Completeness of algebraic CPS simulations

Ali Assaf
LIG, Université Joseh Fourier
Grenoble, France
École Polytechnique
Palaiseau, France
Ali.Assaf@imag.fr

Simon Perdrix
CNRS, LIG, Université de Grenoble
Grenoble, France
Simon.Perdrix@imag.fr

The *algebraic lambda calculus* (λ_{alg}) and the *linear algebraic lambda calculus* (λ_{lin}) are two extensions of the classical lambda calculus with linear combinations of terms. They arise independently in distinct contexts: the former is a fragment of the differential lambda calculus, the latter is a candidate lambda calculus for quantum computation. They differ in the handling of application arguments and algebraic rules. The two languages can simulate each other using an algebraic extension of the well-known call-by-value and call-by-name CPS translations. These simulations are sound, in that they preserve reductions. In this paper, we prove that the simulations are actually complete, strengthening the connection between the two languages.

1 Introduction

Algebraic lambda calculi The *algebraic lambda calculus* (λ_{alg}) [18] and the *linear algebraic lambda calculus* (λ_{lin}) [4] are two languages that extend the classical lambda calculus with linear combinations of terms such as $\alpha.M+\beta.N$. They have been introduced independently in two different contexts. The former is a fragment of the differential lambda calculus, and has been introduced in the context of linear logic with the purpose of quantifying non-determinism: each term of a linear combination represents a possible evolution in a non deterministic setting. The latter has been introduced as a candidate for a language of quantum computation, where a linear combination of terms corresponds to a superposition of states such as $\frac{1}{\sqrt{2}}.|0\rangle+\frac{1}{\sqrt{2}}.|1\rangle$. The strength of λ_{lin} is to allow superpositions of any terms without distinguishing programs and data, whereas most of the candidate languages for quantum computation are based on the slogan *quantum data, classical control* [14, 15, 11].

The two languages, λ_{alg} and λ_{lin}, differ in their operational semantics. It turns out that the first follows a *call-by-name* strategy while the second follows the equivalent of a *call-by-value* strategy. For example, in λ_{alg} the term $(\lambda x.\, fxx)(\alpha.y+\beta.z)$ reduces as follows:

$$(\lambda x.\, fxx)(\alpha.y+\beta.z) \quad\to\quad f(\alpha.y+\beta.z)(\alpha.y+\beta.z)$$

However, this does not agree with the nature of quantum computing. It leads to the cloning of the state $\alpha.y+\beta.z$, which contradicts the *no-cloning* theorem [19]. Only copying of base terms such as y is allowed. Therefore, λ_{lin} reduces the term as follows.

$$\begin{aligned}(\lambda x.\, fxx)(\alpha.y+\beta.z) &\to (\lambda x.\, fxx)(\alpha.y)+(\lambda x.\, fxx)(\beta.z)\\ &\to \alpha.(\lambda x.\, fxx)y+\beta.(\lambda x.\, fxx)z\\ &\to \alpha.fyy+\beta.fzz\end{aligned}$$

E. Kashefi, J. Krivine, F. van Raamsdonk (Eds.)
DCM 2011
EPTCS 88, 2012, pp. 16–27, doi:10.4204/EPTCS.88.2

Despite these differences, the work in [5] showed that the two languages can simulate each other. This was accomplished by defining a translation from one language to the other. Given a term M of λ_{lin}, we can encode it into a term N of λ_{alg} such that reductions of M in λ_{lin} correspond to reductions of N in λ_{alg}. The translation is an algebraic extension of the classical *continuation-passing style* (CPS) encoding used for simulating call-by-name and call-by-value [9, 12, 13].

Contribution The CPS transformations introduced in [5] have been proven to be sound, *i.e.* if a term M reduces to a value V in the source language, then the translation of M reduces to the translation of V in the target language. In this paper we prove that they are actually complete, *i.e.* that the converse is also true: if the translation of M reduces to the translation of V in the target language, then M reduces to V in the source language. We do so by modifying techniques used by Sabry and Wadler in [13] to define an inverse translation and showing that it also preserves reductions. The completeness of these CPS transformations strengthens the connection between works done in linear logic [6, 7, 8, 17] and works on quantum computation [1, 2, 3, 16].

Plan of the paper The rest of the paper is structured as follows. In section 2, the syntax and the reduction rules of both algebraic languages are presented. Section 3 is dedicated to the simulation of λ_{lin} by λ_{alg}, and section 4 to the opposite simulation. In each of the two cases, the translation introduced in [5] is presented, the grammar of the encoded terms in the target language is given, the inverse translation is defined, and finally the completeness of the CPS translation is proven.

2 The algebraic lambda calculi

The languages λ_{lin} and λ_{alg} share the same syntax, defined by the following grammar, where α ranges over a defined ring, the *ring of scalars*.

$$\begin{array}{rcll} M,N,L & ::= & V \mid MN \mid \alpha.M \mid M+N & \text{(terms)} \\ U,V,W & ::= & B \mid 0 \mid \alpha.V \mid V+W & \text{(values)} \\ B & ::= & x \mid \lambda x.M & \text{(base values)} \end{array}$$

We can form sums of terms and multiplication by scalars, and there is a neutral element 0. The values we consider are formed by taking linear combinations of base values, *i.e.* variables and abstractions. This gives the languages the structure of a vector space (a *module* to be precise).

We describe the operational semantics of the two languages using small-step rewrite rules. The rules are presented in Figure 1. As mentioned, λ_{alg} substitutes the argument directly in the body of a function, while λ_{lin} delays the substitution until the argument is a base value. We use the same notation as in [5] to define the following rewrite systems obtained by combining the rules described in Figure 1 .

$$\begin{array}{rclcrcl} \to_{\beta_n} & ::= & \beta_n \cup \xi & \quad & \to_{\beta_v} & ::= & \beta_v \cup \xi \cup \xi_{\lambda_{lin}} \\ \to_a & ::= & A \cup L \cup \xi & & \to_l & ::= & A_l \cup A_r \cup L \cup \xi \cup \xi_{\lambda_{lin}} \end{array}$$

The rewrite systems for the two languages are then defined as follows.

Language	Rewrite system
λ_{lin}	$\to_{l\cup\beta} ::= (\to_l) \cup (\to_{\beta_v})$
λ_{alg}	$\to_{a\cup\beta} ::= (\to_a) \cup (\to_{\beta_n})$

Rules specific to λ_{alg}

Call-by-name (β_n)

$$(\lambda x.M)N \to M[x := N]$$

Linearity of application (A)

$$\begin{array}{rcl}(M+N)L & \to & ML+NL \\ (\alpha.M)N & \to & \alpha.(MN) \\ (0)M & \to & 0\end{array}$$

Rules specific to λ_{lin}

Call-by-value (β_v)

$$(\lambda x.M)B \to M[x := B]$$

Right context rule ($\xi_{\lambda_{lin}}$)

$$\frac{M \to M'}{VM \to VM'}$$

Left linearity of application (A_l)

$$\begin{array}{rcl}(M+N)V & \to & MV+NV \\ (\alpha.M)V & \to & \alpha.(MV) \\ (0)V & \to & 0\end{array}$$

Right linearity of application (A_r)

$$\begin{array}{rcl}B(M+N) & \to & BM+BN \\ B(\alpha.M) & \to & \alpha.(BM) \\ B(0) & \to & 0\end{array}$$

Common rules

Vector space rules ($L = Asso \cup Com \cup F \cup S$)

Associativity ($Asso$)

$$\begin{array}{rcl}M+(N+L) & \to & (M+N)+L \\ (M+N)+L & \to & M+(N+L)\end{array}$$

Commutativity (Com)

$$M+N \to N+M$$

Factorization (F)

$$\begin{array}{rcl}\alpha.M+\beta.M & \to & (\alpha+\beta).M \\ \alpha.M+M & \to & (\alpha+1).M \\ M+M & \to & (1+1).M \\ \alpha.(\beta.M) & \to & (\alpha\beta).M\end{array}$$

Simplification (S)

$$\begin{array}{rcl}\alpha.(M+N) & \to & \alpha.M+\alpha.N \\ 1.M & \to & M \\ 0.M & \to & 0 \\ \alpha.0 & \to & 0 \\ 0+M & \to & M\end{array}$$

Context rules (ξ)

$$\frac{M \to M'}{(M)\,N \to (M')\,N} \qquad \frac{M \to M'}{M+N \to M'+N}$$

$$\frac{M \to M'}{\alpha.M \to \alpha.M'} \qquad \frac{N \to N'}{M+N \to M+N'}$$

Figure 1: Rewrite rules for λ_{lin} and λ_{alg}

Example 1. Let $\langle M,N\rangle := \lambda f.\, fMN$ be the Church encoding of pairs, let $\mathsf{copy} = \lambda x.\, \langle x,x\rangle$, and consider the term $\mathsf{copy}(y+z)$. The term reduces in λ_{alg}:

$$\begin{array}{rcl} \mathsf{copy}(y+z) & = & (\lambda x.\, \langle x,x\rangle)(y+z) \\ & \to_{\beta_n} & \langle y+z, y+z\rangle \end{array}$$

As mentioned above, the term $y+z$ is cloned, and if it represented quantum superposition this would violate the no-cloning theorem. In λ_{lin}, the term reduces instead as:

$$\begin{array}{rcl} \mathsf{copy}(y+z) & = & (\lambda x.\, \langle x,x\rangle)(y+z) \\ & \to_{l} & (\lambda x.\, \langle x,x\rangle)y + (\lambda x.\, \langle x,x\rangle)z \\ & \to_{\beta_v} & \langle y,y\rangle + (\lambda x.\, \langle x,x\rangle)z \\ & \to_{\beta_v} & \langle y,y\rangle + \langle z,z\rangle \end{array}$$

3 Completeness of the call-by-value to call-by-name simulation

The translation in [5] is a direct extension of the classical CPS encoding used by Plotkin [12] to show that the call-by-name lambda calculus simulates call-by-value. The definition is the following.

$$\begin{array}{rcl} [\![x]\!] & = & \lambda k.kx \\ [\![\lambda x.M]\!] & = & \lambda k.k(\lambda x.[\![M]\!]) \\ [\![MN]\!] & = & \lambda k.[\![M]\!](\lambda b_1.[\![N]\!](\lambda b_2.b_1 b_2 k)) \\ [\![0]\!] & = & 0 \\ [\![\alpha.M]\!] & = & \lambda k.(\alpha.[\![M]\!])k \\ [\![M+N]\!] & = & \lambda k.([\![M]\!]+[\![N]\!])k \end{array}$$

This translation simulates the reductions of a term M by the reductions of the term $[\![M]\!]k$, where k is free. It works the same way as the classical CPS simulation: instead of returning the result of a computation, all terms receive an additional argument k called the *continuation*, which describes the rest of the computation. This technique makes evaluation order, intermediate values, and function returns explicit, which allows us to encode the proper evaluation strategy.

The translation preserves the set of free variables. New variables names like k, b, b_1 or b_2 are chosen to be fresh so as to not collide with free variables in the term. We reserve and always use the name k to abstract over continuations, and the names b, b_1, and b_2 for intermediate values. It is a fact that these variables never clash with each other.

Example 2. The reductions of the term $\mathsf{copy}(y+z)$ in λ_{lin} are simulated in λ_{alg} by the following reductions:

$$
\begin{array}{rcl}
[\![\mathsf{copy}(y+z)]\!]k & = & (\lambda k.\,[\![\mathsf{copy}]\!]\,(\lambda b_1.\,[\![y+z]\!]\,(\lambda b_2.\,b_1 b_2 k)))\,k \\
& \to_{\beta_n} & [\![\mathsf{copy}]\!]\,(\lambda b_1.\,[\![y+z]\!]\,(\lambda b_2.\,b_1 b_2 k)) \\
& \to_{\beta_n} & (\lambda b_1.\,[\![y+z]\!]\,(\lambda b_2.\,b_1 b_2 k))\,(\lambda x.\,[\![\langle x,x\rangle]\!]) \\
& \to_{\beta_n} & [\![y+z]\!]\,(\lambda b_2.\,(\lambda x.\,[\![\langle x,x\rangle]\!])\,b_2 k) \\
& \to_{\beta_n} & ([\![y]\!]+[\![z]\!])\,(\lambda b_2.\,(\lambda x.\,[\![\langle x,x\rangle]\!])\,b_2 k) \\
& \to_{a} & [\![y]\!]\,(\lambda b_2.\,(\lambda x.\,[\![\langle x,x\rangle]\!])\,b_2 k)+[\![z]\!]\,(\lambda b_2.\,(\lambda x.\,[\![\langle x,x\rangle]\!])\,b_2 k) \\
& \to^*_{a\cup\beta} & (\lambda b_2.\,(\lambda x.\,[\![\langle x,x\rangle]\!])\,b_2 k)\,y+(\lambda b_2.\,(\lambda x.\,[\![\langle x,x\rangle]\!])\,b_2 k)\,z \\
& \to^*_{a\cup\beta} & (\lambda x.\,[\![\langle x,x\rangle]\!])\,yk+(\lambda x.\,[\![\langle x,x\rangle]\!])\,zk \\
& \to^*_{a\cup\beta} & [\![\langle y,y\rangle]\!]k+[\![\langle z,z\rangle]\!]k
\end{array}
$$

We see that the result is the one that corresponds to call-by-value. As expected, there was no cloning.

Notice in the example above that there are many more steps in the simulation than in the original reduction sequence in Example 1. A lot of the steps replace the continuation variables and intermediate variables introduced by the translation. In a sense, all these intermediary terms represent the "same" term in the source language, and we call these intermediary steps *administrative reductions*.

To deal with this, we use an intermediate translation denoted by $M : K$. This *colon* translation was originally used by Plotkin [12] to describe intermediate reductions of translated terms, where initial *administrative redexes* had been eliminated.

$$
\begin{array}{rcl}
\Psi(x) & = & x \\
\Psi(\lambda x.M) & = & \lambda x.\,[\![M]\!] \\
B:K & = & K\Psi(B) \\
0:K & = & 0 \\
\alpha.M:K & = & \alpha.(M:K) \\
M+N:K & = & M:K+N:K
\end{array}
\qquad
\begin{array}{rcl}
BN:K & = & N:\lambda b.\Psi(B)bK \\
(MN)L:K & = & MN:\lambda b_1.\,[\![L]\!](\lambda b_2.\,b_1 b_2 K) \\
(0)N:K & = & 0:K \\
(\alpha.M)N:K & = & \alpha.(MN):K \\
(M+N)L:K & = & ML+NL:K
\end{array}
$$

This CPS translation was proved to be sound by showing that it preserves reductions: for any term M, if M reduces to M', then $M:K$ reduces to $M':K$ for all K. Combined with the fact that $[\![M]\!]k$ reduces initially to $M:k$, this gave the soundness of the simulation.

Proposition 3 (Soundness [5])**.** *For any term M, if $M \to^*_{l\cup\beta} V$ then $[\![M]\!]k \to^*_{a\cup\beta} V:k$.*

The goal of this paper is to show that the converse is also true:

Theorem 4 (Completeness)**.** *If $[\![M]\!]k \to^*_{a\cup\beta} V:k$ then $M \to^*_{l\cup\beta} V$.*

To prove it, we define an inverse translation and show that it preserves reductions. First, we need to characterize the structure of the encoded terms. We define a subset of λ_{alg} which contains the image of the translation and is closed by $\to_{a\cup\beta}$ reductions with the following grammar:

$$
\begin{array}{rcll}
C & ::= & KB \mid B_1B_2K \mid TK & \text{(base computations)} \\
D & ::= & C \mid 0 \mid \alpha.D \mid D_1 + D_2 & \text{(computation combinations)} \\
\\
S & ::= & \lambda k.C & \text{(base suspensions)} \\
T & ::= & S \mid 0 \mid \alpha.T \mid T_1 + T_2 & \text{(suspension combinations)} \\
\\
K & ::= & k \mid \lambda b.BbK \mid \lambda b_1.T(\lambda b_2.b_1b_2K) & \text{(continuations)} \\
\\
B & ::= & x \mid \lambda x.S & \text{(CPS-values)}
\end{array}
$$

There are four main categories of terms: *computations*, *suspensions*, *continuations*, and *CPS-values*. We distinguish base computations C from linear combinations of computations D, as well as base suspensions S from linear combinations of suspensions T. The translation $[\![M]\!]$ gives a term of the class T, while $[\![M]\!]k$ and $M : K$ are of class D. One can easily check that each of the classes D, T, K and B is closed by $\to_{a\cup\beta}$ reductions.

There are some restrictions on the names of the variables in this grammar. The variable name k that appears in the class K must be the same as the one used in suspensions of the form $\lambda k.C$. It cannot appear as a variable name in any other term. This is to agree with the requirement of freshness that we mentioned above. The same applies for the variables b, b_1 and b_2: they cannot appear (free) in any sub-term. In particular, these restrictions ensure that the grammar for each category is unambiguous. The three kinds of variables (x, k and b) play different roles, which is why we distinguish them using different names.

Computations are the terms that simulate the steps of the reductions, hence the name. They are the only terms that contain applications, so they are the only terms that can β-reduce. In fact, notice that the arguments in applications are always base values. This shows a simple alternative proof for the *indifference* property [5] of the CPS translation, namely that the reductions of a translated term are exactly the same in λ_{lin} and λ_{alg}.

Proposition 5 (Indifference [5])**.** *For any computations D and D', $D \to_{a\cup\beta} D'$ if and only if $D \to_{l\cup\beta} D'$. In particular, if $M \to^*_{l\cup\beta} V$ then $[\![M]\!]k \to^*_{l\cup\beta} V : k$.*

We define the inverse translation using the following four functions, corresponding to each of the four main categories in the grammar.

$$
\begin{array}{rcl}
\overline{KB} & = & \underline{K}[\psi(B)] \\
\overline{B_1B_2K} & = & \underline{K}[\psi(B_1)\psi(B_2)] \\
\overline{TK} & = & \underline{K}[\sigma(T)] \\
\overline{0} & = & 0 \\
\overline{\alpha.D} & = & \alpha.\overline{D} \\
\overline{D_1+D_2} & = & \overline{D_1}+\overline{D_2} \\
\\
\psi(x) & = & x \\
\psi(\lambda x.S) & = & \lambda x.\sigma(S)
\end{array}
\qquad
\begin{array}{rcl}
\sigma(\lambda k.C) & = & \overline{C} \\
\sigma(0) & = & 0 \\
\sigma(\alpha.T) & = & \alpha.\sigma(T) \\
\sigma(T_1+T_2) & = & \sigma(T_1)+\sigma(T_2) \\
\\
\\
\underline{k}[M] & = & M \\
\underline{\lambda b.BbK}[M] & = & \underline{K}[\psi(B)M] \\
\underline{\lambda b_1.T(\lambda b_2.b_1b_2K)}[M] & = & \underline{K}[M\sigma(T)]
\end{array}
$$

These functions are well-defined because the grammar for each category is unambiguous. To prove the completeness of the simulation we need a couple of lemmas. The first two state that the translation defined above is in fact an inverse.

Lemma 6. *For any term M, $\overline{[\![M]\!]k} = M$.*

Proof. We have $\overline{[\![M]\!]k} = \underline{k}[\sigma([\![M]\!])] = \sigma([\![M]\!])$ so we have to show that $\sigma([\![M]\!]) = M$ for all M. The proof follows by induction on the structure of M. □

In general, $\overline{M:k} \neq M$. Although it would be true for a classical translation, it does not hold in the algebraic case. Specifically, we have $(\alpha.M)L : k = \alpha.(ML) : k$ and $(M+N)L : k = ML + NL : K$, so the translation is not injective. However it is still true for values.

Lemma 7. *For any value V, $\overline{V:k} = V$.*

Proof. By induction on the structure of V. □

The third lemma that we need states that the inverse translation preserves reductions.

Lemma 8. *For any computation D, if $D \to_{a\cup\beta} D'$ then $\overline{D} \to^*_{l\cup\beta} \overline{D'}$.*

With these we can prove the completeness theorem.

Proof of Theorem 4. By using Lemma 8 for each step of the reduction, we get $\overline{[\![M]\!]k} \to^*_{l\cup\beta} \overline{V:k}$. By Lemma 6 and Lemma 7, this implies $M \to^*_{l\cup\beta} V$. □

To prove Lemma 8. we need several intermediary lemmas.

Lemma 9 (Substitution)**.** *The following are true.*

1. $\psi(B_1)[x := \psi(B)] = \psi(B_1[x := B])$
2. $\sigma(T)[x := \psi(B)] = \sigma(T[x := B])$
3. $\overline{C}[x := \psi(B)] = \overline{C[x := B]}$
4. $\underline{K}[M][x := \psi(B)] = \underline{K[x := B]}[M[x := \psi(B)]]$

Proof. By induction on the structure of B_1, T, C, and K. □

The next lemma states that we can compose two continuations K_1 and K_2 by replacing k by K_1 in K_2.

Lemma 10. *For all terms M and continuations K_1 and K_2, $\underline{K_1}[\underline{K_2}[M]] = \underline{K_2[k := K_1]}[M]$.*

Proof. By induction on the structure of K_2. □

Lemma 11. For all K and C, $\underline{K}[\overline{C}] = \overline{C[k := K]}$.

Proof. By induction on the structure of C, using Lemma 10 where necessary. □

The following lemma is essential to the preservation of reductions. It shows that reductions of a term M can always be carried in the context $\underline{K}[M]$.

Lemma 12. *For any continuation K and term M, if $M \to_{l\cup\beta} M'$, then $\underline{K}[M] \to_{l\cup\beta} \underline{K}[M']$.*

Proof. By induction on the structure of K. □

Lemma 13. *The following are true.*

- $\underline{K}[M_1 + M_2] \to^*_l \underline{K}[M_1] + \underline{K}[M_2]$
- $\underline{K}[\alpha.M] \to^*_l \alpha.\underline{K}[M]$
- $\underline{K}[0] \to^*_l 0$

Proof. We prove each statement by induction on K, using Lemma 12 where necessary. ☐

Lemma 14. *For any suspension T, if $T \to_a T'$ then $\sigma(T) \to_l \sigma(T')$.*

Proof. By induction on the reduction rule. Since T terms do not contain applications, the only cases possible are $L \cup \xi$, which are common to both languages. ☐

We now have the tools to finish the proof of 8.

Proof of Lemma 8. By induction on the reduction rule, using Lemmas 9, 11, 12, 13 and 14 where necessary ☐

4 Completeness of the call-by-name to call-by-value simulation

The simulation in this direction is similar to the other one, and uses the same techniques. The adjustments we have to make are the same as in the classical case, and deal mainly with our treatment of variables and applications. The CPS translation, as defined in [5], is the following.

$$\begin{aligned}
\{|x|\} &= x \\
\{|\lambda x.M|\} &= \lambda k.k(\lambda x.\{|M|\}) \\
\{|MN|\} &= \lambda k.\{|M|\}(\lambda b.b\{|N|\}k) \\
\{|0|\} &= 0 \\
\{|\alpha.M|\} &= \lambda k.(\alpha.\{|M|\})k \\
\{|M+N|\} &= \lambda k.(\{|M|\}+\{|N|\})k
\end{aligned}$$

Again, this translation simulates the reductions of a term M by the reductions of the term $\{|M|\}k$, where k is free.

Example 15. The reductions of the term $\mathsf{copy}(y+z)$ in λ_{alg} are simulated in λ_{lin} by the following reductions.

$$\begin{aligned}
\{|\mathsf{copy}(y+z)|\}k &= (\lambda k.\,\{|\mathsf{copy}|\}\,(\lambda b.\,b\{|y+z|\}k))\,k \\
&\to_{\beta_v} \{|\mathsf{copy}|\}\,(\lambda b.\,b\{|y+z|\}k) \\
&\to_{\beta_v} (\lambda b.\,b\{|y+z|\}k)\,(\lambda x.\,\{|\langle x,x\rangle|\}) \\
&\to_{\beta_v} (\lambda x.\,\{|\langle x,x\rangle|\})\,\{|y+z|\}k \\
&\to^*_{l\cup\beta} \{|\langle y+z,y+z\rangle|\}k
\end{aligned}$$

We see that the result is the one that corresponds to call-by-name. It is natural to ask how we were able to perform this cloning of the state $y+z$ in a call-by-value setting and how it can agree with the no-cloning theorem. The answer is that the CPS encoding of the term $y+z$ is $\{|y+z|\} = \lambda k.\,(x+y)k$, which is an abstraction. In the quantum point of view, we can interpret this as a program, or a specification, that *prepares* the quantum state $x+y$. Therefore this program can be duplicated.

The soundness of the simulations uses a similar colon translation.

$$
\begin{array}{rcl@{\qquad}rcl}
\Phi(\lambda x.M) & = & \lambda x.\{|M|\} & (\lambda x.M)N : K & = & \Phi(\lambda x.M)\{|N|\}K \\
\lambda x.M : K & = & K\Phi(\lambda x.M) & xN : K & = & x : (\lambda b.b\{|N|\}K) \\
x : K & = & xK & (MN)L : K & = & MN : \lambda b.b\{|L|\}K \\
0 : K & = & 0 & (0)N : K & = & 0 : K \\
\alpha.M : K & = & \alpha.(M : K) & (\alpha.M)N : K & = & \alpha.(MN) : K \\
M+N : K & = & M : K + N : K & (M+N)L : K & = & ML + NL : K
\end{array}
$$

Proposition 16 (Soundness [5]). *For any term M, if $M \to^*_{a\cup\beta} V$ then $\{|M|\}k \to^*_{l\cup\beta} V : k$.*

We will use the same procedure as in the previous section to show that the translation is also complete.

Theorem 17 (Completeness). *If $\{|M|\}k \to^*_{l\cup\beta} V : k$ then $M \to^*_{a\cup\beta} V$.*

Here is the grammar of the target language. It is closed under $\to_{l\cup\beta}$ reductions.

$$
\begin{array}{rcll}
C & ::= & KB \mid BSK \mid TK & \text{(base computations)} \\
D & ::= & C \mid 0 \mid \alpha.D \mid D_1 + D_2 & \text{(computation combinations)} \\
S & ::= & x \mid \lambda k.C & \text{(base suspensions)} \\
T & ::= & S \mid 0 \mid \alpha.T \mid T_1 + T_2 & \text{(suspension combinations)} \\
K & ::= & k \mid \lambda b.bSK & \text{(continuations)} \\
B & ::= & \lambda x.S & \text{(CPS-values)}
\end{array}
$$

Notice how x is now considered a suspension, not a CPS-value. This is because x is replaced by a suspension after beta-reducing a term of the form $(\lambda x.S)SK$. This is the main difference between the call-by-name and call-by-value CPS simulations. Other than that, it satisfies the same properties. In particular, we have the same indifference property.

Proposition 18 (Indifference [5]). *For any computations D and D', $D \to_{a\cup\beta} D'$ if and only if $D \to_{l\cup\beta} D'$. In particular, if $M \to^*_{a\cup\beta} V$ then $\{|M|\}k \to^*_{a\cup\beta} V : k$.*

We define the inverse translation using the following four functions.

$$
\begin{array}{rcl@{\qquad}rcl}
\overline{KB} & = & \underline{K}[\phi(B)] & \sigma(x) & = & x \\
\overline{BSK} & = & \underline{K}[\phi(B)\sigma(S)] & \sigma(\lambda k.C) & = & \overline{C} \\
\overline{TK} & = & \underline{K}[\sigma(T)] & \sigma(0) & = & 0 \\
\overline{0} & = & 0 & \sigma(\alpha.T) & = & \alpha.\sigma(T) \\
\overline{\alpha.D} & = & \alpha.\overline{D} & \sigma(T_1+T_2) & = & \sigma(T_1)+\sigma(T_2) \\
\overline{D_1+D_2} & = & \overline{D_1}+\overline{D_2} & & & \\
 & & & \underline{k}[M] & = & M \\
\phi(\lambda x.S) & = & \lambda x.\sigma(S) & \underline{\lambda b.bSK}[M] & = & \underline{K}[M\sigma(S)]
\end{array}
$$

To prove the completeness of the simulation we need analogous lemmas. Their proofs are similar, but we need to account for the changes mentioned above.

Lemma 19. *For any term M, $\overline{\{|M|\}k} = M$.*

Proof. We have $\overline{\{|M|\}k} = \underline{k}[\sigma(\{|M|\})] = \sigma(\{|M|\})$ so we have to show that $\sigma(\{|M|\}) = M$ for all M. The proof follows by induction on the structure of M. □

Lemma 20. *For any value V, $\overline{V : k} = V$.*

Proof. By induction on the structure of V. □

Lemma 21. *For any computation D, if* $D \to_{l\cup\beta} D'$ *then* $\overline{D} \to^*_{a\cup\beta} \overline{D'}$.

With these we can prove the completeness theorem.

Proof of Theorem 17. By using Lemma 21 for each step of the reduction, we get $\overline{\{|M|\}k} \to^*_{a\cup\beta} \overline{V : k}$. By Lemma 19 and Lemma 20, this implies $M \to^*_{a\cup\beta} V$. □

To prove Lemma 21, we need similar intermediary lemmas.

Lemma 22 (Substitution)**.** *The following are true.*

1. $\phi(B)[x := \sigma(S)] = \phi(B[x := S])$
2. $\sigma(T)[x := \sigma(S)] = \sigma(T[x := S])$
3. $\overline{C}[x := \sigma(S)] = \overline{C[x := S]}$
4. $\underline{K}[M][x := \sigma(S)] = \underline{K[x := S]}[M[x := \sigma(S)]]$

Proof. By induction on the structure of B, T, C and K. □

Lemma 23. *For all terms M and continuations* K_1 *and* K_2, $\underline{K_1}[\underline{K_2}[M]] = \underline{K_2[k := K_1]}[M]$.

Proof. By induction on the structure of K_2. □

Lemma 24. For all K and C, $\underline{K}[\overline{C}] = \overline{C[k := K]}$.

Proof. By induction on the structure of C, using Lemma 23 where necessary. □

Lemma 25. *For any continuation K and term M, if* $M \to_{a\cup\beta} M'$ *then* $\underline{K}[M] \to_{a\cup\beta} \underline{K}[M']$.

Proof. By induction on the structure of K. □

Lemma 26. *The following are true.*

- $\underline{K}[M_1 + M_2] \to^*_a \underline{K}[M_1] + \underline{K}[M_2]$
- $\underline{K}[\alpha.M] \to^*_a \alpha.\underline{K}[M]$
- $\underline{K}[0] \to^*_a 0$

Proof. We prove each statement by induction on K, using Lemma 25 where necessary. □

Lemma 27. *For any suspension T, if* $T \to_l T'$ *then* $\sigma(T) \to_a \sigma(T')$.

Proof. By induction on the reduction rule. Since T terms do not contain applications, the only cases possible are $L \cup \xi$, which are common to both languages. □

We can now prove Lemma 21.

Proof of Lemma 21. By induction on the reduction rule, using Lemmas 22, 24, 25, 26 and 27 where necessary. Notice that the rules $\xi_{\lambda_{lin}}$ and A_r are not applicable since arguments in the target language are always base terms. □

5 Discussion and conclusion

We showed the completeness of two CPS translations simulating algebraic lambda calculi introduced in [5]. We did so by using techniques inspired from [13] to define an inverse translation and showing that it preserves reductions.

Our treatment differs from Sabry and Wadler's [13], not only because they work in a non-algebraic setting, but also because they decompile continuations into abstractions. For example, they defined $\underline{\lambda b.Bbk}[M]$ as let $b = M$ in $\phi(\mathrm{B})b$. This required the modification of the source language and led to the consideration of the *computational lambda calculus* [10] as a source language instead. We avoid this by directly substituting and eliminating variables introduced by the forward translation, which allows us to obtain an exact inverse.

However, the translations defined in [13] satisfy an additional property: they form a Galois connection. Our translations fail to satisfy one of the four required criteria to be a Galois connection, namely that $\overline{N} : k$ reduces to N. It would be interesting to see if we can accomplish the same thing in the algebraic case, all while dealing with the problems mentioned above.

Originally, the work in [5] also considers another version of λ_{lin} and λ_{alg} with *algebraic equalities* instead of *algebraic reductions*. For example, we could go back and forth between $M + N - N$ and M, which is not permitted by the rules we presented above. Algebraic equalities can be formulated as the symmetric closure of the algebraic reductions $\to_a$ and $\to_l$. The resulting four systems $\lambda_{lin}^{\to}$, $\lambda_{alg}^{\to}$, $\lambda_{lin}^{=}$, and $\lambda_{alg}^{=}$ have all been shown to simulate each other. The results of this paper can be extended to these systems as well.

Acknowledgments Many thanks to Alejandro Díaz-Caro, Benoît Valiron, Pablo Arrighi, and Christophe Calvès for fruitful discussions and suggestions. This work is supported by the CNRS - INS2I PEPS project QuAND.

References

[1] Thorsten Altenkirch & Jonathan J. Grattage (2005): *A functional quantum programming language*. In: *Proceedings of LICS-2005*, IEEE Computer Society, pp. 249–258, doi:10.1109/LICS.2005.1.

[2] Pablo Arrighi & Alejandro Díaz-Caro (2011): *Scalar System F for Linear-Algebraic λ-Calculus: Towards a Quantum Physical Logic*. In Bob Coecke, Prakash Panangaden & Peter Selinger, editors: *Proceedings of QPL-2009*, *Electronic Notes in Theoretical Computer Science* 270/2, Elsevier, pp. 219–229, doi:10.1016/j.entcs.2011.01.033.

[3] Pablo Arrighi & Gilles Dowek (2004): *A Computational Definition of the Notion of Vectorial Space*. In Narciso Martí-Oliet, editor: *Proceedings of WRLA-2004*, *Electronic Notes in Theoretical Computer Science* 117, Elsevier, pp. 249–261, doi:10.1016/j.entcs.2004.06.013.

[4] Pablo Arrighi & Gilles Dowek (2008): *Linear-algebraic lambda-calculus: higher-order, encodings, and confluence*. In Andrei Voronkov, editor: *Proceedings of RTA-2008*, *Lecture Notes in Computer Science* 5117, Springer, pp. 17–31, doi:10.1007/978-3-540-70590-1_2.

[5] Alejandro Díaz-Caro, Simon Perdrix, Christine Tasson & Benoît Valiron (2011): *Call by value, call by name and the vectorial behaviour of algebraic λ-calculus*. (Submitted) http://membres-liglab.imag.fr/diazcaro/simulations.pdf.

[6] Thomas Ehrhard (2005): *Finiteness spaces*. *Mathematical Structures in Computer Science* 15(4), pp. 615–646, doi:10.1017/S0960129504004645.

[7] Thomas Ehrhard (2010): *A Finiteness Structure on Resource Terms*. In: *Proceedings of LICS-2010*, IEEE Computer Society, pp. 402–410, doi:10.1109/LICS.2010.38.

[8] Thomas Ehrhard & Laurent Regnier (2003): *The differential lambda-calculus*. *Theoretical Computer Science* 309(1), pp. 1–41, doi:10.1016/S0304-3975(03)00392-X.

[9] John Hatcliff & Olivier Danvy (1994): *A Generic Account of Continuation-Passing Styles*. In: *Proceedings of the Twenty-first Annual ACM Symposium on Principles of Programming Languages*, ACM Press, pp. 458–471.

[10] Eugenio Moggi (1989): *Computational Lambda-Calculus and Monads*. In: *Proceedings of LICS-1989*, IEEE Computer Society, pp. 14–23.

[11] Simon Perdrix (2008): *Quantum Entanglement Analysis Based on Abstract Interpretation*. In: *Proceedings of SAS-2008*, pp. 270–282, doi:10.1007/978-3-540-69166-2_18.

[12] Gordon D. Plotkin (1975): *Call-by-name, call-by-value and the λ-calculus*. *Theoretical Computer Science* 1(2), pp. 125–159.

[13] Amr Sabry & Philip Wadler (1996): *A Reflection on Call-by-Value*. *ACM Transactions on Programming Languages and Systems* 19, pp. 13–24, doi:10.1145/232627.232631.

[14] Peter Selinger (2004): *Towards a quantum programming language*. In: *Mathematical Structures in Computer Science*, pp. 527–586, doi:10.1017/S0960129504004256.

[15] Peter Selinger & Benoît Valiron (2006): *A lambda calculus for quantum computation with classical control*. *Mathematical Structures in Computer Science* 16(3), pp. 527–552, doi:10.1017/S0960129506005238.

[16] Benoît Valiron (2010): *Semantics of a typed algebraic lambda-calculus*. In S. Barry Cooper, Prakash Panangaden & Elham Kashefi, editors: *Proceedings DCM-2010*, *Electronic Proceedings in Theoretical Computer Science* 26, Open Publishing Association, pp. 147–158, doi:10.4204/EPTCS.26.14.

[17] Lionel Vaux (2007): *On Linear Combinations of Lambda-Terms*. In Franz Baader, editor: *Proceedings of RTA-2007*, *Lecture Notes in Computer Science* 4533, Springer, pp. 374–388, doi:10.1007/978-3-540-73449-9_28.

[18] Lionel Vaux (2009): *The algebraic lambda calculus*. *Mathematical Structures in Computer Science* 19(5), pp. 1029–1059, doi:10.1017/S0960129509990089.

[19] William K. Wootters & Wojciech .H. Zurek (1982): *A Single Quantum Cannot be Cloned*. *Nature* 299, pp. 802–803, doi:10.1038/299802a0.