Confluence via strong normalisation in an algebraic λ-calculus with rewriting

Pablo Buiras
Universidad Nacional de Rosario, FCEIA, Argentina

Alejandro Díaz-Caro
Université de Grenoble, LIG, France

Mauro Jaskelioff
Universidad Nacional de Rosario, FCEIA, Argentina
CIFASIS-CONICET, Argentina

The linear-algebraic λ-calculus and the algebraic λ-calculus are untyped λ-calculi extended with arbitrary linear combinations of terms. The former presents the axioms of linear algebra in the form of a rewrite system, while the latter uses equalities. When given by rewrites, algebraic λ-calculi are not confluent unless further restrictions are added. We provide a type system for the linear-algebraic λ-calculus enforcing strong normalisation, which gives back confluence. The type system allows an interpretation in System F.

1 Introduction

Two algebraic versions of λ-calculus arose independently in different contexts: the linear-algebraic λ-calculus (λ_{lin}) [4] and the algebraic λ-calculus (λ_{alg}) [20]. The former was first introduced as a candidate λ-calculus for quantum computation; a linear combination of terms reflect the phenomenon of superposition, *i.e.* the capacity for a quantum system to be in two or more states at the same time. The latter was introduced in the context of linear logic, as a fragment of the differential λ-calculus [11], an extension to λ-calculus with a *differential operator* making the resource-aware behaviour explicit. This extension produces a calculus where superposition of terms may happen. Then λ_{alg} can be seen as *a differential λ-calculus without the differential operator*. In recent years, there has been growing research interest in these two calculi and their variants, as they could provide an explicit link between linear logic and linear algebra [1, 2, 3, 6, 7, 8, 9, 10, 11, 14, 15, 17, 19].

The two languages, λ_{lin} and λ_{alg}, are rather similar: they both merge the untyped λ-calculus –higher-order computation in its simplest and most general form– with linear algebraic constructions –sums and scalars subject to the axioms of vector spaces. In both languages, functions which are linear combinations of terms are interpreted pointwise: $(\alpha.\mathbf{f}+\beta.\mathbf{g})\ x = \alpha.(\mathbf{f}\ x)+\beta.(\mathbf{g}\ x)$, where "." is the external product. However, they differ in their treatment of arguments. In λ_{lin}, the reduction strategy is call-by-value (or strictly speaking, call-by-variables or abstractions) and, in order to deal with the algebraic structure, any function is considered to be a linear map: $\mathbf{f}\ (\alpha.x+\beta.y)$ reduces to $\alpha.(\mathbf{f}\ x)+\beta.(\mathbf{f}\ y)$, reflecting the fact that any quantum evolution is a linear map. On the other hand, λ_{alg} has a call-by-name strategy: $(\lambda x.\mathbf{t})\ \mathbf{r}$ reduces to $\mathbf{t}[\mathbf{r}/x]$, with no restrictions on $\mathbf{r}$. As a consequence, the reductions are different as illustrated by the following example. In λ_{lin}, $(\lambda x.x\ x)\ (\alpha.y+\beta.z)$ reduces to $\alpha.(y\ y)+\beta.(z\ z)$ while in λ_{alg}, $(\lambda x.x\,x)\ (\alpha.y+\beta.z)$ reduces to $(\alpha.y+\beta.z)\ (\alpha.y+\beta.z) = \alpha^2.(y\,y)+\alpha\times\beta.(y\,z)+\beta\times\alpha.(z\,y)+\beta^2.(z\,z)$. Nevertheless, they can simulate each other by means of an extension of the well-known CPS transform that maps call-by-value to call-by-name and vice versa [6].

Another more fundamental difference between them is the way the algebraic part of the calculus is treated. In λ_{lin}, the algebraic structure is captured by a rewrite system, whereas in λ_{alg} terms are identified

Submitted to:
LSFA 2011

up to algebraic equivalence. Thus, while $\mathbf{t}+\mathbf{t}$ reduces to $2.\mathbf{t}$ in λ_{lin}, they are regarded as the same term in λ_{alg}. Using a rewrite system allows λ_{lin} to expose the algebraic structure in its canonical form, but it is not without some confluence issues. Consider the term $Y_{\mathbf{b}} = (\lambda x.\,\mathbf{b}+x\,x)\ (\lambda x.\,\mathbf{b}+x\,x)$. Then $Y_{\mathbf{b}}$ reduces to $\mathbf{b}+Y_{\mathbf{b}}$, so the term $Y_{\mathbf{b}}+Y_{\mathbf{b}}$ in λ_{lin} reduces to $2.Y_{\mathbf{b}}$ but also to $\mathbf{b}+Y_{\mathbf{b}}+Y_{\mathbf{b}}$ and thus to $\mathbf{b}+2.Y_{\mathbf{b}}$. Note that $2.Y_{\mathbf{b}}$ can only produce an even number of $\mathbf{b}$'s whereas $\mathbf{b}+2.Y_{\mathbf{b}}$ will only produce an odd number of $\mathbf{b}$'s, breaking confluence. In λ_{alg}, on the other hand, $\mathbf{b}+2.Y_{\mathbf{b}} = \mathbf{b}+Y_{\mathbf{b}}+Y_{\mathbf{b}}$, solving the problem. The canonical solution in λ_{lin} is to disallow diverging terms. In [6] it is assumed that confluence can be proved in some unspecified way; then, sets of confluent terms are defined and used in the hypotheses of several theorems that require confluence. In the original λ_{lin} paper [4], certain restrictions are introduced to the rewrite system, such as having $\alpha.\mathbf{t}+\beta.\mathbf{t}$ reduce to $(\alpha+\beta).\mathbf{t}$ only when $\mathbf{t}$ is in closed normal form. The rewrite system has been proved locally confluent [3], so by ensuring strong normalisation we obtain confluence [18]. This approach has been followed in other works [1, 2, 3, 7] which discuss similar type systems with strong normalisation. While these type systems give us some information about the terms, they also impose some undesirable restrictions:

- In [1] two type systems are presented: a straightforward extension of System F, which only allows typing $\mathbf{t}+\mathbf{r}$ when both $\mathbf{t}$ and $\mathbf{r}$ have the same type, and a type system with scalars in the types, which keep track of the scalars in the terms, but is unable to lift the previous restriction.
- In [7] a type system solving the previous issue that can be interpreted in System F is introduced. However, it only considers the additive fragment of λ_{lin}: scalars are removed from the calculus, considerably simplifying the rewrite system.
- In [2, 3] a combination of the two previous approaches is set up: a type system where the types can be weighted and added together is devised. While this is a novel approach, the introduction of type-level scalars makes it difficult to relate it to System F or any other well-known theory.

In this paper, we propose an algebraic λ-calculus featuring term-rewriting semantics and a type system strong enough to prove confluence, while remaining expressive and retaining the interpretation in System F from previous works. In addition, the type system provides us with lower bounds for the scalars involved in the terms.

Outline. In section 2 the typed version of λ_{lin}, called λ_{CA}, is presented. Section 3 is devoted to proving that the system possesses some basic properties, namely subject reduction and strong normalisation, which entails the confluence of the calculus. Section 4 shows an abstract interpretation of λ_{CA} into *Additive*, the additive fragment of λ_{lin}. Finally, section 5 concludes.

2 The Calculus

We introduce the calculus λ_{CA}, which extends explicit System F [16] with linear combinations of λ-terms. Figure 1 shows the abstract syntax of types and terms of the calculus, where the terms are based on those of λ_{lin} [4]. Our choice of explicit System F instead of a Curry style presentation [1, 7] stems from the fact that, as shown in [3], the "factorisation" reduction rules (*cf.* Group F in Fig. 2) in a Curry style setting introduce some imprecisions.

We use the convention that abstraction binds as far to the right as possible and that application binds more strongly than sums and scalar multiplication. However, we will freely add parentheses whenever confusion might arise. Metavariables $\mathbf{t}, \mathbf{r}$, $\mathbf{s}$, $\mathbf{u}$, and $\mathbf{v}$ will range over terms.

Types:	*Terms:*
$\mathsf{T} ::= \mathsf{U} \mid \mathsf{T}+\mathsf{T} \mid \bar{0}$	$\mathbf{t} ::= \mathbf{b} \mid \mathbf{t}\ \mathbf{t} \mid \mathbf{t}@\mathsf{U} \mid \mathbf{0} \mid \alpha.\mathbf{t} \mid \mathbf{t}+\mathbf{t}$
$\mathsf{U} ::= X \mid \mathsf{U} \to \mathsf{T} \mid \forall X.\mathsf{U}$	$\mathbf{b} ::= x \mid \lambda x\colon \mathsf{U}.\ \mathbf{t} \mid \Lambda X.\ \mathbf{t}$

Figure 1: Types and Terms of λ_{CA}

Terms known as *basis* terms (nonterminal **b** in Fig. 1) are the only ones that can substitute a variable in a β-reduction step. This "call-by-**b**"[1] strategy plays an important role when interacting with the linearity from linear-algebra, *e.g.* the term $(\lambda x : U.x\,x)\ (y+z)$ may reduce to $(y+z)\ (y+z) \to^* y\,y+y\,z+z\,y+z\,z$ in a call-by-name setting, however if we decide that abstractions should behave as linear maps, then this call-by-**b** strategy can be used and the previous term will reduce to $(\lambda x : U.x\,x)\ y + (\lambda x : U.x\,x)\ z$ and then to $y\,y+z\,z$.

For the same reason, we also make a distinction between *unit* types (nonterminal U in Fig. 1) and general types. Unit types cannot include sums of types except in the codomain of a function type, and they contain all types of System F. General types are either sums of unit types or the special type $\bar{0}$. Basis terms can only be assigned unit types. Scalars (denoted by greek letters) are nonnegative real numbers. There are no scalars at the type level, but we introduce the following notation: for an integer $n \geq 0$, we will write $n.T$ for the type $T+T+\cdots+T$ (n times), considering $0.T = \bar{0}$. We may also use the summation symbol $\sum_{i=1}^{n} T_i$, with $\sum_{i=1}^{0} T_i = \bar{0}$. Metavariables T,R, and S will range over general types and U,V, and W over unit types.

Figure 2 defines the term-rewriting system (TRS) for λ_{CA}, which consists of directed versions of the vector-space axioms and β-reduction for both kinds of abstractions. All reductions are performed modulo associativity and commutativity of the $+$ operator. It is essentially the TRS of λ_{lin} [4], with an extra type-application rule. As usual, $\to^*$ denotes the reflexive transitive closure of the reduction relation $\to$.

Substitution for term and type variables (written $\mathbf{t}[\mathbf{b}/x]$ and $\mathbf{t}[U/X]$, respectively) are defined in the usual way to avoid variable capture. Substitution behaves like a linear operator when acting on linear combinations, *e.g.* $(\alpha.\mathbf{t}+\beta.\mathbf{r})[\mathbf{b}/x] = \alpha.\mathbf{t}[\mathbf{b}/x]+\beta.\mathbf{r}[\mathbf{b}/x]$.

Figure 3 defines the notion of type equivalence and shows the typing rules for the system. The typing judgement $\Gamma \vdash \mathbf{t} : T$ means that the term $\mathbf{t}$ can be assigned type T in the context Γ, with the usual definition of typing context from System F. As a consequence of the design decision of only allowing basis terms to substitute variables in a β-reduction, typing contexts bind term variables to unit types.

Using standard arrow elimination instead of rule $\to_E$ would restrict the calculus, since it would force $\mathbf{t}$ to a be sum of arrows of the same type $U \to T$. The same would happen with the argument type U: for the term $(\mathbf{t}_1+\mathbf{t}_2)\ (\mathbf{r}_1+\mathbf{r}_2)$ to be well-typed, $\mathbf{t}_1$ and $\mathbf{t}_2$ would need to have the same type, and also $\mathbf{r}_1$ and $\mathbf{r}_2$.

In the rule $\to_E$ presented in Fig. 3 we relax this restriction and we allow to have different T's. Continuing with the example, this allows $\mathbf{t}_1$ and $\mathbf{t}_2$ to have different types, provided that they are arrows with the same source type U.

Example Let $\Gamma \vdash \mathbf{b}_1 : U$, $\Gamma \vdash \mathbf{b}_2 : U$, $\Gamma \vdash \lambda x.\mathbf{t} : U \to T$ and $\Gamma \vdash \lambda y.\mathbf{r} : U \to R$. Then

$$\frac{\Gamma \vdash (\lambda x.\mathbf{t})+(\lambda y.\mathbf{r}) : (U \to T)+(U \to R) \qquad \Gamma \vdash \mathbf{b}_1+\mathbf{b}_2 : U+U}{\Gamma \vdash ((\lambda x.\mathbf{t})+(\lambda y.\mathbf{r}))\ (\mathbf{b}_1+\mathbf{b}_2) : T+T+R+R} \to_E$$

[1] The set of terms in **b** is not the set of values of λ_{CA}, so technically it is not "call-by-value".

Group E:

$\mathbf{u}+\mathbf{0} \to \mathbf{u}$

$0.\mathbf{u} \to \mathbf{0}$

$1.\mathbf{u} \to \mathbf{u}$

$\alpha.\mathbf{0} \to \mathbf{0}$

$\alpha.(\beta.\mathbf{u}) \to (\alpha \times \beta).\mathbf{u}$

$\alpha.(\mathbf{u}+\mathbf{v}) \to \alpha.\mathbf{u}+\alpha.\mathbf{v}$

Group F:

$\alpha.\mathbf{u}+\beta.\mathbf{u} \to (\alpha+\beta).\mathbf{u}$

$\alpha.\mathbf{u}+\mathbf{u} \to (\alpha+1).\mathbf{u}$

$\mathbf{u}+\mathbf{u} \to 2.\mathbf{u}$

Group A:

$(\mathbf{u}+\mathbf{v})\,\mathbf{w} \to \mathbf{u}\,\mathbf{w}+\mathbf{v}\,\mathbf{w}$

$\mathbf{w}\,(\mathbf{u}+\mathbf{v}) \to \mathbf{w}\,\mathbf{u}+\mathbf{w}\,\mathbf{v}$

$(\alpha.\mathbf{u})\,\mathbf{v} \to \alpha.(\mathbf{u}\,\mathbf{v})$

$\mathbf{v}\,(\alpha.\mathbf{u}) \to \alpha.(\mathbf{v}\,\mathbf{u})$

$\mathbf{0}\,\mathbf{u} \to \mathbf{0}$

$\mathbf{u}\,\mathbf{0} \to \mathbf{0}$

β-reduction:

$(\lambda x:U.\,\mathbf{t})\,\mathbf{b} \to \mathbf{t}[\mathbf{b}/x]$

$(\Lambda X.\mathbf{t})@U \to \mathbf{t}[U/X]$

$$\frac{\mathbf{t} \to \mathbf{t}'}{\mathbf{t}+\mathbf{r} \to \mathbf{t}'+\mathbf{r}} \qquad \frac{\mathbf{r} \to \mathbf{r}'}{\mathbf{t}+\mathbf{r} \to \mathbf{t}+\mathbf{r}'} \qquad \frac{\mathbf{t} \to \mathbf{t}'}{\alpha.\mathbf{t} \to \alpha.\mathbf{t}'}$$

$$\frac{\mathbf{t} \to \mathbf{t}'}{\mathbf{t}\mathbf{r} \to \mathbf{t}'\mathbf{r}} \qquad \frac{\mathbf{r} \to \mathbf{r}'}{\mathbf{t}\mathbf{r} \to \mathbf{t}\mathbf{r}'} \qquad \frac{\mathbf{t} \to \mathbf{t}'}{\mathbf{t}@T \to \mathbf{t}'@T}$$

$$\frac{\mathbf{t} \to \mathbf{t}'}{\lambda x:U.\mathbf{t} \to \lambda x:U.\mathbf{t}'} \qquad \frac{\mathbf{t} \to \mathbf{t}'}{\Lambda X.\mathbf{t} \to \Lambda X.\mathbf{t}'}$$

Figure 2: One-step Reduction Relation $\to$

Notice that $((\lambda x.\,\mathbf{t})+(\lambda y.\,\mathbf{r}))\;(\mathbf{b}_1+\mathbf{b}_2) \to^* \underbrace{(\lambda x.\,\mathbf{t})\;\mathbf{b}_1}_{T}+\underbrace{(\lambda x.\,\mathbf{t})\;\mathbf{b}_2}_{T}+\underbrace{(\lambda y.\,\mathbf{r})\;\mathbf{b}_1}_{R}+\underbrace{(\lambda y.\,\mathbf{r})\;\mathbf{b}_2}_{R}$

On the other hand, allowing different U's is sightly more complex: on account of the distributive rules (Group A) it is required that all the arrows in the first addend start with a type which has to be the type of all the addends in the second term. For example, if the given term is $(\mathbf{t}+\mathbf{r})\;(\mathbf{b}_1+\mathbf{b}_2)$, the terms $\mathbf{t}$ and $\mathbf{r}$ have to be able to receive both $\mathbf{b}_1$ and $\mathbf{b}_2$ as arguments. This could be done by taking advantage of polymorphism, but the arrow-elimination rule would become much more complex since it would have to do both arrow-elimination and forall-elimination at the same time. Although this approach has been shown to be viable [3], we delay the modification of the rule to future work, and keep the simpler but more restricted version, which is enough for the aims of the present paper.

The main novelty of the calculus is its treatment of scalars (rule SI). In order to avoid having scalars at the type level, when typing $\alpha.\mathbf{t}$ we take the floor of the term-level scalar α and assign the type $\lfloor\alpha\rfloor.T$ to the term, which is a sum of Ts. The intuitive interpretation is that a type $n.T$ provides a lower bound for the "amount" of $\mathbf{t}:T$ in the term.

The rest of the rules are straightforward. The $\forall_E$ and $\forall_I$ rules enforce the restriction that only unit types can participate in type abstraction and type application.

Type Equivalence: Equivalence is the least congruence $\equiv$ s.t.

$$T+\bar{0}\equiv T, \qquad T+R\equiv R+T, \qquad T+(R+S)\equiv(T+R)+S$$

Typing rules:

$$\frac{}{\Gamma,x:U\vdash x:U}\ \text{AX} \qquad \frac{}{\Gamma\vdash\mathbf{0}:\bar{0}}\ \text{AX}_{\bar{0}}$$

$$\frac{\Gamma\vdash\mathbf{t}:\sum_{i=1}^{\alpha}(U\to T_i)\qquad\Gamma\vdash\mathbf{r}:\beta.U}{\Gamma\vdash\mathbf{t}\mathbf{r}:\sum_{i=1}^{\alpha}(\beta.T_i)}\ \to_{\text{E}} \qquad \frac{\Gamma,x:U\vdash\mathbf{t}:T}{\Gamma\vdash\lambda x:U.\mathbf{t}:U\to T}\ \to_{\text{I}}$$

$$\frac{\Gamma\vdash\mathbf{t}:\forall X.U}{\Gamma\vdash\mathbf{t}@V:U[V/X]}\ \forall_E \qquad \frac{\Gamma\vdash\mathbf{t}:U\qquad X\notin\text{FV}(\Gamma)}{\Gamma\vdash\Lambda X.\mathbf{t}:\forall X.U}\ \forall_I$$

$$\frac{\Gamma\vdash\mathbf{t}:T\qquad\Gamma\vdash\mathbf{r}:R}{\Gamma\vdash\mathbf{t}+\mathbf{r}:T+R}\ +\text{I} \qquad \frac{\Gamma\vdash\mathbf{t}:T}{\Gamma\vdash\alpha.\mathbf{t}:\lfloor\alpha\rfloor.T}\ \text{SI}$$

$$\frac{\Gamma\vdash\mathbf{t}:T\qquad T\equiv R}{\Gamma\vdash\mathbf{t}:R}\ \text{EQ}$$

Figure 3: λ_{CA} Type Equivalence and Typing Rules

3 Properties

3.1 Subject Reduction with Imprecise Types

A basic soundness property in a typed calculus is the guarantee that types will be preserved by reduction. However, in λ_{CA} types are imprecise about the "amount" of each type in a term. For example, let $\Gamma\vdash\mathbf{t}:T$ and consider the term $\mathbf{s}=(0.9).\mathbf{t}+(1.1).\mathbf{t}$. We see that $\Gamma\vdash\mathbf{s}:T$ and $\mathbf{s}\to^{*}2.\mathbf{t}$, but $\Gamma\vdash 2.\mathbf{t}:T+T$. In this example a term with type T reduces to a term with type $T+T$, proving that strict subject reduction does not hold for λ_{CA}. Nevertheless, we prove a similar property: as reduction progresses, types are either preserved or *strengthened*, *i.e.* they become more precise according to the relation $\preccurlyeq$ (*cf.* Fig. 4). This entails that the derived type for a term is a lower-bound (with respect to $\preccurlyeq$) for the actual type of the reduced term.

Theorem 3.1 (Subject Reduction up to $\preccurlyeq$) *For any terms* $\mathbf{t}$ *and* $\mathbf{t}'$*, context* Γ *and type* T*, if* $\mathbf{t}\to\mathbf{t}'$ *and* $\Gamma\vdash\mathbf{t}:T$ *then there exists some type* R *such that* $\Gamma\vdash\mathbf{t}':R$ *and* $T\preccurlyeq R$*, where the relation* $\preccurlyeq$ *is inductively defined in Fig. 4.*

Intuitively, $T\preccurlyeq R$ (R is at least as precise as T) means that there are more summands of the same type in R than in T, *e.g.* $A\preccurlyeq A+A$ for a fixed type A. Note that $\preccurlyeq$ is not the trivial order relation: although $T\preccurlyeq T+R$ for any R (because $T\equiv T+0.R\preccurlyeq T+1.R\equiv T+R$), type T cannot disappear from the sum; if $T\preccurlyeq S$, then T will always appear at least once in S (and possibly more than once).

$$\frac{\alpha \le \beta}{\alpha.T \preccurlyeq \beta.T}\ \textsc{Sub-Wk} \qquad \frac{T \equiv R}{T \preccurlyeq R}\ \textsc{Sub-Eq}$$

$$\frac{T \preccurlyeq S \qquad S \preccurlyeq R}{T \preccurlyeq R}\ \textsc{Sub-Tr} \qquad \frac{T_1 \preccurlyeq T_2 \qquad S_1 \preccurlyeq S_2}{T_1 + S_1 \preccurlyeq T_2 + S_2}\ \textsc{Sub-Ctxt}_1$$

$$\frac{U_2 \preccurlyeq U_1 \qquad T_1 \preccurlyeq T_2}{U_1 \to T_1 \preccurlyeq U_2 \to T_2}\ \textsc{Sub-Ctxt}_2 \qquad \frac{T \preccurlyeq R}{\forall X.T \preccurlyeq \forall X.R}\ \textsc{Sub-Ctxt}_3$$

Figure 4: Inductive definition of the relation $\preccurlyeq$

The proof of this theorem requires several preliminary lemmas. We give the most important of them and some details about the proof of the theorem.

Lemma 3.1 (Generation lemmas) *Let T be a type and Γ a typing context.*

1. *For arbitrary terms $\mathbf{u}$ and $\mathbf{v}$, if $\Gamma \vdash \mathbf{u}\,\mathbf{v} : T$, then there exist natural numbers α, β, and types $U \in \mathsf{U}, T_1, \dots, T_\alpha \in \mathsf{T}$, such that $\Gamma \vdash \mathbf{u} : \sum_{i=1}^{\alpha}(U \to T_i)$ and $\Gamma \vdash \mathbf{v} : \beta.U$ with $\sum_{i=1}^{\alpha}(\beta.T_i) \equiv T$.*
2. *For any term $\mathbf{t}$ and unit type U, if $\Gamma \vdash \lambda x : U.\mathbf{t} : T$, then there exists a type R such that $\Gamma, x : U \vdash \mathbf{t} : R$ and $U \to R \equiv T$.*
3. *For any terms $\mathbf{u}$ and $\mathbf{v}$, if $\Gamma \vdash \mathbf{u} + \mathbf{v} : T$, then there exist types R and S such that $\Gamma \vdash \mathbf{u} : R$ and $\Gamma \vdash \mathbf{v} : S$, with $R + S \equiv T$.*
4. *For any term $\mathbf{u}$ and nonnegative real number α, if $\Gamma \vdash \alpha.\mathbf{u} : T$, then there exists a type R such that $\Gamma \vdash \mathbf{u} : R$ and $\lfloor\alpha\rfloor.R \equiv T$.*
5. *For any term $\mathbf{t}$, if $\Gamma \vdash \Lambda X.\mathbf{t} : T$, then there exists a type R such that $\Gamma \vdash \mathbf{t} : R$ and $\forall X.R \equiv T$ with $X \notin FV(\Gamma)$.*
6. *For any term $\mathbf{t}$ and unit type U, if $\Gamma \vdash \mathbf{t}@U : T$, then there exists a type V such that $\Gamma \vdash \mathbf{t} : \forall X.V$ and $V[U/X] \equiv T$.* ∎

The following lemma is standard in proofs of subject reduction for System F-like systems [13, 5]. It ensures that well-typedness is preserved under substitution on type and term variables.

Lemma 3.2 (Substitution lemma) *For any term $\mathbf{t}$, basis term $\mathbf{b}$, context Γ, unit type U and type T,*

1. *If $\Gamma \vdash \mathbf{t} : T$, then $\Gamma[U/X] \vdash \mathbf{t}[U/X] : T[U/X]$.*
2. *If $\Gamma, x : U \vdash \mathbf{t} : T$ and $\Gamma \vdash \mathbf{b} : U$, then $\Gamma \vdash \mathbf{t}[\mathbf{b}/x] : T$.* ∎

Now we can give some details about the proof of Theorem 3.1.

Proof of Theorem 3.1 (Subject Reduction up to $\preccurlyeq$) By induction on the reduction relation $\to$. We check that every reduction rule preserves the type up to the relation $\preccurlyeq$. In each case, we first apply one or more generation lemmas to the left-hand side of the rule. Then we construct a type for the right-hand side which is either more precise (in the sense of relation $\preccurlyeq$) or equivalent to that of the left-hand side.

For illustration purposes, we show the proof of the case corresponding to the rewrite rule $\alpha.\mathbf{t}+\beta.\mathbf{t} \to (\alpha+\beta).\mathbf{t}$.

We must prove that for any term $\mathbf{t}$, nonnegative real numbers α and β, context Γ and type T, if $\Gamma \vdash \alpha.\mathbf{t}+\beta.\mathbf{t} : T$ then $\Gamma \vdash (\alpha+\beta).\mathbf{t} : R$ with $T \preccurlyeq R$.

By lemma 3.1.3, there exist T_1, T_2 such that $\Gamma \vdash \alpha.\mathbf{t} : T_1$ and $\Gamma \vdash \beta.\mathbf{t} : T_2$, with $T_1 + T_2 \equiv T$. Also by lemma 3.1.4, there exist R_1, R_2 such that $\Gamma \vdash \mathbf{t} : R_1$ with $\lfloor\alpha\rfloor.R_1 \equiv T_1$, and $\Gamma \vdash \mathbf{t} : R_2$ with $\lfloor\beta\rfloor.R_2 \equiv T_2$. Then from $\Gamma \vdash \mathbf{t} : R_1$ we can derive the sequent $\Gamma \vdash (\alpha+\beta).\mathbf{t} : \lfloor\alpha+\beta\rfloor.R_1$ using rule SI.

We will now prove that $T \preccurlyeq \lfloor\alpha+\beta\rfloor.R_1$. Since R_1 and R_2 are both types for $\mathbf{t}$, we have $R_1 \equiv R_2$ so $\lfloor\alpha+\beta\rfloor.R_1 \succcurlyeq (\lfloor\alpha\rfloor+\lfloor\beta\rfloor).R_1 \equiv \lfloor\alpha\rfloor.R_1 + \lfloor\beta\rfloor.R_1 \equiv \lfloor\alpha\rfloor.R_1 + \lfloor\beta\rfloor.R_2 \equiv T_1 + T_2 \equiv T$. Therefore, we conclude $T \preccurlyeq \lfloor\alpha+\beta\rfloor.R_1$. ∎

3.2 Strong Normalisation

In this section, we prove the strong normalisation property for λ_{CA}. That is, we show that all possible reductions for well-typed terms are finite. We use the standard notion of *reducibility candidates* [12, Chapter 14], extended to account for linear combinations of terms. Confluence follows as a corollary. Notice that we cannot reuse the proofs of previous typed versions of λ_{lin} (*e.g.* [1, 7]) since in [1] only terms of the same type can be added together, and in [7] the calculus under consideration is a fragment of λ_{CA}. Therefore, none of them have the same set of terms as λ_{CA}.

A closed term in λ_{CA} is a *value* if it is an abstraction, a sum of values or a scalar multiplied by a value, *i.e.* values are closed terms that conform to the following grammar:

$$\mathbf{v} ::= \lambda x \colon U.\mathbf{t} \mid \Lambda X.\mathbf{t} \mid \mathbf{v}+\mathbf{v} \mid \alpha.\mathbf{v}$$

If a closed term is not a value, it is said to be *neutral*. A term $\mathbf{t}$ is *normal* if it has no reducts, *i.e.* there is no term $\mathbf{s}$ such that $\mathbf{t} \to \mathbf{s}$. A *normal form* for a term $\mathbf{t}$ is a normal term $\mathbf{t}'$ such that $\mathbf{t} \to^* \mathbf{t}'$. We define $\mathrm{Red}(\mathbf{t})$ as the set of reducts of $\mathbf{t}$ reachable in one step.

A term $\mathbf{t}$ is *strongly normalising* if there are no infinite reduction sequences starting from $\mathbf{t}$. We write SN_0 for the set of strongly normalising closed terms of λ_{CA}.

Reducibility candidates A set of terms A is a *reducibility candidate* if it satisfies the following conditions:

(CR$_1$) *Strong normalisation*: $\mathsf{A} \subseteq \mathsf{SN}_0$

(CR$_2$) *Stability under reduction*: If $\mathbf{t} \in \mathsf{A}$ and $\mathbf{t} \to^* \mathbf{t}'$, then $\mathbf{t}' \in \mathsf{A}$.

(CR$_3$) *Stability under neutral expansion*: If $\mathbf{t}$ is neutral and $\mathrm{Red}(\mathbf{t}) \subseteq \mathsf{A}$, then $\mathbf{t} \in \mathsf{A}$.

In the sequel, A, B stand for reducibility candidates, and RC stands for the set of all reduciblity candidates.

The idea of the strong normalisation proof is to interpret types by reducibility candidates and then show that whenever a term has a type, it is in a reducibility candidate.

Remark Note that SN_0 is a reducibility candidate. In addition, the term $\mathbf{0}$ is a neutral and normal term, so it is in every reducibility candidate.

The following lemma ensures that the strong normalisation property is preserved by linear combination.

Lemma 3.3 *If* $\mathbf{t}$ *and* $\mathbf{r}$ *are strongly normalising, then* $\alpha.\mathbf{t}+\beta.\mathbf{r}$ *is strongly normalising.*

Proof Induction on a positive algebraic measure defined on terms of λ_{lin} [4], showing that every algebraic reduction makes this number strictly decrease. ∎

The following operators ensure that all types of λ_{CA} are interpreted by a reducibility candidate.

Operators in RC Let A, B be reducibility candidates. We define operators $\rightarrow$, $\oplus$, Λ over RC and $\overline{\emptyset}$ such that

- $\mathsf{A} \rightarrow \mathsf{B}$ is the closure of $\{\mathbf{t} \mid \forall \mathbf{b} \in \mathsf{A}, \mathbf{b}$ a basis term $\Rightarrow (\mathbf{t})\,\mathbf{b} \in \mathsf{B}\}$ under (CR_3),
- $\mathsf{A} \oplus \mathsf{B}$ is the closure of $\{\alpha.\mathbf{t} + \beta.\mathbf{r} \mid \mathbf{t} \in \mathsf{A}, \mathbf{r} \in \mathsf{B}\}$ under (CR_2) and (CR_3),
- $\Lambda\mathsf{A}$ is the set $\{\mathbf{t} \mid \forall V, \mathbf{t}@V \in \mathsf{A}\}$
- $\overline{\emptyset}$ is the closure of $\emptyset$ under (CR_3).

Lemma 3.4 *Let* A *and* B *be reducibility candidates. Then* $\mathsf{A} \rightarrow \mathsf{B}$, $\mathsf{A} \oplus \mathsf{B}$, $\Lambda\mathsf{A}$, $A \cap B$ *and* $\overline{\emptyset}$ *are all reducibility candidates.*

Proof We show the proof for $\overline{\emptyset}$ and $\mathsf{A} \oplus \mathsf{B}$. The rest of the cases are similar.

- The three conditions hold trivially for $\overline{\emptyset}$.
- Let $\mathbf{t} \in \mathsf{A} \oplus \mathsf{B}$. We must check that the three conditions hold.

  (CR_1) Induction on the construction of $\mathsf{A} \oplus \mathsf{B}$. If $\mathbf{t} \in \{\alpha.\mathbf{t} + \beta.\mathbf{r} \mid \mathbf{t} \in \mathsf{A}, \mathbf{r} \in \mathsf{B}\}$, the result is trivial by condition (CR_1) on A and B and lemma 3.3. If $\mathbf{t} \rightarrow^* \mathbf{t}'$ with $\mathbf{t} \in \mathsf{A} \oplus \mathsf{B}$, then $\mathbf{t}$ is strongly normalising by induction hypothesis; therefore, so is $\mathbf{t}'$. If $\mathbf{t}$ is neutral and $\text{Red}(\mathbf{t}) \subseteq \mathsf{A} \oplus \mathsf{B}$, then $\mathbf{t}$ is strongly normalising since by induction hypothesis all elements of $\text{Red}(\mathbf{t})$ are strongly normalising.

  (CR_2) and (CR_3) Trivial by construction of $\mathsf{A} \oplus \mathsf{B}$. ∎

We can now introduce the interpretation function for the types of λ_{CA}. The definition relies on the operators for reducibility candidates defined above.

A *valuation* ρ is a partial function from type variables to reducibility candidates, written as a sequence of comma-separated mappings of the form $X \mapsto \mathsf{A}$, with $\emptyset$ denoting the empty valuation.

Reducibility model Let T be a type and ρ a valuation. We define the *interpretation* $[\![T]\!]_\rho$ as follows:

$$
\begin{array}{rcl}
[\![X]\!]_\rho &=& \rho(X) \\
[\![\bar{0}]\!]_\rho &=& \overline{\emptyset} \\
[\![U \rightarrow T]\!]_\rho &=& [\![U]\!]_\rho \rightarrow [\![T]\!]_\rho \\
[\![T + R]\!]_\rho &=& [\![T]\!]_\rho \oplus [\![R]\!]_\rho \\
[\![\forall X.U]\!]_\rho &=& \bigcap_{\mathsf{S} \in \mathsf{RC}} \Lambda [\![U]\!]_{\rho, X \mapsto S}
\end{array}
$$

Note that lemma 3.4 ensures that every type is interpreted by a reducibility candidate.

A *substitution* σ is a partial function from term variables to basis terms, written as a sequence of semicolon-separated mappings of the form $x \mapsto \mathbf{b}$, with $\emptyset$ denoting the empty substitution. The action of substitutions on terms is given by

$$
\mathbf{t}_\emptyset = \mathbf{t}, \qquad \mathbf{t}_{x \mapsto \mathbf{b}; \sigma} = \mathbf{t}[\mathbf{b}/x]_\sigma
$$

A *type substitution* δ is a partial function from type variables to unit types, written as a sequence of semicolon-separated mappings of the form $X \mapsto U$, with $\emptyset$ denoting the empty substitution. The action of type substitutions on types is given by

$$T_\emptyset = T, \qquad\qquad T_{X\mapsto U;\delta} = T[U/X]_\delta$$

They are extended to act on terms in the natural way.

Let Γ be a typing context, then we say that a substitution pair $\langle\sigma,\delta\rangle$ *satisfies* Γ for a valuation ρ (written $\langle\sigma,\delta\rangle \in [\![\Gamma]\!]_\rho$) if $(x:U)\in\Gamma$ implies $x_\sigma \in [\![U_\delta]\!]_\rho$.

A typing judgement $\Gamma\vdash \mathbf{t}:T$ is said to be *valid* (written $\Gamma\vDash \mathbf{t}:T$) if for every valuation ρ, for every type substitution δ and every substitution σ such that $\langle\sigma,\delta\rangle \in [\![\Gamma]\!]_\rho$, we have $(\mathbf{t}_\delta)_\sigma \in [\![T]\!]_\rho$. The following lemma proves that every derivable typing judgement is valid.

Lemma 3.5 (Adequacy Lemma) *Let* $\Gamma\vdash\mathbf{t}:T$*, then* $\Gamma\vDash\mathbf{t}:T$.

Proof We proceed by induction on the derivation of $\Gamma\vdash\mathbf{t}:T$. The base cases (rules AX and $\text{AX}_{\vec{0}}$) are trivial. We show the cases for rules $\to_I$ and SI for illustration purposes.

- Case $\to_I$: $\dfrac{\Gamma,x:U\vdash\mathbf{t}:T}{\Gamma\vdash\lambda x:U.\mathbf{t}:U\to T}$

  By induction hypothesis, we have $\Gamma,x:U\vDash\mathbf{t}:T$. We will prove that for all ρ and $\langle\sigma,\delta\rangle\in[\![\Gamma]\!]_\rho$, $((\lambda x:U.\mathbf{t})_\delta)_\sigma\in[\![U\to T]\!]_\rho$. Suppose that $\sigma=(x\mapsto\mathbf{v};\sigma'')\in[\![\Gamma,x:U]\!]_\rho$. Let $\mathbf{b}\in[\![U]\!]_\rho$ (note that there is at least one basis term, $\mathbf{v}$, in $[\![U]\!]_\rho$), and let $\sigma'=(x\mapsto\mathbf{b};\sigma'')$. So $\sigma'\in[\![\Gamma,x:U]\!]_\rho$, hence $(\mathbf{t}_\delta)_{\sigma'}\in[\![T]\!]_\rho$. This means both $\mathbf{b}$ and $(\mathbf{t}_\delta)_{\sigma'}$ are strongly normalising, so we shall first prove that all reducts of $((\lambda x:U.\mathbf{t})_\delta)_\sigma\,\mathbf{b}$ are in $[\![T]\!]_\rho$.

  - $((\lambda x:U.\mathbf{t})_\delta)_\sigma\,\mathbf{b}\to(\lambda x:U.\mathbf{t}')\,\mathbf{b}$ or $((\lambda x:U.\mathbf{t})_\delta)_\sigma\,\mathbf{b}\to((\lambda x:U.\mathbf{t})_\delta)_\sigma\,\mathbf{b}'$, with $(\mathbf{t}_\delta)_\sigma\to\mathbf{t}'$ or $\mathbf{b}\to\mathbf{b}'$. The result follows by induction on the reductions of $\mathbf{b}$ and $(\mathbf{t}_\delta)_{\sigma'}$, respectively: by induction hypothesis we have $\mathbf{t}',\mathbf{b}'\in[\![T]\!]_\rho$, so both $(\lambda x:U.\mathbf{t}')\,\mathbf{b}$, $(((\lambda x:U.\mathbf{t})_\delta)_\sigma)\,\mathbf{b}'\in[\![T]\!]_\rho$.
  - $((\lambda x:U.\mathbf{t})_\delta)_\sigma\,\mathbf{b}\to(\mathbf{t}_\delta)_\sigma[\mathbf{b}/x]=(\mathbf{t}_\delta)_{\sigma'}\in[\![T]\!]_\rho$.

  Therefore, $((\lambda x:U.\mathbf{t})_\delta)_\sigma\,\mathbf{b}$ is a neutral term with all of its reducts in $[\![T]\!]_\rho$, so $((\lambda x:U.\mathbf{t})_\delta)_\sigma\,\mathbf{b}\in[\![T]\!]_\rho$. Hence, by definition of $\to$, we conclude $((\lambda x:U.\mathbf{t})_\delta)_\sigma\in[\![U\to T]\!]_\rho$.

- Case SI: $\dfrac{\Gamma\vdash\mathbf{t}:T}{\Gamma\vdash\alpha.\mathbf{t}:\lfloor\alpha\rfloor.T}$

  By induction hypothesis, we have $\Gamma\vDash\mathbf{t}:T$. Let ρ be a valuation and $\langle\sigma,\delta\rangle$ a substitution pair satisfying Γ in ρ. So $(\mathbf{t}_\delta)_\sigma\in[\![T]\!]_\rho$, hence $\alpha.(\mathbf{t}_\delta)_\sigma\in\bigoplus_{i=1}^{\lfloor\alpha\rfloor}[\![T]\!]_\rho=[\![\lfloor\alpha\rfloor.T]\!]_\rho$ by construction. ∎

Since this proves that every well-typed term is in a reducibility candidate, we can easily show that such terms are strongly normalising.

Theorem 3.2 (Strong Normalisation for λ_{CA}) *All typeable terms of λ_{CA} are strongly normalising.*

Proof Let $\mathbf{t}$ be a term of λ_{CA} of type T. Then, by the Adequacy Lemma (lemma 3.5), we know that $(\mathbf{t}_\emptyset)_\emptyset\in[\![T]\!]_\emptyset$. Furthermore, by lemma 3.4, we know $[\![T]\!]_\emptyset$ is a reducibility candidate, and therefore $[\![T]\!]_\emptyset\subseteq\mathsf{SN}_0$. Hence, $\mathbf{t}$ is strongly normalising. ∎

3.2.1 Confluence

Now confluence follows as a corollary of the strong normalisation theorem.

Corollary 3.3 (Confluence) *The typed language λ_{CA} is confluent: for any term* $\mathbf{t}$*, if* $\mathbf{t} \to^* \mathbf{r}$ *and* $\mathbf{t} \to^* \mathbf{u}$*, then there exists a term* $\mathbf{t}'$ *such that* $\mathbf{r} \to^* \mathbf{t}'$ *and* $\mathbf{u} \to^* \mathbf{t}'$*.*

Proof The proof of the local confluence of the system, *i.e.* the property saying that $\mathbf{t} \to \mathbf{r}$ and $\mathbf{t} \to \mathbf{u}$ imply that there exists a term $\mathbf{t}'$ such that $\mathbf{r} \to^* \mathbf{t}'$ and $\mathbf{u} \to^* \mathbf{t}'$, is an extension of the one presented for the untyped calculus in [3], where the set of algebraic rules (*i.e.* all rules but the beta reductions) have been proved to be locally confluent using the proof assistant Coq. Then, a straightfoward induction entails the (local) commutation between the algebraic rules and the β-reductions. Finally, the confluence of the β-reductions is a trivial extension of the proof for λ-calculus. Local confluence plus strong normalisation (*cf.* Theorem 3.2) implies confluence [18]. ∎

4 Abstract Interpretation

The type system of λ_{CA} approximates the more precise types that are obtained under reduction. The approximation suggests that a λ-calculus without scalars can be seen as an abstract interpretation of λ_{CA}: its terms can approximate the terms of λ_{CA}. Scalars can be approximated to their floor, and hence be represented by sums, just as the types in λ_{CA} do. This intuition is formalised in this section, using *Additive*, the calculus presented in [7]. This calculus is a typed version of the additive fragment of λ_{lin} [4], which in turn is the untyped version of λ_{CA}.

The *Additive* calculus is shown in Fig. 5. It features strong normalisation, subject reduction and confluence. For details on those proofs, please refer to [7]. The types and equivalences coincide with those from λ_{CA}. We write the types explicitly in the terms to match the presentation of λ_{CA}, although the original presentation is in Curry style. We use $\vdash_A$ to distinguish the judgements in λ_{CA} ($\vdash$) from the judgements in *Additive*. Also, we write the reductions in *Additive* as $\to_A$, $\mathbf{t}{\downarrow_A}$ for the normal form of the term $\mathbf{t}$ in *Additive* and $\mathbf{t}{\downarrow}$ for the normal form of $\mathbf{t}$ in λ_{CA}.

Let T_c be the set of terms in the calculus c. Consider the following *abstraction* function $\sigma : T_{\lambda_{CA}} \to T_{\lambda^{\text{add}}}$ from terms in λ_{CA} to terms in *Additive*:

$$\begin{array}{rclrcl}
\sigma(x:U) & = & x:U & \sigma(\mathbf{t}@U) & = & \sigma(\mathbf{t})@U \\
\sigma(\lambda x{:}\,U.\mathbf{t}) & = & \lambda x{:}\,U.\sigma(\mathbf{t}) & \sigma(\mathbf{0}) & = & \mathbf{0} \\
\sigma(\Lambda X.\mathbf{t}) & = & \Lambda X.\sigma(\mathbf{t}) & \sigma(\alpha.\mathbf{t}) & = & \sum_{i=1}^{\lfloor\alpha\rfloor}\sigma(\mathbf{t}) \\
\sigma((\mathbf{t})\,\mathbf{t}') & = & (\sigma(\mathbf{t}))\,\sigma(\mathbf{t}') & \sigma(\mathbf{t}+\mathbf{t}') & = & \sigma(\mathbf{t})+\sigma(\mathbf{t}')
\end{array}$$

where for any term $\mathbf{t}$, $\sum_{i=1}^{0}\mathbf{t} = \mathbf{0}$.

We can also define a *concretisation* function $\gamma : T_{\lambda^{\text{add}}} \to T_{\lambda_{CA}}$, which is the obvious embedding of terms: $\gamma(\mathbf{t}) = \mathbf{t}$.

Let $\sqsubseteq\ \subseteq\ T_{\lambda^{\text{add}}} \times T_{\lambda^{\text{add}}}$ be the least relation satisfying:

$$\begin{array}{c}
\alpha \leq \beta \ \Rightarrow\ \sum_{i=1}^{\alpha}\mathbf{t} \sqsubseteq \sum_{i=1}^{\beta}\mathbf{t} \\
\begin{array}{ll}
\mathbf{t} \sqsubseteq \mathbf{t}' \ \Rightarrow\ \lambda x{:}\,U.\mathbf{t} \sqsubseteq \lambda x{:}\,U.\mathbf{t}' & \mathbf{t} \sqsubseteq \mathbf{t}' \wedge \mathbf{r} \sqsubseteq \mathbf{r}' \ \Rightarrow\ (\mathbf{t})\,\mathbf{r} \sqsubseteq (\mathbf{t}')\,\mathbf{r}' \\
\mathbf{t} \sqsubseteq \mathbf{t}' \ \Rightarrow\ \Lambda X.\mathbf{t} \sqsubseteq \Lambda X.\mathbf{t}' & \mathbf{t} \sqsubseteq \mathbf{t}' \wedge \mathbf{r} \sqsubseteq \mathbf{r}' \ \Rightarrow\ \mathbf{t}+\mathbf{r} \sqsubseteq \mathbf{t}'+\mathbf{r}' \\
\mathbf{t} \sqsubseteq \mathbf{t}' \ \Rightarrow\ \mathbf{t}@U \sqsubseteq \mathbf{t}'@U & \mathbf{t} \sqsubseteq \mathbf{r} \wedge \mathbf{r} \sqsubseteq \mathbf{s} \ \Rightarrow\ \mathbf{t} \sqsubseteq \mathbf{s}
\end{array}
\end{array}$$

and let $\lesssim$ be the relation defined by $\mathbf{t}_1 \lesssim \mathbf{t}_2 \Leftrightarrow \mathbf{t}_1{\downarrow_A} \sqsubseteq \mathbf{t}_2{\downarrow_A}$.

$$\begin{array}{lrcl} \textit{Terms:} & \mathbf{t},\mathbf{r},\mathbf{s} & ::= & \mathbf{b} \mid (\mathbf{t})\ \mathbf{r} \mid \mathbf{t}@U \mid \mathbf{0} \mid \mathbf{t}+\mathbf{r} \\ \textit{Basis terms:} & \mathbf{b} & ::= & x \mid \lambda x:U.\mathbf{t} \mid \Lambda X.\mathbf{t} \end{array}$$

Group A:
$(\mathbf{u}+\mathbf{t})\ \mathbf{r} \to_{A} (\mathbf{u})\ \mathbf{r} + (\mathbf{t})\ \mathbf{r}$
$(\mathbf{r})\ (\mathbf{u}+\mathbf{t}) \to_{A} (\mathbf{r})\ \mathbf{u} + (\mathbf{r})\ \mathbf{t}$
$(\mathbf{0})\ \mathbf{t} \to_{A} \mathbf{0}$
$(\mathbf{t})\ \mathbf{0} \to_{A} \mathbf{0}$

Group E:
$\mathbf{t}+\mathbf{0} \to_{A} \mathbf{t}$

β-reduction:
$(\lambda x:U.\mathbf{t})\ \mathbf{b} \to_{A} \mathbf{t}[\mathbf{b}/x]$
$(\Lambda X.\mathbf{t})@U \to_{A} \mathbf{t}[U/X]$

$$\frac{}{\Gamma, x:U \vdash_A x:U}\ ax \qquad \frac{}{\Gamma \vdash_A \mathbf{0}:\bar{0}}\ ax_{\bar{0}} \qquad \frac{\Gamma \vdash_A \mathbf{t}:\sum_{i=1}^{\alpha}(U \to T_i) \qquad \Gamma \vdash_A \mathbf{r}:\beta.U}{\Gamma \vdash_A (\mathbf{t})\ \mathbf{r}:\sum_{i=1}^{\alpha}(\beta.T_i)}\ \to_E$$

$$\frac{\Gamma, x:U \vdash_A \mathbf{t}:T}{\Gamma \vdash_A \lambda x.\mathbf{t}:U \to T}\ \to_I \qquad \frac{\Gamma \vdash_A \mathbf{t}:\forall X.U}{\Gamma \vdash_A \mathbf{t}@V:U[V/X]}\ \forall_E \qquad \frac{\Gamma \vdash_A \mathbf{t}:U \qquad X \notin FV(\Gamma)}{\Gamma \vdash_A \Lambda X.\mathbf{t}:\forall X.U}\ \forall_I$$

$$\frac{\Gamma \vdash_A \mathbf{t}:T \qquad \Gamma \vdash_A \mathbf{r}:R}{\Gamma \vdash_A \mathbf{t}+\mathbf{r}:T+R}\ +_I \qquad \frac{\Gamma \vdash_A \mathbf{t}:T \qquad T \equiv R}{\Gamma \vdash_A \mathbf{t}:R}\ \equiv$$

Figure 5: The *Additive* calculus. Type syntax and equivalence coincide with those from λ_{CA}.

The relation $\sqsubseteq$ is a partial order. Also, $\lesssim$ is a partial order if we quotient terms by the relation $\sim$, defined by $\mathbf{t} \sim \mathbf{r}$ if and only if $\mathbf{t}{\downarrow} = \mathbf{r}{\downarrow}$. We formalise this in the following lemma.

Lemma 4.1

1. $\sqsubseteq$ *is a partial order relation*
2. $\lesssim$ *is a partial order relation in* $T_{\lambda^{add}}/_{\sim}$. ∎

The following theorem states that the terms in λ_{CA} can be seen as a refinement of those in *Additive*, *i.e.* we can consider *Additive* as an abstract interpretation of λ_{CA}. It follows by a nontrivial structural induction on $\mathbf{t} \in T_{\lambda_{CA}}$.

Theorem 4.1 (Abstract interpretation) *The function* $\downarrow$ *is a valid concretisation of the function* $\downarrow_{A}$*:* $\forall \mathbf{t} \in T_{\lambda_{CA}}$, $\sigma(\mathbf{t}){\downarrow_{A}} \lesssim \sigma(\mathbf{t}{\downarrow})$. ∎

The following lemma states that the abstraction preserves the typings.

Lemma 4.2 *For arbitrary context* Γ*, term* $\mathbf{t}$ *and type* T*, if* $\Gamma \vdash \mathbf{t}:T$ *then* $\Gamma \vdash_A \sigma(\mathbf{t}):T$. ∎

Taking *Additive* as an abstract interpretation of λ_{CA} entails the extension of the interpretation of *Additive* into System F with pairs, F_p (*cf.* [7]) as an abstract interpretation of λ_{CA}, as depicted in Fig. 6. The complete language F_p is defined in Fig. 7. We denote by $t{\downarrow_{F}}$ the normal form of a term t in F_p. The relation $\lessapprox$ is a straightforward translation of the relation $\lesssim$ into a relation in F_p. The function $[\cdot]_{\mathsf{D}}$ is the translation from typed terms in *Additive* into terms in F_p; this translation depends on the typing derivation D of the term in *Additive* (*cf.* [7] for more details). We formalise this in Theorem 4.2 and also give the formal definition of the relation $\lessapprox$ in definition 4.

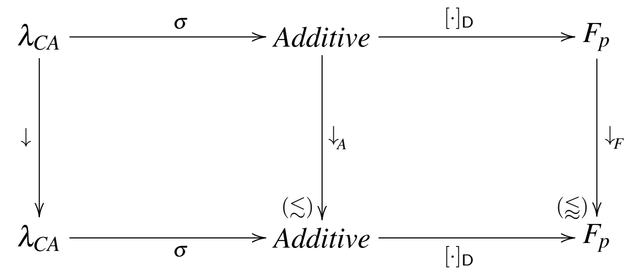

Figure 6: Abstract interpretation of λ_{CA} into System F with pairs

Terms: $t,u ::= x \mid \lambda x.t \mid tu \mid \star \mid \langle t,u\rangle \mid \pi_1(t) \mid \pi_2(t)$

Types: $A,B ::= X \mid A \to_F B \mid \forall X.A \mid \mathbf{1} \mid A \times B$

$$(\lambda x.t)u \to_F t[u/x] \qquad ; \qquad \pi_i\langle t_1,t_2\rangle \to_F t_i$$

$$\frac{}{\Delta, x:A \vdash_F x:A}Ax \qquad \frac{}{\Delta \vdash_F \star : \mathbf{1}}\mathbf{1} \qquad \frac{\Delta, x:A \vdash_F t:B}{\Delta \vdash_F \lambda x.t : A \to_F B}\to_F I \qquad \frac{\Delta \vdash_F t : A \to_F B \qquad \Delta \vdash_F u:A}{\Delta \vdash_F tu : B}\to_F E$$

$$\frac{\Delta \vdash_F t:A \qquad \Delta \vdash_F u:B}{\Delta \vdash_F \langle t,u\rangle : A \times B}\times I \qquad \frac{\Delta \vdash_F t : A \times B}{\Delta \vdash_F \pi_1(t):A}\times E_\ell \qquad \frac{\Delta \vdash_F t : A \times B}{\Delta \vdash_F \pi_2(t):B}\times E_r$$

$$\frac{\Delta \vdash_F t:A \qquad X \notin FV(\Delta)}{\Delta \vdash_F t : \forall X.A}\forall I \qquad \frac{\Delta \vdash_F t : \forall X.A}{\Delta \vdash_F t : A[B/X]}\forall E$$

Figure 7: System F with pairs

Definition Let $\sqsubseteq_F \subseteq T_{F_p} \times T_{F_p}$ be the least relation between terms of F_p satisfying:

$$\begin{array}{lll}
\star \sqsubseteq_F t & t \sqsubseteq_F t & t \sqsubseteq_F (t,t) \\
t \sqsubseteq_F t' \wedge r \sqsubseteq_F r' \Rightarrow (t,r) \sqsubseteq_F (t',r') & & t \sqsubseteq_F t' \Rightarrow \lambda x.t \sqsubseteq_F \lambda x.t' \\
t \sqsubseteq_F t' \wedge r \sqsubseteq_F r' \Rightarrow t\ r \sqsubseteq_F t'\ r' & & t \sqsubseteq_F t' \Rightarrow \pi_1(t) \sqsubseteq_F \pi_1(t') \\
t \sqsubseteq_F r \wedge r \sqsubseteq_F s \Rightarrow t \sqsubseteq_F s & & t \sqsubseteq_F t' \Rightarrow \pi_2(t) \sqsubseteq_F \pi_2(t')
\end{array}$$

and let $\lessapprox$ be the relation defined by $t_1 \lessapprox t_2 \Leftrightarrow t_1{\downarrow_A} \sqsubseteq_F t_2{\downarrow_A}$.

The relation $\sqsubseteq_F$ is a partial order. Moreover $\lessapprox$ is a partial order if we quotient terms in F_p by the equivalence relation $\approx$, defined as: $t \approx r$ if and only if $t{\downarrow_F} = r{\downarrow_F}$.

Lemma 4.3

1. *$\sqsubseteq_F$ is a partial order relation.*
2. *$\lessapprox$ is a partial order relation over $T_{F_p}/_\approx$.* ∎

In [7, Thm. 3.8] it is shown that the translation $[\cdot]_{\mathsf{D}}$ is well behaved. So it will trivially keep the order.

Lemma 4.4 *Let D be a derivation tree ending in $\Gamma \vdash_A \mathbf{t} : T$ and D' be a derivation tree corresponding to $\Gamma \vdash_A \mathbf{r} : R$, where $\mathbf{t} \lesssim \mathbf{r}$. Then $[\mathbf{t}]_{\mathsf{D}} \lessapprox [\mathbf{r}]_{\mathsf{D}'}$.* ∎

Theorem 4.2 *The function* $\downarrow$ *is a valid concretisation of* $\downarrow_F$*:* $\forall \mathbf{t} \in T_{\lambda_{CA}}$ *if* D *is a derivation of* $\Gamma \vdash \sigma(\mathbf{t}) : T$ *and* D' *is the derivation of* $\Gamma \vdash \sigma(\mathbf{t}\downarrow) : T'$*, then* $[\sigma(\mathbf{t})]_{\mathsf{D}}\downarrow_F \lessapprox [\sigma(\mathbf{t}\downarrow)]_{\mathsf{D}'}$.

Proof Theorem 4.1 states that the left square in Fig. 6 commutes, lemma 4.2 states that the typing is preserved by this translation, and finally lemma 4.4 states that the square on the right commutes. ∎

5 Summary of Contributions

We have presented a confluent, typed, strongly normalising, algebraic λ-calculus, based on λ_{lin}, which has an algebraic rewrite system without restrictions. Typing guarantees confluence, thereby allowing us to simplify the rewrite rules for the system with respect to λ_{lin}. Moreover, λ_{CA} differs from λ_{alg} in that it presents vectors in a canonical form by using a rewrite system instead of an equational theory.

In this work, scalars are approximated by natural numbers. This approximation yields a subject reduction property which is exact about the types involved in a term, but only approximate in their "amount" or "weight". In addition, the approximation is a lower bound: if a term has a type that is a sum of some amount of different types, then after reducing it these amounts can be incremented but never decremented.

One of the original motivations for this work was to ensure confluence in the presence of algebraic rewrite rules, while remaining "classic", in the sense that the type system does not introduce uninterpretable elements such as scalars. To prove that we have achieved this goal, we have shown that terms in *Additive*, the additive fragment of λ_{lin}, can be seen as an abstract interpretation of terms in λ_{CA}, and then System F can also be used as an abstract interpretation of terms in λ_{CA} by the translation from *Additive* into F_p.

In our calculus, we have chosen to take the floor of the scalars to approximate types. However, this decision is arbitrary, and we could have chosen to approximate types using the ceiling instead. Therefore, an obvious extension of this system is to take both floor and ceiling of scalars to produce type intervals, thus obtaining more accurate approximations.

Since this paper is meant as a "proof of concept" we have not worked around a known restriction in *Additive*, which allows sums as arguments only when all their constituent terms have the same type. However, it has been proved that this can be solved by using a more sophisticated arrow elimination typing rule [3].

Since the type system derives from System F, there are some total functions which cannot be represented in λ_{CA}, even though they are expressible in λ_{lin}. This is not a problem in practice because these functions are quite hard to find, so it is a small price to pay for having a simpler, confluent rewrite system.

It is still an open question how to obtain a similar result for a calculus where scalars are members of an arbitrary ring.

References

[1] Pablo Arrighi & Alejandro Díaz-Caro (2011): *Scalar System F for Linear-Algebraic λ-Calculus: Towards a Quantum Physical Logic*. In Peter Selinger, editor: *Proceedings of QPL-2009*, *Electronic Notes in Theoretical Computer Science* 270/1, Elsevier, pp. 219–229. (Extended version at `arXiv:0903.4741`).

[2] Pablo Arrighi, Alejandro Díaz-Caro & Benoît Valiron (2011): *Subject reduction in a Curry-style polymorphic type system with a vectorial space structure*. In: *QAPL-2011*, Saarbrücken, Germany. (presentation report).

[3] Pablo Arrighi, Alejandro Díaz-Caro & Benoît Valiron (2011): *A type system for the vectorial aspects of the linear-algebraic lambda-calculus*. `arXiv:1012.4032`.

[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.

[5] Henk P. Barendregt (1992): *Lambda calculi with types*. *Handbook of Logic in Computer Science* 2, Clarendon Press, Oxford, UK.

[6] Alejandro Díaz-Caro, Simon Perdrix, Christine Tasson & Benoît Valiron (2010): *Equivalence of Algebraic λ-calculi*. In: *Informal proceedings of HOR-2010*, Edinburgh, UK, pp. 6–11. `arXiv:1005.2897`.

[7] Alejandro Díaz-Caro & Barbara Petit (2010): *Sums in linear algebraic lambda-calculus*. `arXiv:1011.3542`.

[8] Thomas Ehrhard (2003): *On Köthe sequence spaces and linear logic*. *Mathematical Structures in Computer Science* 12(5), pp. 579–623.

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

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

[11] Thomas Ehrhard & Laurent Regnier (2003): *The differential lambda-calculus*. *Theoretical Computer Science* 309(1), pp. 1–41.

[12] Jean-Yves Girard, Yves Lafont & Paul Taylor (1989): *Proofs and Types*. *Cambridge Tracts in Theoretical Computer Science* 7, Cambridge University Press, Cambridge, UK.

[13] Jean-Louis Krivine (1990): *Lambda-calcul: types et modèles*. Études et recherches en informatique, Masson, Paris, France.

[14] Michele Pagani & Simona Ronchi Della Rocca (2010): *Solvability in Resource Lambda Calculus*. In Luke Ong, editor: *Proceedings of FOSSACS-2010*, *Lecture Notes in Computer Science* 6014, Springer, pp. 358–373.

[15] Michele Pagani & Paolo Tranquilli (2009): *Parallel Reduction in Resource Lambda-Calculus*. In Zhenjiang Hu, editor: *Proceedings of APLAS-2009*, *Lecture Notes in Computer Science* 5904, Springer, pp. 226–242.

[16] John C. Reynolds (1974): *Towards a theory of type structure*. In B. Robinet, editor: *Proceedings of the Colloque sur la Programmation*, *Lecture Notes in Computer Science* 19, Springer, pp. 408–425.

[17] Christine Tasson (2009): *Algebraic Totality, towards Completeness*. In Pierre-Louis Curien, editor: *Proceedings of TLCA-2009*, *Lecture Notes in Computer Science* 5608, Springer, pp. 325–340.

[18] TeReSe (2003): *Term Rewriting Systems*. *Cambridge Tracts in Theoretical Computer Science* 55, Cambridge University Press.

[19] Lionel Vaux (2007): *On Linear Combinations of Lambda-Terms*. In Franz Baader, editor: *Proceedings of RTA-07*, *Lecture Notes in Computer Science* 4533, Springer, pp. 374–388.

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