Cost-analysis: How do monads and comonads differ?

Ezgi Çiçek[1], Marco Gaboardi[2], and Deepak Garg[1]

[1]Max Planck Institute for Software Systems, Germany
[2]University at Buffalo, SUNY, USA

1 Introduction

Monads are a de facto standard for the type-based analysis of impure aspects of programs, such as runtime cost [9, 5]. Recently, the logical dual of a monad, the comonad, has also been used for the cost analysis of programs, in conjunction with a linear type system [6, 8]. The logical duality of monads and comonads extends to cost analysis: In monadic type systems, costs are (side) *effects*, whereas in comonadic type systems, costs are *coeffects*. However, it is not clear whether these two methods of cost analysis are related and, if so, how. Are they equally expressive? Are they equally well-suited for cost analysis with all reduction strategies? Are there translations from type systems with effects to type systems with coeffects and viceversa? The goal of this work-in-progress paper is to explore some of these questions in a simple context — the simply typed lambda-calculus (STLC). As we show, even this simple context is already quite interesting technically and it suffices to bring out several key points.

To compare monadic and comonadic type systems for cost analysis of the STLC, we first need to build these type systems. The cost of execution of a program and, hence, its analysis depends on the reduction strategy. Accordingly, we consider call-by-value (CBV) and call-by-name (CBN) reduction semantics separately. For each of these two reduction strategies, we first build a monadic cost type system and a comonadic linear cost type system, for a total of four type systems. Our development of these four type systems is based on known, standard translations of the CBN and CBV STLC into the computational lambda calculus and intuitionistic linear logic due to Moggi and Girard, respectively. See [1] for a standard presentation of the four translations. The standard computational lambda calculus and the standard linear logic do not take explicit reduction costs into account, but they can be extended in a natural way to do so. We can grade the monad of the computational lambda calculus with cost as an effect as in [13]. In a dual way, we can grade the comonad of linear logic (the operator "bang" or !) with the cost as a coeffect, as in [14, 11, 4], and reminiscent of Bounded Linear Logic [12].

Next, we pre-image the type systems from the targets of the translation (i.e., the cost-graded type systems for the computational lambda calculus and intuitionistic linear logic) back to the STLC. This yields the four cost type systems for STLC, two each — one monadic and one comonadic — for each of CBV and CBN. The monadic and the comonadic type systems we obtain for CBV are known from prior work [9, 8] as is the comonadic type system for CBN [6]. We have not seen the CBN monadic type system before, but it is completely natural in hindsight. For brevity, in this paper we directly present these four type systems, skipping how we obtain them through the translations, since that background is inessential for our remaining development.

Having built the four type systems, we start examining them closely. First, we show that all four type systems are sound: Any cost provable in any of the type systems is an upper bound on the actual cost of reduction, counting a unit cost for each step of β-reduction (in whatever strategy, CBV or CBN, the type system was designed for). To do this, we use cost-annotated unary logical relations. Logical relations have appeared in the literature on cost analysis in various forms [7, 5, 3, 10]. For instance, in the context of cost analysis for linear logic they are often used in the form of Krivine's realizability [2]. Second, we make an interesting observation: For CBV, the monadic type system is more amenable to an implementation than the comonadic one, while for CBN, the picture reverses. This observation is instructive for researchers and practitioners looking to build type

systems for cost analysis.

Third, we present translations of the monadic type systems into the comonadic ones by syntactic transformation of typing derivations. This suggests that the comonadic style is at least as expressive as the monadic style, mathematically speaking. The existence of such a translation may sound contradictory to our previous point that, in the case of CBN, the comonadic system is more amenable to an implementation. However, there is no inconsistency here, because the translations are non-deterministic and do not lend themselves to a straightforward implementation.

Ongoing work In ongoing work, we are looking at (a) soundness proofs for our translations and discovering translations in the other direction — from the comonadic type systems to the monadic type systems and (b) relative completeness of the type systems. Both questions are difficult. Finding translations for comonadic type systems in monadic ones is difficult because the former track a single cost for the entire program, whereas the latter track costs of subprograms locally (via the monad). Hence, some amount of guessing or splitting of costs is needed for the translation. Relative completeness is known to be a hard question because the annotations in which costs are expressed symbolically in the type system must be sufficiently expressive. Prior work has shown how to describe cost annotations completely for the comonadic type systems for PCF [6, 8], but to the best of our knowledge, no results are known for the STLC, in either the monadic or the comonadic style.

Organization of the paper In Section 2, we present our development for CBV semantics. This includes the monadic and comonadic cost type systems, their soundness results and the translation from the monadic to the comonadic type system. In Section 3, we present the parallel (but technically very different) development for CBN. Section 4 presents some additional discussion and concludes the paper.

2 Call-by-value

We present the monadic and comonadic cost type systems for CBV in Figure 1 and Figure 2, respectively.

In the monadic type system, cost annotations ϵ appear in two places: on the function type $A \xrightarrow{\epsilon} B$ and on the right side of a typing judgment as in $:_\epsilon$. The first of these annotations describes the cost of the evaluation of the body of the function. In monadic notation with a graded cost monad T, the type would be written $A \to T_\epsilon(B)$ (given a value of type A, we get a computation of type B and runtime cost ϵ). The second kind of annotation describes the cost of the evaluation of the term that is the subject of the judgment. These two annotations interact in the rule (lam) and in the rule (app).

$$\begin{array}{lllll} \text{Types} & A, B & ::= & \mathsf{Nat} \mid A \xrightarrow{\epsilon} B \\ \text{Context} & \Gamma & ::= & x_1 : A_1, \ldots, x_n : A_n \\ \text{Judgment} & & & \Gamma \vdash t :_\epsilon A \end{array}$$

$$\frac{}{\Gamma, x : A \vdash x :_0 A}\ \mathsf{id} \qquad \frac{\Gamma, x : A \vdash t :_\epsilon B}{\Gamma \vdash \lambda x.t :_0 A \xrightarrow{\epsilon} B}\ \mathsf{lam}$$

$$\frac{\Gamma \vdash t :_{\epsilon_1} A \xrightarrow{\epsilon} B \qquad \Gamma \vdash u :_{\epsilon_2} A}{\Gamma \vdash t\, u :_{\epsilon+\epsilon_1+\epsilon_2+1} B}\ \mathsf{app}$$

$$\frac{}{\Gamma \vdash \underline{n} :_0 \mathsf{Nat}}\ \mathsf{nat}$$

Figure 1: CBV monadic system

The comonadic type system is quite different. Annotations appear in two places: in the exponential modality $!_r$, and in the conclusion of the typing judgment $\mid \epsilon$. The first of these describes the number of times a term can be used, while the second describes the cost of evaluating the term.

There are two salient points of difference between the two type systems. First, unlike the monadic system's function types, the comonadic system's function types carry no cost (ϵ) annotations. Instead, in the comonadic system, there is only one global cost on the typing judgment. This also means that the two systems count costs for functions differently. Second, in the comonadic type system the rule (lam) anticipates the number of times r the constructed function will be used by its context, by adding the annotation $!_r$. In an implementation (type checking or type inference), this number must be guessed or approximated by an analysis of the surrounding context. Similar guessing is not required in the monadic type system, which is, therefore, likely more amenable to an implementation. However, both type systems are sound.

Theorem 1 (Soundness, CBV)**.** If either $\vdash t :_\epsilon A$ or $\vdash t : A \mid \epsilon$, then there is a $k \in \mathbb{N}$ such that $t \to^* v$ in CBV with k applications of rule (app) and $k \leq \epsilon$.

$$\begin{array}{llll} \text{Types} & A, B & ::= & \mathsf{Nat} \mid !_r(A \to B) \\ \text{Context} & \Gamma & ::= & x_1 : A_1, \ldots, x_n : A_n \\ \text{Judgment} & & & \Gamma \vdash t : A \mid \epsilon \end{array}$$

$$\begin{aligned} \Gamma_1 \boxplus \Gamma_2 = {} & \{x : \mathsf{Nat} \mid x : \mathsf{Nat} \in \Gamma_1 \cup \Gamma_2\} \cup \\ & \{x :!_{n+m}(A \to B) \mid x :!_n(A \to B) \in \Gamma_1 \\ & \qquad \text{and } x :!_m(A \to B) \in \Gamma_2\} \end{aligned}$$

$$\begin{aligned} r \star \Gamma \quad = {} & \{x : \mathsf{Nat} \mid x : \mathsf{Nat} \in \Gamma\} \cup \\ & \{x :!_{r \cdot n}(A \to B) \mid x :!_n(A \to B) \in \Gamma\} \end{aligned}$$

$$\frac{}{x : A \vdash x : A \mid 0}\ \mathsf{id}$$

$$\frac{\Gamma, x : A \vdash t : B \mid \epsilon}{r \star \Gamma \vdash \lambda x.t :!_r(A \to B) \mid r \cdot \epsilon}\ \mathsf{lam}$$

$$\frac{\Gamma_1 \vdash t :!_1(A \to B) \mid \epsilon_1 \qquad \Gamma_2 \vdash u : A \mid \epsilon_2}{\Gamma_1 \boxplus \Gamma_2 \vdash t\, u : B \mid \epsilon_1 + \epsilon_2 + 1}\ \mathsf{app}$$

$$\frac{}{\vdash \underline{n} : \mathsf{Nat} \mid 0}\ \mathsf{nat}$$

Figure 2: CBV comonadic system

Proof. By separate logical relations for the monadic and comonadic type systems. □

To better understand the differences in how the two type systems count costs, we develop a translation from the monadic calculus to the comonadic one.[1] This translation is shown in Figure 3. The most instructive clause of the translation is that for typing judgments, which explains the relation between the ways in which the two type systems count costs. The translation is *non-deterministic*: The number r in the translation of function types can be arbitrary, which reflects our earlier point about the need to guess the number of uses of a function in the comonadic type system, but not the monadic one. We conjecture that the translation maps a valid type derivation to a valid type derivation for at least one resolution of this nondeterminism and we are currently working on a proof of this conjecture.

$$\begin{array}{llll} \text{Types} & A, B & ::= & \mathsf{Nat} \mid A \xrightarrow{\epsilon_1, \epsilon_2} B \\ \text{Context} & \Gamma & ::= & x_1 :_{\epsilon_1} A_1, \ldots, x_n :_{\epsilon_n} A_n \\ \text{Judgment} & & & \Gamma \vdash t :_\epsilon A \end{array}$$

$$\frac{}{\Gamma, x :_\epsilon A \vdash x :_\epsilon A}\ \mathsf{id} \qquad \frac{\Gamma, x :_{\epsilon_1} A \vdash t :_{\epsilon_2} B}{\Gamma \vdash \lambda x.t :_0 A \xrightarrow{\epsilon_1, \epsilon_2} B}\ \mathsf{lam}$$

$$\frac{\Gamma \vdash t :_\epsilon A \xrightarrow{\epsilon_1, \epsilon_2} B \qquad \Gamma \vdash u :_{\epsilon_1} A}{\Gamma \vdash t\, u :_{\epsilon + \epsilon_2 + 1} B}\ \mathsf{app}$$

$$\frac{}{\Gamma \vdash \underline{n} :_0 \mathsf{Nat}}\ \mathsf{nat}$$

Figure 4: CBN monadic system

3 Call-by-name

We carry out a similar development for CBN. The monadic and comonadic cost type systems for CBN are shown in Figure 4 and Figure 5, respectively.

In the monadic type system, cost annotations ϵ appear in two places: On the function type $A \xrightarrow{\epsilon_1, \epsilon_2} B$ and on all typing judgments as in $:_\epsilon$. $A \xrightarrow{\epsilon_1, \epsilon_2} B$ ascribes functions whose body has cost ϵ_2, given an argument of cost ϵ_1. In CBN, the argument to a function is a term, not a value, so the cost of evaluating the argument must be assumed in order to compute the cost of the function's body. Similarly, the costs of all variables in the context must be assumed to compute the cost of a term. This explains the annotation ϵ_1 on $A \xrightarrow{\epsilon_1, \epsilon_2} B$, and the annotation $:_\epsilon$ on all variables in the context, neither of which was required in the CBV monadic type system.

The comonadic type system also has two cost annotations: $!_r$ on function types $(!_r A) \to B$ and $\mid \epsilon$ in the conclusion of the typing judgment. The first describes the number of times a function uses its argument, while the second describes the cost of evaluating the typed term. The comonadic type system does not need to assume the costs of arguments and context variables because it counts the cost of a term *without* the cost of its leaves (variables). This is sound because the type system is linear.

Dual to CBV, the *monadic* CBN type system requires guessing the cost ϵ_1 in the rule (lam) whereas the comonadic CBN type system requires no similar guessing. Consequently, for CBN, the comonadic

[1] We are currently attempting to build a translation in the other direction as well.

$\ell(\mathsf{Nat})_{\mathsf{v}} \triangleq \mathsf{Nat}$ $\ell(A \xrightarrow{\epsilon} B)_{\mathsf{v}} \triangleq !_r(\ell(A)_{\mathsf{v}} \multimap \ell(B)_{\mathsf{v}})$	Type translation
$\ell(\Gamma, x : A)_{\mathsf{v}} \triangleq \ell(\Gamma)_{\mathsf{v}}, x : \ell(A)_{\mathsf{v}}$	Context translation
$\ell(\Gamma \vdash t :_\epsilon A)_{\mathsf{v}} \triangleq \Delta \vdash t : A' \mid \epsilon + (A)_{A'} - (\Gamma)_\Delta$ where $\Delta = \ell(\Gamma)_{\mathsf{v}}$, $A' = \ell(A)_{\mathsf{v}}$ and: $(\mathsf{Nat})_{\mathsf{Nat}} \triangleq 0 \quad (A \xrightarrow{\epsilon} B)_{!_r(A' \to B')} \triangleq r \cdot (\epsilon + (B)_{B'} - (A)_{A'})$ $(\bullet)_\bullet \triangleq 0 \quad (\Gamma, x : A)_{\Delta, x:A'} \triangleq (\Gamma)_\Delta + (A)_{A'}$	Typing translation

Figure 3: Translation of the CBV monadic system to the CBV comonadic system.

Types	A, B	$::=$	$\mathsf{Nat} \mid (!_r A) \to B$
Context	Γ	$::=$	$x_1 :!_{r_1} A_1, \ldots, x_n :!_{r_n} A_n$
Judgment			$\Gamma \vdash t : A \mid \epsilon$

$$\Gamma_1 + \Gamma_2 = \{x :!_{n+m} A \mid x :!_n A \in \Gamma_1 \text{ and } x :!_m A \in \Gamma_2\}$$

$$r \star \Gamma = \{x : \mathsf{Nat} \mid x : \mathsf{Nat} \in \Gamma\} \cup \{x :!_{r \cdot n}(A \to B) \mid x :!_n (A \to B) \in \Gamma\}$$

$$\frac{}{x :!_1 A \vdash x : A \mid 0}\ \mathsf{id} \qquad \frac{\Gamma, x :!_r A \vdash t : B \mid \epsilon}{\Gamma \vdash \lambda x.t : (!_r A) \to B \mid \epsilon}\ \mathsf{lam}$$

$$\frac{\Gamma_1 \vdash t : (!_r A) \to B \mid \epsilon_1 \qquad \Gamma_2 \vdash u : A \mid \epsilon_2}{\Gamma_1 + r \star \Gamma_2 \vdash t\, u : B \mid \epsilon_1 + r \cdot \epsilon_2 + 1}\ \mathsf{app}$$

$$\frac{}{\vdash \underline{n} : \mathsf{Nat} \mid 0}\ \mathsf{nat}$$

Figure 5: CBN comonadic system

type system might be easier to implement. Both type systems are sound.

Theorem 2 (Soundness, CBN)**.** If either $\vdash t :_\epsilon A$ or $\vdash t : A \mid \epsilon$, then there is a $k \in \mathbb{N}$ such that $t \to^* v$ in CBN with k applications of rule (app) and $k \leq \epsilon$.

Proof. By separate logical relations for the monadic and comonadic type systems. □

Figure 6 shows the translation from the monadic type system to the comonadic one. The interesting clause is the one for typing judgments, which explains how cost accounting in the two type systems relates. This translation is non-deterministic because the translation of function types and of contexts has a free r on the right side. Again, we conjecture that the translation maps a valid monadic type derivation to a valid comonadic type derivation for at least one resolution of the non-determinism.

4 Conclusion and Future Work

Based on our type systems, we note that the primary difference between monadic and comonadic cost analysis is that in the former the costs of arguments must be assumed locally in analyzing a function, while in the latter that cost is added at the point of function application. Naturally, the monadic analysis fits only CBV well, where arguments are values with fixed cost 0. The comonadic analysis fits CBN better because of the way it counts costs for functions.

A significant open question at this point is that of relative completeness: Can every STLC expression be typed in either the monadic or the comonadic style with a precise cost? Prior work answers this question

$\ell(\mathsf{Nat})_{\mathsf{n}} \triangleq \mathsf{Nat}$ $\ell(A \xrightarrow{\epsilon_1,\epsilon_2} B)_{\mathsf{n}} \triangleq !_r\ell(A)_{\mathsf{n}} \multimap \ell(B)_{\mathsf{n}}$	Type translation
$\ell(\Gamma, x :_\epsilon A)_{\mathsf{n}} \triangleq \ell(\Gamma)_{\mathsf{n}}, x : !_r\ell(A)_{\mathsf{n}}$	Context translation
$\ell(\Gamma \vdash t :_\epsilon A)_{\mathsf{n}} \triangleq \Delta \vdash t : A' \mid \epsilon + (A)_{A'} - (\Gamma)_\Delta$ where $\Delta = \ell(\Gamma)_{\mathsf{n}}$, $A' = \ell(A)_{\mathsf{n}}$ and: $(\mathsf{Nat})_{\mathsf{Nat}} \triangleq 0 \quad (A \xrightarrow{\epsilon_1,\epsilon_2} B)_{!_rA'\to B'} \triangleq \epsilon_2 + (B)_{B'} - r \cdot (\epsilon_1 + (A)_{A'})$ $(\bullet)_\bullet \triangleq 0 \quad (\Gamma, x :_\epsilon A)_{\Delta, x:!_rA'} \triangleq (\Gamma)_\Delta + r \cdot (\epsilon + (A)_{A'})$	Typing translation

Figure 6: Translation of the CBN monadic system to the CBN comonadic system.

affirmatively for PCF in the comonadic style for both CBN and CBV [6, 8]. The main difficulty is in determining a sufficiently expressive language for costs. While it is trivial to see that the language used for PCF can also be used to obtain relative completeness for the STLC (in the comonadic style), the more interesting question is whether that language can be simplified for the STLC. Also, the question of relative completeness in the monadic style remains open.

References

[1] P. N. Benton and Philip Wadler. Linear logic, monads and the lambda calculus. In *Proc. LICS*, pages 420–431, 1996.

[2] Aloïs Brunel. Quantitative classical realizability. *Inf. Comput.*, 241:62–95, 2015.

[3] Aloïs Brunel and Marco Gaboardi. Realizability models for a linear dependent PCF. *Theor. Comput. Sci.*, 585:55–70, 2015.

[4] Aloïs Brunel, Marco Gaboardi, Damiano Mazza, and Steve Zdancewic. A core quantitative coeffect calculus. In *Proc. ESOP*, pages 351–370, 2014.

[5] Ezgi Çiçek, Deepak Garg, and Umut A. Acar. Refinement types for incremental computational complexity. In *Proc. ESOP*, pages 406–431, 2015.

[6] Ugo Dal Lago and Marco Gaboardi. Linear dependent types and relative completeness. In *Proc. LICS*, pages 133–142, 2011.

[7] Ugo Dal Lago and Martin Hofmann. Realizability models and implicit complexity. *Theor. Comput. Sci.*, 412(20):2029–2047, 2011.

[8] Ugo Dal Lago and Barbara Petit. Linear dependent types in a call-by-value scenario. In *Proc. PPDP*, pages 115–126, 2012.

[9] Nils Anders Danielsson. Lightweight semiformal time complexity analysis for purely functional data structures. In *Proc. POPL*, pages 133–144, 2008.

[10] Norman Danner, Daniel R. Licata, and Ramyaa Ramyaa. Denotational cost semantics for functional languages with inductive types. *SIGPLAN Not.*, 50(9):140–151, August 2015.

[11] Dan R. Ghica and Alex I. Smith. Bounded linear types in a resource semiring. In *Proc. ESOP*, pages 331–350, 2014.

[12] Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. Bounded linear logic: A modular approach to polynomial-time computability. *Theor. Comput. Sci.*, 97(1):1–66, 1992.

[13] Shin-ya Katsumata. Parametric effect monads and semantics of effect systems. In *Proc. POPL*, pages 633–646, 2014.

[14] Tomas Petricek, Dominic A. Orchard, and Alan Mycroft. Coeffects: a calculus of context-dependent computation. In *Proc. ICFP*, pages 123–135, 2014.