defined as $\mathrm{p}_{\Delta.S} = \mathrm{id}^{\#1}_{\Delta.S}$.
2. For $\Delta \in \mathcal{C}$ and $S \in \mathrm{Ty}(\Delta)$, the *second projection* $\mathrm{v}_{\Delta.S} \in \mathrm{Tm}(\Delta.S, S\{\mathrm{p}_{\Delta.S}\})$ is defined as $\mathrm{v}_{\Delta.S} = \mathrm{id}^{\#2}_{\Delta.S}$.

3. For any Δ, Δ' in C, $S \in \mathrm{Ty}(\Delta)$ and morphism $f : \Delta' \to \Delta$, the *weakening* of f by S, $\mathrm{wk}(f, S) : \Delta'.S\{f\} \to \Delta.S$ is defined as $\mathrm{wk}(f, S) = \langle f \circ \mathrm{p}_{\Delta'.S\{f\}}, \mathrm{v}_{\Delta'.S\{f\}} \rangle$.
4. For any Δ in C, $S \in \mathrm{Ty}(\Delta)$ and $M \in \mathrm{Tm}(\Delta, S)$, we define the morphism $\overline{M} : \Delta \to \Delta.S$ as $\overline{M} = \langle \mathrm{id}_\Delta, M \rangle$.

The definition of a Quantitative Category with Families incorporates a CwF, and layers over it facilities for representing contexts, simultaneous substitutions and terms, all with usage information.

Definition 3.2. For a usage semiring R, an *R-Quantitative Category with Families* (R-QCwF) consists of the data:

1. A CwF (Definition 3.1) $(C, \mathrm{Ty}, \mathrm{Tm}, 1, -.-, \langle -, - \rangle)$;
2. A category $\mathcal{L}$, whose objects will be used to interpret contexts with resource annotations and whose morphisms will interpret simultaneous substitutions, together with a faithful functor $U : \mathcal{L} \to C$;
3. *(Addition structure)* Let $\mathcal{L} \times_C \mathcal{L}$ be the pullback of $\mathcal{L} \xrightarrow{U} C \xleftarrow{U} \mathcal{L}$. We require a functor $(+) : \mathcal{L} \times_C \mathcal{L} \to \mathcal{L}$ such that $U(\Gamma_1 + \Gamma_2) = U\Gamma_1$ $(= U\Gamma_2)$. Also, there is an object $\diamond \in \mathcal{L}$ such that $U\diamond = \top$.
4. *(Scaling structure)* For $\rho \in R$, there is a functor $\rho(-) : \mathcal{L} \to \mathcal{L}$ such that $U(\rho(-)) = U(-)$.
5. *(Resourced Terms)* For Γ in $\mathcal{L}$ and $S \in \mathrm{Ty}(U\Gamma)$, there is a collection of semantic resourced terms $\mathrm{RTm}(\Gamma, S)$, with injective functions $U_{\Gamma.S} : \mathrm{RTm}(\Gamma, S) \to \mathrm{Tm}(U\Gamma, S)$. For morphisms $f : \Gamma' \to \Gamma$, and types $S \in \mathrm{Ty}(U\Gamma)$, there is a function $-\{f\} : \mathrm{RTm}(\Gamma, S) \to \mathrm{RTm}(\Gamma', S\{f\})$ such that $U(M\{f\}) = (UM)\{Uf\}$.
6. *(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
$$\begin{array}{lcl} \mathsf{emp}_\pi & : & \diamond \to \pi\diamond \\ \mathsf{emp}_+ & : & \diamond \to \diamond + \diamond \\ \mathsf{ext}_\pi & : & \pi\Gamma.(\pi\rho)S \to \pi(\Gamma.\rho S) \\ \mathsf{ext}_+ & : & (\Gamma_1 + \Gamma_2).(\rho_1 + \rho_2)S \to \Gamma_1.\rho_1 S + \Gamma_2.\rho_2 S \end{array}$$
such that $U(\mathsf{emp}_\pi) = U(\mathsf{emp}_+) = \mathrm{id} : \top \to \top$, $U(\mathsf{ext}_\pi) = \mathrm{id}$, and $U(\mathsf{ext}_+) = \mathrm{id}$.
7. Resourced counterparts of the projection, weakening, and term substitution of the underlying CwF:
   a. For Γ and $S \in \mathrm{Ty}(U\Gamma)$, there is a morphism $\mathrm{p}_{\Gamma.S} : \Gamma.0S \to \Gamma$ such that $U(\mathrm{p}_{\Gamma.S}) = \mathrm{p}_{U\Gamma.S}$;
   b. For Γ and $S \in \mathrm{Ty}(U\Gamma)$ there exists an element $\mathrm{v}_{\Gamma.S} \in \mathrm{RTm}(0\Gamma.1S, S\{\mathrm{p}_{U\Gamma.S}\})$ such that $U(\mathrm{v}_{\Gamma.S}) = \mathrm{v}_{U\Gamma.S}$;
   c. For $f : \Gamma \to \Gamma'$, $\rho \in R$, and $S \in \mathrm{Ty}(U\Gamma')$ there is a morphism $\mathrm{wk}(f, \rho S) : \Gamma.\rho S\{Uf\} \to \Gamma'.\rho S$ such that $U(\mathrm{wk}(f, \rho S)) = \mathrm{wk}(Uf, S)$;
   d. For $M \in \mathrm{RTm}(\Gamma_2, S)$ and Γ_1 such that $U\Gamma_1 = U\Gamma_2$, there is a morphism $\overline{\rho M} : \Gamma_1 + \rho\Gamma_2 \to \Gamma_1.\rho S$ such that $U(\overline{\rho M}) = \overline{UM}$.
   e. For $M \in \mathrm{Tm}(U\Gamma, S)$ there is a morphism $\overline{M} : \Gamma \to \Gamma.0S$ such that $U(\overline{M}) = \overline{M}$.

Note that 7d does not imply 7e in this definition. As noted after Lemma 2.2, it is not possible to go from unresourced terms to resourced terms.

The following proposition highlights the fact that the usage information present in QTT/QCwFs is a *refinement* of the usual structure of Type Theory. It is possible to completely ignore this refinement and interpret the system in any model of Type Theory. In Section 4 we present non-trivial models that use realisability information to model the refinements brought about by the usage annotations.

Proposition 3.3 (Trivial QCwFs)**.** *Let R be any usage semiring. Every Category with Families $(C, \mathrm{Ty}, \mathrm{Tm}, 1, -.-, \langle -, - \rangle)$ yields an R-QCwF with $\mathcal{L} = C$ and* $\mathrm{RTm} = \mathrm{Tm}$.

As would be expected, the syntax of QTT also forms a model in the sense of R-QCwFs:

Proposition 3.4 (Syntactic Model)**.** *The syntax of QTT forms a QCwF, where the underlying CwF is built from (equivalence classes of) 0-usage annotated contexts, simultaneous substitutions, types, and terms in the $\sigma = 0$ fragment. Objects of $\mathcal{L}$ are equivalence classes of usage annotated contexts, and* $\mathrm{RTm}(\Gamma, S)$ *consists of equivalence classes of terms in the $\sigma = 1$ fragment.*

Type Formers Definition 3.2 only defines enough structure to interpret the construction of contexts and the Var and Conv rules. We must stipulate the existence of further structure if we wish to interpret the function, tensor product, boolean, and universe types. Each of these follows the same pattern as the definition of QCwFs. We first require the standard structure on the CwF for the type former without resource restrictions, and then we define a corresponding refined structure for resourced semantic terms that maps back to the unresourced version via U. To conserve space, we only do the case for dependent function types in detail.

Definition 3.5. A CwF C *supports dependent products with usage information* if for all $\Delta \in \mathrm{Ob}C$, $S \in \mathrm{Ty}(\Delta)$, $T \in \mathrm{Ty}(\Delta.A)$, and $\pi \in R$, there exists a semantic type $\Pi\pi ST \in \mathrm{Ty}(\Delta)$ with a bijection $\Lambda : \mathrm{Tm}(\Delta.S, T) \cong \mathrm{Tm}(\Delta, \Pi\pi ST)$, all natural in Δ.

For Δ, S and T as in Definition 3.5, and $M \in \mathrm{Tm}(\Delta, \Pi\pi ST)$ and $N \in \mathrm{Tm}(\Delta, S)$, we define $App^{00}_{\Delta,S,T}(M, N) \in \mathrm{Tm}(\Delta, T\{\overline{N}\})$ as $App^{00}_{\Delta,S,T} = (\Lambda^{-1}(M))\{\overline{N}\}$. This is used to interpret application in the $\sigma = 0$ fragment of QTT, the superscript indicating that both the function and argument are in this fragment. To interpret abstraction and application when $\sigma = 1$, we need a further definition:

Definition 3.6. A QCwF $(\mathcal{L}, C, \dots)$ *supports dependent products with usage information* if the CwF C supports dependent products with usage information (Definition 3.5), and there is an bijection $\Lambda^{\mathcal{L}} : \mathrm{RTm}(\Gamma.\pi S, T) \cong \mathrm{RTm}(\Gamma, \Pi\pi ST)$, natural in Γ such that $U \circ \Lambda_{\mathcal{L}} = \Lambda \circ U$ and $U \circ \Lambda^{-1}_{\mathcal{L}} = \Lambda^{-1} \circ U$.

To interpret application in the $\sigma = 1$ fragment, we need two derived operations. When both premises are in the $\sigma = 1$ fragment, we will have $M \in \mathrm{RTm}(\Gamma_1, \Pi\pi ST)$ and $N \in \mathrm{RTm}(\Gamma_2, S)$, such that $U\Gamma_1 = U\Gamma_2$, and we define
$$\begin{array}{l} App^{11}_{\Gamma_1,\Gamma_2,\pi,S,T}(M, N) = \\ \quad \Lambda^{-1}_{\Gamma_1,\pi,S,T}(M)\{\overline{\pi N}\} \in \mathrm{RTm}(\Gamma_1 + \pi\Gamma_2, T\{\overline{UN}\}) \end{array}$$
When the argument is erased we have $\pi = 0$, $M \in \mathrm{RTm}(\Gamma_1, \Pi 0 ST)$ and $N \in \mathrm{Tm}(U\Gamma, S)$, and we define
$$App^{10}_{\Gamma,S,T}(M, N) = \Lambda^{-1}_{\Gamma,0,S,T}(M)\{\overline{N}\} \in \mathrm{RTm}(\Gamma, T\{\overline{UN}\})$$
We have $U(App^{1x}(M, N)) = App^{00}(UM, UN)$, for $x \in \{0, 1\}$, part of the semantic counterpart of Lemma 2.2.

The required structure for the remaining type formers is similar, albeit for the Set type, where we do not require any usage annotation refined counterpart.

Interpretation of QTT in a QCwF We follow Hofmann [15]'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. For contexts with all annotations 0, types, and terms in the $\sigma = 0$ fragment, the interpretation is as it is for unannotated Type Theory. This fragment is self contained by Lemma 2.3.

To interpret contexts with resource annotations, and terms in the $\sigma = 1$ fragment, we use the additional structure. We interpret precontexts with resource annotations, Γ as objects $[\![\Gamma]\!]$ of $\mathcal{L}$ and simultaneously partially define two families of morphisms for splitting and distributing resource annotations, using the emp and ext morphisms of the QCwF.

$$\begin{array}{lcl} \mathsf{s}_{\Gamma_1,\Gamma_2} & : & [\![\Gamma_1 + \Gamma_2]\!] \to [\![\Gamma_1]\!] + [\![\Gamma_2]\!] \\ \mathsf{d}_{\pi,\Gamma} & : & [\![\pi\Gamma]\!] \to \pi[\![\Gamma]\!] \end{array}$$

These morphisms "rearrange" the interpretations of contexts to allow the use of the categorical combinators that interpret terms. In the realisability semantics in Section 4, they will be directly interpreted as rearrangement of resources.

Preterms in the $\sigma = 1$ fragment are partially assigned interpretations with respect to some type assignment Δ (i.e. a precontext without usage annotations). The interpretation $[\![\Delta; M]\!]$ yields a pair (Γ, t) of a precontext with usage annotations matching the variables and types in Δ, and a resourced semantic term t. We write $[\![\lfloor\Gamma\rfloor; M]\!]$ for the partial assignment of resourced semantic terms to preterms such that the usage annotations returned by the interpretation match the precontext Γ. The actual interpretation of preterms is defined by induction over their structure, using $\mathsf{v}_{\Gamma.S}$ and $\mathsf{wk}(f, \rho S)$ to interpret variables and the appropriate structure for each type's introduction and elimination rules.

Soundness of the interpretation is stated as the following theorem, similar to [15]. The challenge in proving this theorem is in showing that weakening and substitution are soundly interpreted in the $\sigma = 1$ fragment. This is accomplished by showing that their interpretations are tracked by the interpretations of weakening and substitution in the underlying CwF.

Theorem 3.7.
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 $\Gamma \vdash M \overset{0}{:} S$, then $[\![\Gamma; M]\!]$ is an element of $\mathrm{Tm}(U[\![\Gamma]\!], [\![\Gamma; S]\!])$;*
5. *If $\Gamma \vdash M \equiv N \overset{0}{:} S$, then $[\![\Gamma; M]\!] = [\![\Gamma; N]\!]$.*
6. *If $\Gamma \vdash M \overset{1}{:} S$, then $[\![\lfloor\Gamma\rfloor; M]\!]$ is an element of $\mathrm{RTm}([\![\Gamma]\!], [\![\Gamma; S]\!])$;*
7. *If $\Gamma \vdash M \equiv N \overset{1}{:} S$, then $[\![\lfloor\Gamma\rfloor; M]\!] = [\![\lfloor\Gamma\rfloor; N]\!]$.*

Comparing R-QCwFs to other models Vákár [35] gives a categorical semantics for linear dependent Type Theory with split contexts as we described in Section 2.1. His models consist of indexed symmetric monoidal closed categories $H : \mathcal{C}^{\mathrm{op}} \to \mathrm{SMCCat}$ with a comprehension structure. In contrast to QCwFs, Vákár's models permit more freedom in choosing how to interpret linear types and linearity; there is no requirement that every construct in the linear portion erases to a non-linear representative, as QCwFs require via the U functor. QCwFs are intended to model a situation like QTT where we wish to remain with normal Type Theory for the purposes of reasoning at the type level, and allow unrestricted appearance of terms in types, but we also wish to be able to give resource-aware realisations of programs. We substantiate this claim in the next section with realisability models of QTT. We suspect that QCwFs are more specific in that every QCwF (augmented with a $\otimes$-product type former) forms one of Vákár's indexed symmetric monoidal categories.

4 Realisability Models of QTT

In our informal descriptions of QTT and QCwFs, we have distinguished between the erased and present, or resourced and unresourced fragments of the system. We make this distinction formal by constructing realisability models of QTT. Resource sensitive computational content will be represented in the model by realisers from a R-Linear Combinatory Algebras, a refinement of Abramsky, Haghverdi and Scott's *Linear Combinatory Algebras* [3].

4.1 R-Linear Combinatory Algebras

R-Linear Combinatory Algebras (R-LCAs) are extensions of *BCI*-algebras:

Definition 4.1. A BCI-algebra $(\mathcal{A}, (\cdot), B, C, I)$ comprises a set $\mathcal{A}$, a binary operation $(\cdot)$ (called "application", and written left associatively), and $B, C, I \in \mathcal{A}$, such that $B \cdot x \cdot y \cdot z = x \cdot (y \cdot z)$, $C \cdot x \cdot y \cdot z = x \cdot z \cdot y$, and $I \cdot x = x$.

BCI-algebras 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 connected to one another via application. The B, C, I combinators are a basis for linear implication, $\multimap$, allowing composition (B), exchange (C), and identity (I).

BCI-algebras are *combinatorily complete*: given an expression M with variables and the combinators that contains a variable x exactly once, there is an expression $\lambda^* x.M$ that does not contain x, such that $(\lambda^* x.M) \cdot N = M[N/x]$. Using this fact, we define an n-ary tupling operation:

Definition 4.2. Let $p_1, \dots, p_n$ be elements of a *BCI*-algebra. Define their tupling $[p_1, \dots, p_n] = \lambda^* q.\ q\, p_1\ \dots\ p_n$. We use the notation let $[x_1, \dots, x_n] = p$ in q for $p\ (\lambda^* x_1. \dots \lambda^* x_n.q)$ to represent pattern matching decomposition of tuples.

Example 4.3 (Graph Models)**.** Let D be a set isomorphic to $\mathbb{N} + D \times D$ (for example, finite binary trees with natural numbers at the leaves). We elide the injections into D and just write n for a natural number in D and $\langle a, b\rangle$ for an element of D made from a pair of elements of D. The powerset of D, $\mathcal{P}(D)$, can be endowed with the structure of a *BCI*-algebra with booleans. Define application to be:

$$\alpha \cdot \beta = \{b \mid \exists a.\ \langle a, b\rangle \in \alpha, a \in \beta\}$$

Application interprets α as a set of pairs describing the graph of a function. Each pair in the graph can make a single observation of β to produce a single output observation. This is linear application; compare to Scott's $\mathcal{P}(\omega)$ model that allows a finite but arbitrary number of observations of the input [31]. The B, C, and I combinators are defined by the appropriate graphs:

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

Example 4.4 (Geometry of Interaction)**.** Abramsky *et al.*'s *Geometry of Interaction* (GoI) situations supply a large range of *BCI*-algebras. For concreteness, we explain the *partial functions* GoI situation in detail. Let D be as in Example 4.3. The set of partial functions $D \rightharpoonup D$ can be endowed with the structure of a *BCI*-algebra. We think of elements of this algebra as memoryless computational devices that input and (possibly) output messages represented as elements of D. Define application to be:

$$p \cdot q = p_{00} \cup (p_{01}; q; (p_{11}; q)^*; p_{10})$$

where we treat partial functions as subsets of $D \times D$, $p_{ij} = \{(a, b) \mid (\langle i, a\rangle, \langle j, b\rangle) \in p\}$, and we use the standard combinators for relations. Conceptually, we are thinking of p as having two ports, 0 and 1, that are connected to the environment and to q, respectively. Messages come in the 0 port and are either responded to (p_{00}) or are forwarded to q via p_{01}, which may result in a dialogue, $(p_{11}; q)^*$, before being forwarded back to the environment by p_{10}. We exploit the fact that we can tag messages in D with 0 or 1 to encode the multiplexing of p's messages. The combinator $I = \{(\langle 0, x\rangle, \langle 1, x\rangle) \mid x \in D\} \cup \{(\langle 1, x\rangle, \langle 0, x\rangle) \mid x \in D\}$ – it forwards messages from the environment to its argument, and from its argument to the environment. The B and C combinators do similar, if more involved, routing. Abramsky notes [2] that the relations involved for the *BCI* combinators are always symmetric, yielding a model of *reversible* computation. However, we will see below in Example 4.12 that the structure required for the boolean type will break this reversibility.

Another GoI situation that is suitable for modelling QTT is the resumptions model presented by Abramsky *et al.* [3]. In this model, the combinators are no longer necessarily memoryless, and can update their behaviour based on previous inputs. Abramsky has drawn attention to the relationship between this model and concurrency models [1]. Hoshino *et al.* [17] have used the "memoryful" aspect of this situation to further allow for effectful computations. An exciting avenue of future research will be to examine the implications of this model for an effectful QTT.

Definition 4.5 (R-Linear Combinatory Algebra)**.** Let R be a usage semiring. An *R-Linear Combinatory Algebra* (R-LCA) is a BCI-algebra $(\mathcal{A}, (\cdot), B, C, I)$ with a 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:

$$\begin{array}{lcl} K \cdot x \cdot !_0 y &=& x \\ W_{\pi\rho} \cdot x \cdot !_{\pi+\rho} y &=& x \cdot !_\pi y \cdot !_\rho y \\ D \cdot !_1 x &=& x \end{array} \qquad \begin{array}{lcl} \delta_{\pi\rho} \cdot !_{\pi\rho} x &=& !_\pi !_\rho x \\ F_\rho \cdot !_\rho x \cdot !_\rho y &=& !_\rho (x \cdot y) \end{array}$$

We also require that, for all $x, y \in \mathcal{A}$, $!_0 x = !_0 y$.

The condition that $!_0 x = !_0 y$ ensures that we have a canonical representation of erased data in our models. Abramsky *et al.*'s original LCAs correspond to R-LCAs for the trivial one-element semiring, without this condition. Given an LCA, we can build an R-LCA for the $R = \{0, 1, \omega\}$ semiring:

Proposition 4.6. *Let $(\mathcal{A}, (\cdot), B, C, I, !, K, W, D, \delta, F)$ be an LCA. We can construct a $\{0, 1, \omega\}$-LCA over $(\mathcal{A}, (\cdot), B, C, I)$ with $!_0 x = I$, $!_1 x = x$ and $!_\omega x = !x$.*

Example 4.7. In the graph model of Example 4.3, let us write $[a_1, \cdots, a_n]$ to stand for a representation of lists as right-nested tuples: $\langle a_1, \cdots \langle a_n, 0\rangle \cdots \rangle$. We (partially) define the appending of two elements of D as:

$$\begin{array}{rcl} \langle a, a'\rangle +\!\!+\, a'' &=& \langle a, a' +\!\!+\, a''\rangle \\ 0 +\!\!+\, a'' &=& a'' \end{array}$$

where $n +\!\!+\, a$ is undefined for $n \neq 0$. We similarly define a list membership predicate $a \in b$. Now we define

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

and

$$\begin{array}{rcl} K &=& \{\langle a, \langle [], a\rangle\rangle \mid a \in A\} \\ W &=& \{\langle\langle a_1, \langle a_2, b\rangle\rangle, \langle a_1 +\!\!+\, a_2, b\rangle\rangle \mid a_1, a_2, b \in A\} \\ D &=& \{\langle [a], a\rangle \mid a \in A\} \\ \delta &=& \{\langle a_1 +\!\!+ \cdots +\!\!+\, a_n, [a_1, \ldots, a_n]\rangle \mid a_1, \ldots, a_n \in A\} \\ F &=& \{\langle f, \langle a, [b_1, \ldots, b_n]\rangle\rangle \mid \forall b_i.\exists c.\ c \in a \wedge \langle c, b_i\rangle \in f\} \end{array}$$

Example 4.8. Following Abramsky *et al.*, exponentials in the partial functions GoI model are given by $!p = \{(\langle n, x\rangle, \langle n, y\rangle \mid (x, y) \in p\}$. Thinking of partial functions $D \rightharpoonup D$ as having I/O ports, exponentiation multiplexes $\mathbb{N}$-many ports through a single port by tagging messages with a port number. The K, W, D, δ, F combinators are all defined by manipulating the port numbers to perform the required routing. See Abramsky *et al.* for details [3].

Another source of R-LCAs for the natural number semiring is given by taking a *BCI*-algebra and representing duplicated resources by tupling:

Proposition 4.9. *Every BCI-algebra $(\mathcal{A}, (\cdot), B, C, I)$ can be given the structure of an $(\mathbb{N}, +, \times, 0, 1)$-LCA, where $!_n$ is defined by duplication: $!_n x = [\underbrace{x, \ldots, x}_{n}]$, using tupling (Definition 4.2).*

We also require structure to model the datatype of booleans:

Definition 4.10. An *BCI*-algebra $\mathcal{A}_0 \subseteq \mathcal{A}$ has booleans 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 $E(p, q) \cdot F = q$.

Example 4.11. In the Graph Model of Example 4.3, booleans are represented by $T = \{1\}$ and $F = \{0\}$ and

$$E(\alpha, \beta) = \{\langle 1, x\rangle \mid x \in \alpha\} \cup \{\langle 0, x\rangle \mid x \in \beta\}$$

Example 4.12. Booleans are represented in the partial functions GoI situation algebra (Example 4.4) by $T(x) = \langle 1, x\rangle$, $F(x) = \langle 0, x\rangle$, and:

$$\begin{array}{rcl} E(p, q)\langle 0, x\rangle &=& \langle 1, x\rangle \\ E(p, q)\langle 1, \langle 1, x\rangle\rangle &=& \langle 0, p(x)\rangle \\ E(p, q)\langle 1, \langle 0, x\rangle\rangle &=& \langle 0, q(x)\rangle \end{array}$$

That is, $E(p, q)$ forwards its input to its argument, which tags it with 1 or 0 depending on whether it is true or false. On its return to $E(p, q)$, the appropriate one of p or q is applied before forwarding to the environment. Note that the relations described by $E(p, q)$ are not symmetric, indicating the irreversible nature of following a conditional—there is no general way of working out whether the conditional was true or false given the final output. An interesting avenue for future work is to formulate a version of QTT with a type of reversible booleans, yielding a calculus of dependently typed reversible computation.

4.2 R-QCwFs from R-LCAs

We now fix an R-LCA with booleans and describe how to use it to give a realisability model of QTT that makes use of the resource information from the types. Hoshino [16] has already investigated assemblies over *BCI*-algebras and LCAs in the simply typed setting. Here, we extend this analysis to dependent types.

Interpreting Contexts as Assemblies For the category $\mathcal{L}$ for interpreting contexts and simultaneous substitutions between them, we use the standard notion of the category of *Assemblies* (see, e.g., Longley and Normann [22], §3.3.6). An assembly Γ is a pair[1] 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 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 combinator I to realise identities, and B to realise composition. We use the usual category of sets and functions for our CwF C. The functor $U : \mathcal{L} \to C$ projects out the underlying sets and functions.

Scaling of an assembly $\Gamma = (|\Gamma|, \models_\Gamma)$ yields an assembly with $|\pi\Gamma| = |\Gamma|$, and the realisability relation is defined by: $x \models_{\pi\Gamma} \gamma$ iff $\exists y.x = !_\pi y \wedge y \models_\Gamma \gamma$. 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 $x \models_{\Gamma_1+\Gamma_2} \gamma$ iff there exist y, z such that $x = [y, z]$ and $y \models_{\Gamma_1} \gamma$ and $z \models_{\Gamma_2} \gamma$. Scaling and addition of intensional interpretations of contexts does not affect the underlying meaning but does affect the computational resources available. The empty context $\diamond$ is interpreted using the assembly $\diamond = (\{*\}, \models_\diamond)$, where $x \models_\diamond *$ iff $x = I$. The morphisms $\mathrm{emp}_+ : \diamond \to \diamond + \diamond$ and $\mathrm{emp}_\pi : \diamond \to \pi\diamond$ are the identity as functions, and are realised by elimination of the I combinator by treating it as a zero-element tuple.

Intepreting 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, the collection $\mathrm{Ty}(\Delta)$ of semantic types consists of Δ-indexed collections of assemblies: $\{(|S(\delta)|, \models_{S(\delta)})\}_{\delta \in \Delta}$.

Unresourced Semantic Terms For a set Δ and a family of assemblies $S \in \mathrm{Ty}(\Delta)$, the semantic terms $\mathrm{Tm}(\Delta, S)$ comprise functions of type $\forall \delta \in \Delta.\ |S(\delta)|$, ignoring the realisability information in S. The rest of the CwF structure on C is now standard [15].

Resourced Context Extension Given an assembly $(|\Gamma|, \models_\Gamma)$, a family of assemblies $S \in \mathrm{Ty}(|\Gamma|)$ and a usage $\rho \in R$, the set-theoretic part 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:

$$x \models_{\Gamma.\rho S} (\gamma, s) \quad \text{iff} \quad \exists y, z.\ x = [y, !_\rho z] \wedge y \models_\Gamma \gamma \wedge z \models_{S(\gamma)} s$$

Note the use of $!_\rho$ to ensure that there are enough computational resources provided to satisfy the promise made. The natural transformations $\mathrm{ext}_+ : (\Gamma_1 + \Gamma_2).(\rho_1 + \rho_2)S \to \Gamma_1.\rho_1 S + \Gamma_2.\rho_2 S$ and $\mathrm{ext}_\pi : \pi\Gamma.\pi\rho S \to \pi(\Gamma.\rho S)$ are both the identity as functions, and are realised by rearrangement of tupling.

Resourced Semantic Terms For an assembly Γ and a family of assemblies $S \in \mathrm{Ty}(U\Gamma)$, the resource semantic terms from Γ to S consist of functions that are tracked by some realiser in $\mathcal{A}$:

$$\begin{aligned}\mathrm{RTm}(\Gamma, S) = \{f : \forall \gamma \in |\Gamma|.\ |S(\gamma)| \mid \\ \exists x. \forall \gamma \in |\Gamma|, y \in \mathcal{A}.\ y \models_\Gamma \gamma \Rightarrow x \cdot y \models_{S(\gamma)} f(\gamma)\}\end{aligned}$$

Part (7) of Definition 3.2 is now fulfilled by giving R-LCA "programs" that realise the necessary resource rearrangement for each combinator. For example, variables $\mathrm{v}_{\Gamma.S} \in \mathrm{RTm}(0\Gamma.1S, S\{\mathrm{p}_{U\Gamma.S}\})$ are realised by using K to discard the context, and D to "derelict" away the $!_1$ application.

Dependent Function Types Finally, we must give realisability interpretations for the type formers. For space, we only treat the case of function types. For a set Δ, the underlying sets of the function space family of assemblies consists of the realisable functions:

$$\begin{aligned}|\Pi\pi ST(\delta)| = \{f : \forall s \in |S(\delta)|.\ |T(\delta, s)| \mid \\ \exists x. \forall s, y.\ y \models_{S(\delta)} s \Rightarrow x \cdot !_\pi y \models_{T(\delta,s)} f(s)\}\end{aligned}$$

Note the use of $!_\pi$ to model the fact that the function's realiser is given π-many copies of the input. The bijection Λ is realised using currying and uncurrying in the *BCI*-algebra. See Hoshino [16].

For the remaining type formers: booleans are realised using the structure we stipulated in Definition 4.10. Dependent tensor product types are realised using tuples and $!_\pi$. The universe of small types is interpreted as the (large) set of families of small assemblies.

5 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's [9] Linear Logical Framework, which Vákár [35] provided a categorical semantics for; Krishnaswami *et al.* [18] also investigated a variant of the split context system, also based on Benton's Linear Non-Linear Logic [6].

Another approach to linear dependent type theory has recently been proposed by Luo [23], 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.

QTT is a kind of modal dependent type theory, particularly with the graded exponential modality we derived from our tensor product type. Syntax and semantics for modal dependent type theories have been studied by de Paiva and Ritter [10]. Licata, Shulman and Riley [21] have studied a general form of substructual and modal simple type theories with an eye towards extending them to dependent types. Modal type theories are particularly interesting in the context of Homotopy Type Theory, where they can be used to represent the presence or absence of topological structure on types, as shown by Shulman [32].

McBride's system [25], 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.* [29] and Brunel *et al.* [8] 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.* use a realisability model) to ours. Ghica and Smith [13] particularly emphasises the quantitative nature of the annotations, and presents an application to timing information.

[1]We use capital Greek letters to stand for assemblies as well as the precontexts they interpret.

We have used Abramsky *et al.*'s [3] 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 Example 4.4. This appears to be related to the problem of interpretating the additive connectives of Linear Logic in their axiomatic approach. Nevertheless, the connection with Geometry of Interaction style models is intriguing, and may help develop the theory of QTT into areas such as reversible computation, as described by Abramsky [2], hardware synthesis, as described by Ghica [12], and effectful computation, as described by Hoshino *et al* [17].

6 Conclusions and Future Work

Quantitative Type Theory (QTT) reformulates McBride's combination of linear and dependent types to fix the inadmissibility of substitution. We defined a class of categorical models, *Quantitative Categories with Families*, and given instances that interpret the precise intensional usage information non trivially, using R-Linear Combinatory Algebras. In addition to reversible and effectful variants arising from the GoI models, we plan future on applying QTT to implicit computational complexity in the style of Dal Lago and Hofmann [19, 20] and to low-level imperative computation in the style of Ahmed *et al.* [4].

Acknowledgments

Thanks to Conor McBride, Edwin Brady and James Wood for discussions relating to this paper. This work is dedicated to Orwell the dog. Orwell was a good dog and knew well the difference between zero, one, and many.

References

[1] Samson Abramsky. 1996. Retracing Some Paths in Process Algebra. In *CONCUR '96, Concurrency Theory, 7th International Conference, Pisa, Italy, August 26-29, 1996, Proceedings*. 1–17. https://doi.org/10.1007/3-540-61604-7_44

[2] 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

[3] 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

[4] 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

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

[6] 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

[7] 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

[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*. 351–370. https://doi.org/10.1007/978-3-642-54833-8_19

[9] 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

[10] Valeria de Paiva and Eike Ritter. 2016. Fibrational Modal Type Theory. *Electr. Notes Theor. Comput. Sci.* 323 (2016), 143–161. https://doi.org/10.1016/j.entcs.2016.06.010

[11] 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.

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

[13] 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*. 331–350. https://doi.org/10.1007/978-3-642-54833-8_18

[14] Jean-Yves Girard. 1987. Linear Logic. *Theor. Comp. Sci.* 50 (1987), 1–101.

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

[16] 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

[17] 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

[18] 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

[19] 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

[20] 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

[21] Daniel R. Licata, Michael Shulman, and Mitchell Riley. 2017. A Fibrational Framework for Substructural and Modal Logics. In *2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017, September 3-9, 2017, Oxford, UK*. 25:1–25:22. https://doi.org/10.4230/LIPIcs.FSCD.2017.25

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

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

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

[25] 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

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

[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 (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

[28] 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

[29] 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

[30] 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

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

[32] Michael Shulman. 2017. Brouwer's fixed-point theorem in real-cohesive homotopy type theory. *CoRR* abs:1509.07584v3 (2017). https://arxiv.org/abs/1509.07584v3

[33] 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

[34] The Agda Team. 2018. (2018). http://wiki.portal.chalmers.se/agda.

[35] Matthijs Vákár. 2015. A Categorical Semantics for Linear Logical Frameworks. In *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015 (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