

[Electronic Notes in Theoretical Computer Science 265 (2010) 213–230](http://dx.doi.org/10.1016/j.entcs.2010.08.013)

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Categorical Models for Simply Typed Resource Calculi

Antonio Bucciarelli[3](#_bookmark0) Thomas Ehrhard[4](#_bookmark0)*,*[1](#_bookmark0)

*Laboratoire PPS, Universit´e Paris-Diderot, CNRS UMR 7126 2, place Jussieu (case 7014), 75251 Paris Cedex 05, France*

Giulio Manzonetto[5](#_bookmark0)*,*[2](#_bookmark0)

*Laboratoire LIPN, Universit´e Paris Nord, CNRS UMR 7030 99, avenue Jean-Baptiste Cl´ement 93430 Villetaneuse, France*

**Abstract**

We introduce the notion of *differential λ-category* as an extension of Blute-Cockett-Seely’s differential Cartesian categories. We prove that differential *λ*-categories can be used to model the simply typed versions of: (i) the *differential λ-calculus*,a *λ*-calculus extended with a syntactic derivative operator; (ii) the *resource calculus*, a non-lazy axiomatisation of Boudol’s *λ*-calculus with multiplicities. Finally, we provide two concrete examples of differential *λ*-categories, namely, the category **MRel** of sets and relations, and the category **MFin** of finiteness spaces and finitary relations.

*Keywords:* differential *λ*-calculus, resource calculus, differential categories, relational models.

# Introduction

The development of formal systems for proving computational properties of pro- grams constitutes a crucial research area of contemporary computer science. Among the vast spectrum of aspects needing to be checked, one of the most important is the amount of resources a program will need during its execution. Resources to be bounded can be of very different kinds, for instance *memory space* (especially in

1 This work is partly funded by the ANR project BLAN07-1 189926 *Curry-Howard for Concurrency*

(CHOCO).

2 This work is partly funded by Digiteo/ˆIle-de-France project COLLODI (2009-28HD), by MIUR project CONCERTO and by NWO project CALMOC (612.000.936).

3 Email: [Antonio.Bucciarelli@pps.jussieu.fr](mailto:Antonio.Bucciarelli@pps.jussieu.fr)

4 Email: [Thomas.Ehrhard@pps.jussieu.fr](mailto:Thomas.Ehrhard@pps.jussieu.fr)

5 Email: [Giulio.Manzonetto@lipn.univ-paris13.fr](mailto:Giulio.Manzonetto@lipn.univ-paris13.fr)

1571-0661 © 2010 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

doi:10.1016/j.entcs.2010.08.013

presence of very small computing devices) or *non-replicable data* (naturally arising in the context of quantum computing).

In [[4](#_bookmark23)] Boudol designed the *λ-calculus with multiplicities*, a paradigmatic pro-

gramming language developed for handling explicitly the problem of resource con- sumption within the *λ*-calculus. In this calculus two kinds of arguments are avail- able: intuitionistic arguments that can be erased and copied as usual, and depletable arguments that must be used exactly once. Depletable arguments impose the pres- ence of non-deterministic choices in the language. Suppose indeed that we have (*λx.xx*)*L* where *L* is a depletable argument: what occurrence of *x* should receive the only available copy of *L*? Another novelty with respect to *λ*-calculus is the fact that the arguments come in multisets called ‘bags’.

The original version of *λ*-calculus with multiplicities was only endowed with a (weak) head-reduction rule. This is crucial if we want to actually perform the non- deterministic choices during the reduction. In [[13](#_bookmark34)] Tranquilli defined the *resource*

*calculus* which is a revisitation of Boudol’s calculus having more general forms

of reductions that have been studied in [[12](#_bookmark33)]. In this calculus non-determinism is accounted for by means of formal sums of terms.

A formal system modeling this idea of ‘resource consumption’ was already present in Girard’s quantitative semantics [[11](#_bookmark32)]. This semantics establishes an anal- ogy between linearity in the sense of computer science (programs using arguments exactly once) and algebraic linearity (commutation of sums and products with scalars), giving a new mathematically very appealing interpretation of resource consumption. Drawing on these insights, Ehrhard and Regnier designed a formal

programming language, called the *differential λ-calculus* [[6](#_bookmark24)], that has a unique kind

of arguments but two kinds of applications: the usual one, and a *linear application*. The result of applying linearly *λx.M* to *L* is the term *λx.M* where *L* is substituted for *x* in *M* exactly once (we keep the *λx* because one day we may want to substi- tute the other occurrences of *x* in *M* ). The breakthrough in [[6](#_bookmark24)] is the fact that this ‘linear’ substitution operation can be seen as a formal derivative. Such a syntactic derivative operator can be fruitfully used to increase control over programs executed in environments with bounded resources (see, *e.g.*, [[8](#_bookmark29),[10](#_bookmark31)]).

Although the differential *λ*-calculus is born from semantical considerations (*i.e.*, the deep analysis of denotational semantics of linear logic performed by Ehrhard and Regnier) the investigations on its denotational semantics are at the very beginning. On the one hand, it is known *in the folklore* that the Cartesian closed category (ccc, for short) of finiteness spaces and finitary relations [[7](#_bookmark25)] and the ccc of sets and (multi-)relations [[5](#_bookmark26)] are models of the simply typed differential *λ*-calculus, but no

abstract definition of model has been provided. On the other hand, Blute, Cockett and Seely - inspired by the works on differential *λ*-calculus - defined the *differential categories* [[2](#_bookmark27)] and the *differential Cartesian categories* [[3](#_bookmark28)]. In these categories a derivative operator *D*(*−*) on morphisms is equationally axiomatized. The authors have then proved that these categories are sound and complete to model suitable term calculi. However, it turns out that the properties of differential categories are too weak for modeling the full differential *λ*-calculus.

The aim of the present paper is to provide an abstract model theory (based on differential Cartesian categories) for the simply typed differential *λ*-calculus and resource calculus: a unifying categorical approach having all known semantics of these calculi as instances. We are confident that our work in this domain will open the way to find mathematical tools for studying quantitative properties of programs. The main object of our studies will be the differential *λ*-calculus, but we will draw conclusions also for the resource calculus.

In Section [3](#_bookmark1) we provide the formal definition of the differential *λ*-calculus, we define the type system characterizing its simply typed version and we recall some basic properties of the language.

In Section [4](#_bookmark4), starting from the work of Blute *et Al.* [[3](#_bookmark28)], we introduce the notion of

*differential λ-category*. Basically, differential *λ*-categories are Cartesian differential categories which are Cartesian closed and satisfy a natural condition guaranteeing that the differential operator *D*(*−*) behaves well with the Cartesian closed struc- ture. We then prove that every differential *λ*-category constitutes a model of the simply typed differential *λ*-calculus: we first define the interpretation of (typing judgements of) differential *λ*-terms in the category, and secondly we prove that such an interpretation is *sound*.

In Section [5](#_bookmark13) we show that - as expected - the category **MFin** of finiteness spaces and finitary relations and the category **MRel** of sets and relations are instances of differential *λ*-categories.

Finally, Section [6](#_bookmark15) is devoted to recall the syntax of the simply typed resource calculus and to prove that differential *λ*-categories can be used to model this cal- culus. This is done by defining a translation map from the resource calculus to the differential *λ*-calculus and proving that this translation is ‘faithful’. We then define the interpretation of a (typing judgement of a) resource term in a differential *λ*-category as the interpretation of its translation.

The investigations in this paper fit in a more ambitious program whose research lines are discussed in Section [7](#_bookmark21).

# Preliminaries

To keep this article self-contained we summarize some definitions and results that will be used in the sequel. Our main reference for category theory is [[1](#_bookmark22)].

* 1. *Sets and Multisets*

Let *S* be a set. We denote by *P*(*S*) the powerset of *S*. A *multiset m* over *S* can be defined as an unordered list *m* = [*a*1*, a*2*,.. .*] with repetitions such that *ai ∈ S* for all *i*. A multiset *m* is called *ﬁnite* if it is a finite list, we denote by [] the empty multiset. If *m* is a multiset over *S*, then its *support* supp(*S*) is the set of elements of *S* occurring in *m*. Given two multisets *m*1 = [*a*1*, a*2*,.. .*] and *m*2 = [*b*1*, b*2*,.. .*] the *multi-union* of *m*1*, m*2 is defined by *m*1 *m*2 = [*a*1*, b*1*, a*2*, b*2*,.. .*]. We will write *Mf* (*S*) for the set of all finite multisets over *S*. Moreover, given a set *U ⊆ Mf* (*S*) we set supp(*U* ) = *∪m∈U* supp(*m*).

* 1. *Cartesian (Closed) Categories*

Let **C** be a small *Cartesian category* and *A, B, C* be arbitrary objects of **C**. We write **C**(*A, B*) for the homset of morphisms from *A* to *B*; when there is no chance of confusion we will write *f* : *A → B* instead of *f ∈* **C**(*A, B*). We usually denote by *A × B* the *product* of *A* and *B*, by *π*1 : *A × B → A*, *π*2 : *A × B → B* the associated *projections* and, given a pair of arrows *f* : *C → A* and *g* : *C → B*, by

*⟨f, g⟩* : *C → A × B* the unique arrow such that *π*1 *◦ ⟨f, g⟩* = *f* and *π*2 *◦ ⟨f, g⟩* = *g*.

If **C** is a Cartesian closed category (ccc, for short) we write *A ⇒ B* for the *exponential object* and ev*AB* : (*A ⇒ B*) *× A → B* for the *evaluation morphism*. Moreover, for any object *C* and arrow *f* : *C × A → B*, Λ(*f* ) : *C → A⇒B* stands for the (unique) morphism such that ev*AB ◦* (Λ(*f* ) *×* Id*A*) = *f* . Finally, denotes the terminal object and !*A* the only morphism in **C**(*A,* ).

We recall that in every ccc the following equalities hold:

|  |  |  |  |
| --- | --- | --- | --- |
| (pair) | *⟨f, g⟩◦h* = *⟨f ◦h, g ◦h⟩* | Λ(*f* ) *◦g* = Λ(*f ◦* (*g ×* Id)) | (Curry) |
| (beta-cat) | ev *◦ ⟨*Λ(*f* )*, g⟩* = *f ◦ ⟨*Id*, g⟩* | Λ(ev) = Id | (Id-Curry) |

Moreover, we can define Λ*−* = ev*◦*(*−×*Id). From (beta-cat), (Curry) and (Id-Curry) it follows that Λ(Λ*−*(*f* )) = *f* and Λ*−*(Λ(*g*)) = *g*.

# The Simply Typed Differential *λ*-calculus

In this section we recall the definition of the simply typed version of differential *λ*- calculus [[6](#_bookmark24)], together with some standard properties of the language. The set Λ*d* of *differential λ-terms* and the set Λ*s* of *simple terms* are defined by mutual induction as follows:

Λ*d* : *S, T, U, V* ::= 0 *| s | s* + *T* Λ*s* : *s, t, u, v* ::= *x | λx.s | sT |* D *s· t*

The term D *s · t* is the *linear application* of *s* to *t*. Intuitively, this means that *s* is provided with exactly one copy of *t*.

We consider differential *λ*-terms up to *α*-conversion, and up to associativity and commutativity of the sum. The term 0 is the neutral element of the sum, thus we add *S* +0 = *S*. We write *S ≡ T* if *S* and *T* are syntactically equal up to the above mentioned equivalences. The set FV(*S*) of *free variables* of *S* is defined as usual.

**Notation 1** *We will often use the following abbreviations: λx.*(Σ*k*

*i*=1

Σ

*i*=1

*si*) *for*

*k i*=1

*λx.si,* D(Σ*k*

*si*) *·* (Σ*n*

*tj*) *for* Σ

*i,j*

D *si · tj and* (Σ*k*

*si*)*t for* Σ*k*

*sit.*

*Notice that these are just syntactic sugar, not real terms.*

*i*=1

*j*=1

*i*=1

We introduce two kinds of substitutions on differential *λ*-terms: (i) the *capture- free substitution*, denoted by *S*[*T/x*] and defined as usual; (ii) the *differential sub-*

*stitution*, denoted by *∂S · T* and defined by induction on *S* as follows [6](#_bookmark3) :

*∂x*

*∂y · T* = ,⎨ *T* if *x* = *y*

*∂x*

*∂* (*sU* ) *· T* = ( *∂s · T* )*U* + (D *s ·* ( *∂U · T* ))*U*

*∂x*

*∂x*

*∂x*

, 0 otherwise

*∂* (*λy.s*) *· T* = *λy. ∂s · T* *∂* (D *s · u*) *· T* = D( *∂s · T* ) *· u* + D *s ·* ( *∂u · T* )

*∂x ∂x ∂x ∂x ∂x*

*∂*0 *· T* = 0 *∂* (*s* + *U* ) *· T* = *∂s · T* + *∂U · T*

*∂x ∂x ∂x ∂x*

The *differential λ-calculus* is generated by the *β*-reduction (*λx.s*)*T →β s*[*T/x*] and

the linear reduction D(*λx.s*) *· t →βD λx. · t*. We write *→D* for the contextual

*∂s*

*∂x*

closure of *→β ∪ →βD* , and →*D* (resp. =*D*) for the transitive and reflexive (resp. transitive, reflexive and symmetric) closure of *→D*.

We now introduce the type system *D* that characterizes the *simply typed differ- ential λ-calculus*.

## Type System *D*.

Γ(*x*) = *σ* Γ *▶D x* : *σ*

(Dx) Γ; *x* : *σ ▶D s* : *τ*

Γ *▶D λx.s* : *σ → τ*

(D*λ*) Γ *▶D s* : *σ → τ* Γ *▶D U* : *σ*

Γ *▶D sU* : *τ*

(D@)

Γ *▶D s* : *σ → τ* Γ *▶D t* : *σ*

Γ *▶D si* : *σ* for all 1 *≤ i ≤ k /*= 1 (D+)

Γ *▶D* D *s · t* : *σ → τ* (Dd)

*i*=1

Γ *▶D* Σ*k si* : *σ*

Note that, if *s* has a function type *σ → τ* and *t* has type *σ*, then D *s · t* has the same function type of *s*, thus the derivative does not decrease the type.

**Lemma 3.1** *Let* Γ; *x* : *σ ▶D S* : *τ and* Γ *▶D T* : *σ. We have:*

1. Γ *▶D S*[*T/x*] : *τ,*
2. Γ; *x* : *σ ▶D ∂S · T* : *τ,*

*∂x*

1. *(Subject Reduction for D) if T →D T ' then* Γ *▶D T '* : *σ.*

**Proof.** *(i)* and *(ii)* follow by straightforward induction on the length of the proofs of Γ *▶D S*[*T/x*] : *τ* and Γ; *x* : *σ ▶D ∂S · T* : *τ* , respectively.

*∂x*

*(iii)* Suppose *T ≡* (*λx.s*)*U* and *T ' ≡ S*[*U/x*], then it follows from (*i*). Suppose *T ≡* D(*λx.s*) *· U* and *T ' ≡ λx. ∂S · U* , then it follows from (*ii*). We conclude the proof since type derivations are contextual. *2*

*∂x*

**Definition 3.2** Let *T* be a collection of judgments of the shape Γ *▶ S* = *T* : *σ*

such that Γ *▶ S* : *σ* and Γ *▶ T* : *σ*. *T* is called a (typed) *differential λ-theory* if it is

6 The definition of differential substitution strongly uses the abbreviations introduced above.

closed under the following rules:

Γ *▶* (*λx.s*)*T* : *τ*

Γ *▶* (*λx.s*)*T* = *s*[*T/x*] : *τ*

Γ *▶* D(*λx.s*) *· t* : *τ*

(*β*) Γ *▶* D(*λx.s*) *· t* = *λx. ∂s · t* : *τ*

*∂x*

(*βD*)

Γ; *x* : *σ ▶ s* = *t* : *τ* (*ξ*) Γ *▶ s* = *v* : *σ → τ* Γ *▶ T* = *T '* : *σ*

(*Ap*)

Γ *▶ λx.s* = *λx.t* : *σ → τ* Γ *▶ sT* = *vT'* : *τ*

Γ *▶ s* = *t* : *σ x* : *τ ∈/* Γ (*W* ) Γ *▶ s* = *v* : *σ → τ* Γ *▶ t* = *t'* : *σ*

(*DAp*)

Γ; *x* : *τ ▶ s* = *t* : *σ* Γ *▶* D *s · t* = D *v · t'* : *σ → τ*

plus the obvious rules for symmetry, reflexivity, transitivity and the sums.

# A Differential Model Theory

In this section we will provide the categorical framework characterizing the models of the simply typed differential *λ*-calculus. The material presented in Subsection [4.1](#_bookmark5) is borrowed from [[3](#_bookmark28)].

* 1. *Cartesian Differential Categories*

A category **C** is *left-additive* whenever each homset has a structure of commutative monoid (**C**(*A, B*)*,* +*AB,* 0*AB*) and (*g* + *h*) *◦f* = (*g ◦f* )+ (*h◦f* ) and 0 *◦f* = 0.

A morphism *f* in **C** is said to be *additive* if, moreover, it satisfies *f ◦* (*g* + *h*) =

(*f ◦g*)+ (*f ◦h*) and *f ◦* 0 = 0.

A category is *Cartesian left-additive* if it is a left-additive category with prod- ucts such that all projections and pairings of additive maps are additive. A ccc is *Cartesian closed left-additive* if it is a Cartesian left-additive category satisfying:

(+-curry) Λ(*f* + *g*) = Λ(*f* )+ Λ(*g*) Λ(0) = 0 (0-curry)

**Remark 4.1** From (+-curry) it also follows that Λ*−*(*f* + *g*) = Λ*−*(*f* ) + Λ*−*(*g*). Precomposing *⟨*Id*, h⟩* on both sides we get ev *◦ ⟨f* + *g, h⟩* = ev *◦ ⟨f, h⟩* + ev *◦ ⟨g, h⟩*.

**Definition 4.2** A *Cartesian (closed) differential category* is a Cartesian (closed) left-additive category having an operator *D*(*−*) that maps a morphism *f* : *A → B* into a morphism *D*(*f* ) : *A × A → B* and satisfies the following axioms:

D1. *D*(*f* + *g*) = *D*(*f* )+ *D*(*g*) and *D*(0) = 0

D2. *D*(*f* ) *◦ ⟨h* + *k, v⟩* = *D*(*f* ) *◦ ⟨h, v⟩* + *D*(*f* ) *◦ ⟨k, v⟩* and *D*(*f* ) *◦ ⟨*0*, v⟩* = 0 D3. *D*(Id) = *π*1, *D*(*π*1) = *π*1 *◦π*1 and *D*(*π*2) = *π*2 *◦π*1

D4. *D*(*⟨f, g⟩*) = *⟨D*(*f* )*, D*(*g*)*⟩*

D5. *D*(*f ◦g*) = *D*(*f* ) *◦ ⟨D*(*g*)*,g ◦π*2*⟩*

D6. *D*(*D*(*f* )) *◦⟨⟨g,* 0*⟩, ⟨h, k⟩⟩* = *D*(*f* ) *◦ ⟨g, k⟩*

D7. *D*(*D*(*f* )) *◦⟨⟨*0*, h⟩, ⟨g, k⟩⟩* = *D*(*D*(*f* )) *◦⟨⟨*0*, g⟩, ⟨h, k⟩⟩*

We try to provide some intuitions on these axioms. (D1) says that the operator

*D*(*−*) is linear; (D2) says that *D*(*−*) is additive in its first coordinate; (D3) and

(D4) ask that *D*(*−*) behaves coherently with the product structure; (D5) is the usual chain rule; (D6) requires that *D*(*f* ) is linear in its first component. (D7) states the independence of order of “partial differentiation”.

**Remark 4.3** In a Cartesian differential category we obtain partial derivatives from the full ones by “zeroing out” the components on which the differentiation is not required. *E.g.*, suppose that we want to define the partial derivative *D*1(*f* ) of *f* : *C ×A → B* on its 1st component; then, it is sufficient to set *D*1(*f* ) = *D*(*f* ) *◦* (*⟨*Id*C,* 0*A⟩×* Id*C×A*): *C ×* (*C×A*) *→B*. Similarly, we define *D*2(*f* ): *A×* (*C×A*) *→ B*, the partial derivative of *f* on its 2nd component.

This remark follows since every differential *D*(*f* ) can be reconstructed from its partial derivatives as follows:

*D*(*f* ) = *D*(*f* ) *◦⟨⟨π*1 *◦π*1*, π*2 *◦π*1*⟩, π*2*⟩*

= *D*(*f* ) *◦⟨⟨π*1 *◦π*1*,* 0*⟩, π*2*⟩* + *D*(*f* ) *◦⟨⟨*0*, π*2 *◦π*1*⟩, π*2*⟩*

= *D*(*f* ) *◦* (*⟨*Id*,* 0*⟩×* Id) *◦* (*π*1 *×* Id) + *D*(*f* ) *◦* (*⟨*0*,* Id*⟩×* Id) *◦* (*π*2 *×* Id)

= *D*1(*f* ) *◦* (*π*1 *×* Id) + *D*2(*f* ) *◦* (*π*2 *×* Id)*.*

* 1. *Differential λ-Categories*

Cartesian closed differential categories are not enough to interpret the differential *λ*-calculus, since the differential operator does not behave automatically well with respect to the Cartesian close structure. For this reason, we now introduce the notion of *differential λ-category*.

**Definition 4.4** A *differential λ-category* is a Cartesian closed differential category such that, for all *f* : *C × A → B*:

*(D-curry) D*(Λ(*f* )) = Λ(*D*(*f* ) *◦ ⟨π*1 *×* 0*A, π*2 *×* Id*A⟩*)

Intuitively, (D-curry) requires that in a *λ*-category we have two equivalent ways to derivate *f* : *C×A → B* in its 1st component: we can use the trick of Remark [4.3](#_bookmark7), or we can ‘hide’ the component *A* by currying *f* and then derive Λ(*f* ).

**Lemma 4.5** *In every differential λ-category the following axiom holds (for all h* :

*C → A⇒B and g* : *C → A):*

(D-eval) *D*(ev *◦ ⟨h, g⟩*) = ev *◦ ⟨D*(*h*)*,g ◦π*2*⟩* + *D*(Λ*−*(*h*)) *◦⟨⟨*0*C, D*(*g*)*⟩, ⟨π*2*,g ◦π*2*⟩⟩.*

**Proof.** Let *h'* = Λ*−*(*h*) : *C × A → B*.

*D*(ev *◦ ⟨h, g⟩*) = by def. of *h'*

*D*(ev *◦ ⟨*Λ(*h'*)*, g⟩*) = by (beta-cat)

*D*(*h' ◦ ⟨*Id*C, g⟩*) = by (D5)

*D*(*h'*) *◦ ⟨D*(*⟨*Id*C, g⟩*)*, ⟨*Id*C, g⟩◦π*2*⟩* = by (D4) and (D3)

*D*(*h'*) *◦⟨⟨π*1*, D*(*g*)*⟩, ⟨π*2*,g ◦π*2*⟩⟩* = since pairing is additive

*D*(*h'*) *◦⟨⟨π*1*,* 0*A⟩* + *⟨*0*C, D*(*g*)*⟩, ⟨π*2*,g ◦π*2*⟩⟩* = by (D2)

*D*(*h'*) *◦⟨⟨π*1*,* 0*A⟩, ⟨π*2*,g ◦π*2*⟩⟩*

+ *D*(*h'*) *◦⟨⟨*0*C, D*(*g*)*⟩, ⟨π*2*,g ◦π*2*⟩⟩* = by (proj)

*D*(*h'*) *◦ ⟨π*1 *×* 0*A, π*2 *×* Id*A⟩◦ ⟨*Id*C×C,g ◦π*2*⟩*

+ *D*(*h'*) *◦⟨⟨*0*C, D*(*g*)*⟩, ⟨π*2*,g ◦π*2*⟩⟩* = by (beta-cat) ev *◦ ⟨*Λ(*D*(*h'*) *◦ ⟨π*1 *×* 0*A, π*2 *×* Id*A⟩*)*,g ◦π*2*⟩*

+ *D*(*h'*) *◦⟨⟨*0*C, D*(*g*)*⟩, ⟨π*2*,g ◦π*2*⟩⟩* = by (D-curry) ev *◦ ⟨D*(Λ(*h'*))*,g ◦π*2*⟩*

+ *D*(Λ*−*(Λ(*h'*))) *◦⟨⟨*0*C, D*(*g*)*⟩, ⟨π*2*,g ◦π*2*⟩⟩* = by def. of *h'*

ev *◦ ⟨D*(*h*)*,g ◦π*2*⟩* + *D*(Λ*−*(*h*)) *◦⟨⟨*0*C, D*(*g*)*⟩, ⟨π*2*,g ◦π*2*⟩⟩*

*2*

(D-eval) can be seen as a chain rule for denotations of terms (cf. Lemma [4.10](#_bookmark9)(i)).

**Definition 4.6** In differential *λ*-categories we are able to define a binary opera- toron morphisms, that can be seen as the semantic counterpart of differential substitution:

*f* : *C × A → B g* : *C → A* ()

*f g* : *C × A → B*

This operator is defined by *f g* = *D*(*f* ) *◦⟨⟨*0*C×A,g ◦π*1*⟩,* Id*C×A⟩*.

*C*

Intuitively, the morphism *f g* is obtained by force-feeding the 2nd argument *A* of *f* with *one copy* of the result of *g*. The type is not modified because *f g* may still depend on *A*.

**Definition 4.7** Let sw*ABC* = *⟨⟨π*1 *◦π*1*, π*2*⟩, π*2 *◦π*1*⟩* : (*A × B*) *× C →* (*A × C*) *× B*.

**Remark 4.8** sw*◦*sw = Id(*A×B*)*×C*, sw*◦⟨⟨f, g⟩, h⟩* = *⟨⟨f, h⟩, g⟩* and *D*(sw) = sw*◦π*1.

The following two technical lemmas will be used in Subsection [4.3](#_bookmark10) to show that differential *λ*-categories are models of the simply typed differential *λ*-calculus.

**Lemma 4.9** *Let f* : (*C × A*) *× D → B and g* : *C → A, h* : *C → B'. Then:*

1. *π*2 *g* = *g ◦π*1
2. (*h◦π*1) *g* = 0
3. Λ(*f* ) *g* = Λ(((*f ◦* sw)(*g ◦π*1)) *◦* sw)*.*

**Proof. (Outline)** (*i*) follows by applying *D3*. (*ii*) follows by applying *D2*, *D3* and

*D5*. (*iii*) follows by *(Curry)*, *(D-curry)* and *D2, D3, D5*. *2*

**Lemma 4.10** *Let f* : *C × A →* [*D⇒B*] *and g* : *C → A, h* : *C × A → D. Then:*

1. (ev *◦ ⟨f, h⟩*) *g* = ev *◦ ⟨f g* + Λ(Λ*−*(*f* )(*h g*))*, h⟩*
2. Λ(Λ*−*(*f* ) *h*) *g* = Λ(Λ*−*(*f g*) *h*)+ Λ(Λ*−*(*f* )(*h g*))
3. Λ(Λ*−*(*f* ) *h*) *◦ ⟨*Id*C, g⟩* = Λ(Λ*−*(*f ◦ ⟨*Id*C, g⟩*)(*h◦ ⟨*Id*C, g⟩*))

**Proof. (Outline)** *(i)* follows by applying *(D-eval)* and *(beta-cat)*.

* 1. This equation can be simplified by using the axioms of Cartesian closed left-additive categories. Indeed, the right side can be written as Λ((Λ*−*(*f g*) *h*)+ Λ*−*(*f* )(*h g*)). By taking a morphism *f'* such that *f* = Λ(*f'*) and by applying Lemma [4.9](#_bookmark8)(*iii*) the item (*ii*) becomes equivalent to ((*f' h*) *◦* sw)(*g ◦ π*1) *◦* sw = (((*f' ◦* sw)(*g ◦π*1)) *◦* sw) *h* + *f'* (*h g*)*.* This follows by *(Curry)* and *D2-7*.
  2. follows by *(Curry)* and *D2-5*. *2*
  3. *Interpreting the Differential Lambda Calculus*

In this section we define the interpretation of the simply typed differential *λ*-calculus in a fixed differential *λ*-category **C**.

Types are interpreted as follows:

* + - *|α|* = *A*, for some object *A*,
    - *|σ → τ|* = *|σ|⇒|τ |*.

Contexts are interpreted as usual:

* + - *|∅|* = ,
    - *|*Γ; *x* : *σ|* = *|*Γ*|× |σ|*.

The interpretation of a judgement Γ *▶ S* : *σ* will be a morphism from *|*Γ*|* to *|σ|*

denoted by *|Sσ|*Γ and defined inductively as follows.

## Interpretation of judgements

* + - *|xσ|*Γ;*x*:*σ* = *π*2 : *|*Γ*|× |σ|→ |σ|*,
    - *|yτ |*Γ;*x*:*σ* = *|yτ |*Γ *◦π*1 : *|*Γ*|× |σ|→ |τ |* for *x /*= *y*,
    - *|*(*sU* )*τ |*Γ = ev *◦⟨|sσ→τ |*Γ*, |Uσ|*Γ*⟩* : *|*Γ*|→ |τ |*,
    - *|*(*λx.s*)*σ→τ |*Γ = Λ(*|sτ |*Γ;*x*:*σ*) : *|*Γ*|→ |σ|⇒|τ |*,
    - *|*(D *s · t*)*σ→τ |*Γ = Λ(Λ*−*(*|sσ→τ |*Γ) *|tσ|*Γ) : *|*Γ*|→ |σ|⇒|τ |*,
    - *|*0*σ|*Γ = 0 : *|*Γ*|→ |σ|*,
    - *|*(*s* + *S*)*σ|*Γ = *|sσ|*Γ + *|Sσ|*Γ : *|*Γ*|→ |σ|*.

We will sometimes omit the superscript *σ* in *|Sσ|*Γ, when there is no chance of confusion. Given a differential *λ*-category **C** we can define the *theory of* **C** by:

Th(**C**) = *{*Γ *▶ S* = *T* : *σ |* Γ *▶D S* : *σ,* Γ *▶D T* : *σ, |Sσ|*Γ = *|Tσ|*Γ*}.*

At the end of this section we will prove that the interpretation *|−|* is *sound* for the differential *λ*-calculus, *i.e.,* that Th(**C**) is a differential *λ*-theory. In order to prove this result, we need first some technical results.

The following lemma is easy to prove.

**Lemma 4.11** *Let S ∈* Λ*d, then:*

1. *If* Γ *▶D S* : *τ and x ∈/* FV(*S*) *then |Sτ |*Γ;*x*:*σ* = *|Sτ |*Γ *◦π*1*,*
2. *|S|*Γ;*x*:*σ*;*y*:*τ* = *|S|*Γ;*y*:*τ* ;*x*:*σ ◦* sw*.*

**Theorem 4.12** *(Substitutions) Let* Γ; *x* : *σ ▶D S* : *τ and* Γ *▶D T* : *σ, then:*

1. *|*(*S*[*T/x*])*τ |*Γ = *|Sτ |*Γ;*x*:*σ ◦ ⟨*Id*|*Γ*|, |Tσ|*Γ*⟩,*
2. *|*( *∂S · T* )*τ |*Γ;*x*:*σ* = *|Sτ |*Γ;*x*:*σ |Tσ|*Γ*.*

*∂x*

**Proof.** *(i)* By induction on *S*. The only interesting case is *S ≡* D *s · u*.

By def. of substitution we have *|*(D *s· u*)[*T/x*]*|*Γ = *|*D *s*[*T/x*] *· u*[*T/x*]*|*Γ. By def. of

*|−|* this is equal to Λ(Λ*−*(*|s*[*T/x*]*|*Γ) *|u*[*T/x*]*|*Γ). By induction hypothesis (IH) we get Λ(Λ*−*(*|s|*Γ;*x*:*σ ◦ ⟨*Id*, |T |*Γ*⟩*)(*|u|*Γ;*x*:*σ ◦ ⟨*Id*, |T |*Γ*⟩*)). By applying Lemma [4.10](#_bookmark9)(*iii*) this is equal to Λ(Λ*−*(*|s|*Γ;*x*:*σ*) *|u|*Γ;*x*:*σ*) *◦ ⟨*Id*, |T |*Γ*⟩* = *|*D *s · u|*Γ;*x*:*σ ◦ ⟨*Id*, |T |*Γ*⟩*.

*(ii)* By structural induction on *S*.

* + case *S ≡ x*. Then *|∂x · T|*Γ;*x*:*σ* = *|T |*Γ;*x*:*σ* = *|T |*Γ *◦ π*1 = *π*2 *|T |*Γ = *|x|*Γ;*x*:*σ |T |*Γ

*∂x*

by Lemma [4.9](#_bookmark8)(*i*).

* + case *S ≡ z /*= *x*. Then *| ∂z · T|*Γ;*x*:*σ* = *|*0*|*Γ;*x*:*σ* = 0. By Lemma [4.9](#_bookmark8)(*ii*) we have 0 = (*|z|*Γ *◦π*1) *|T |*Γ = *|z|*Γ;*x*:*σ |T |*Γ.

*∂x*

* + case *S ≡ λz.v*. By def. of differential substitution we have that *| ∂*(*λz.v*) *· T|*Γ;*x*:*σ* =

*∂x*

*|λz. ∂v · T|*Γ;*x*:*σ* = Λ(*|∂v · T|*Γ;*x*:*σ*;*z*:*γ*). Applying Lemma [4.11](#_bookmark11)(*ii*), this is equal to

*∂x ∂x*

Λ(*|∂v · T|*Γ;*z*:*γ*;*x*:*σ ◦* sw). By IH we obtain Λ((*|v|*Γ;*z*:*γ*;*x*:*σ |T |*Γ;*z*:*γ*) *◦* sw). Sup-

*∂x*

posing wlog that *z ∈/* FV(*T* ) we have *|T |*Γ;*z*:*γ* = *|T |*Γ *◦ π*1 (by Lemma [4.11](#_bookmark11)(*i*)).

Thus, by Lemma [4.9](#_bookmark8)(*iii*), we have that Λ((*|v|*Γ;*z*:*γ*;*x*:*σ* (*|T |*Γ *◦ π*1)) *◦* sw) = Λ(*|v|*Γ;*z*:*γ*;*x*:*σ ◦* sw) *|T |*Γ, which is Λ(*|v|*Γ;*x*:*σ*;*z*:*γ*) *|T |*Γ by Lemma [4.11](#_bookmark11)(*ii*). We conclude since Λ(*|v|*Γ;*x*:*σ*;*z*:*γ*) *|T |*Γ = *|λz.v|*Γ;*x*:*σ |T |*Γ.

* + case *S ≡ sU* . By def. of differential substitution we have that *| ∂*(*sU*) *· T|*Γ;*x*:*σ* =

*∂x*

*|*(

*∂s · T* )*U|*Γ;*x*:*σ* + *|*(D *s ·* ( *∂U · T* ))*U|*Γ;*x*:*σ*. Let us consider the two addenda

*∂x*

*∂x*

componentwise. We have *|*( *∂s · T* )*U|*Γ;*x*:*σ* = ev *◦ ⟨| ∂s · T|*Γ;*x*:*σ, |U|*Γ;*x*:*σ⟩* which

*∂x ∂x*

is equal, by IH, to ev *◦ ⟨|s|*Γ;*x*:*σ |T |*Γ*, |U|*Γ;*x*:*σ⟩*. On the other side we have:

*|*(D *s ·* ( *∂U · T* ))*U|*Γ;*x*:*σ* = ev *◦ ⟨*Λ(Λ*−*(*|s|*Γ;*x*:*σ*) *|∂U · T|*Γ;*x*:*σ*)*, |U|*Γ;*x*:*σ⟩*, by IH

*∂x*

*∂x*

this is equal to ev *◦ ⟨*Λ(Λ*−*(*|s|*Γ;*x*:*σ*)(*|U|*Γ;*x*:*σ |T |*Γ))*, |T |*Γ;*x*:*σ⟩*. By applying

Remark [4.1](#_bookmark6) we can rewrite the sum of this two addenda as follows: ev *◦*

*⟨|s|*Γ;*x*:*σ |T |*Γ + Λ(Λ*−*(*|s|*Γ;*x*:*σ*)(*|U|*Γ;*x*:*σ |T |*Γ))*, |U|*Γ;*x*:*σ⟩.* By Lemma [4.10](#_bookmark9)(*i*) this is (ev *◦⟨|s|*Γ;*x*:*σ, |U|*Γ;*x*:*σ⟩*) *|T |*Γ = *|sU|*Γ;*x*:*σ |T |*Γ.

* + - case *S ≡* D *v · u*. By definition, we have that *|* *∂* (D *v · u*) *· T|*Γ;*x*:*σ* =

*∂x*

*|*D( *∂v · T* ) *· u|*Γ;*x*:*σ* + *|*D *v ·* ( *∂u · T* )*|*Γ;*x*:*σ*. Consider the two addenda separately.

*∂x ∂x*

*|*D( *∂v · T* ) *· u|*Γ;*x*:*σ* = Λ(Λ*−*(*| ∂v · T|*Γ;*x*:*σ*) *|u|*Γ;*x*:*σ*). By IH this is equal to

*∂x ∂x*

Λ(Λ*−*(*|v|*Γ;*x*:*σ |T |*Γ) *|u|*Γ;*x*:*σ*). On the other hand, we have that *|*D *v·*( *∂u·T* )*|*Γ;*x*:*σ* =

*∂x*

Λ(Λ*−*(*|v|*Γ;*x*:*σ*) *|∂u · T|*Γ;*x*:*σ*). By IH this is Λ(Λ*−*(*|v|*Γ;*x*:*σ*)(*|u|*Γ;*x*:*σ |T |*Γ)). By applying Lemma [4.10](#_bookmark9)(*ii*) to the sum of the two morphisms, we obtain Λ(Λ*−*(*|v|*Γ;*x*:*σ*) *|u|*Γ;*x*:*σ*) *|T |*Γ which is equal to *|*D *v · u|*Γ;*x*:*σ |T |*Γ.

*∂x*

* + - all other cases (*i.e., S ≡* 0 and *S ≡ s* + *U* ) are straightforward.

*2*

We are now able to prove the main theorem of this section.

**Theorem 4.13** *Let* **C** *be a differential λ-category. Then* Th(**C**) *is a differential*

*λ-theory.*

**Proof.** We have to check that Th(**C**) is closed under the rules presented in Def. [3.2](#_bookmark2). (*β*) Let *|*(*λx.s*)*T|*Γ = ev *◦ ⟨*Λ(*|s|*Γ;*x*:*σ*)*, |T |*Γ*⟩*. By the Theorem [4.12](#_bookmark12)(i), we have *|s*[*T/x*]*|*Γ = *|s|*Γ;*x*:*σ ◦ ⟨*Id*, |T |*Γ*⟩* and, by (beta-cat), *|s|*Γ;*x*:*σ ◦ ⟨*Id*, |T |*Γ*⟩* =

ev *◦ ⟨*Λ(*|s|*Γ;*x*:*σ*)*, |T |*Γ*⟩*.

(*βD*) Let *|*D(*λx.s*)*·t|*Γ = Λ(Λ*−*(Λ(*|s|*Γ;*x*:*σ*)) *|t|*Γ) = Λ(*|s|*Γ;*x*:*σ |t|*Γ)*.* By applying Theorem [4.12](#_bookmark12)(ii), this is equal to Λ(*| ∂s · t|*Γ;*x*:*σ*) = *|λx. ∂s · t|*Γ.

*∂x ∂x*

For the weakening (*W* ) we use Lemma [4.11](#_bookmark11)(*i*). Symmetry, reflexivity and tran-

sitivity hold since Th(**C**) is an equivalence. The rule for sums follows from the definition of the interpretation of sums. Finally, (*ξ*), (*Ap*) and (*DAp*) follow by definition of the interpretation of abstraction, application and linear application

(respectively). *2*

# Examples of Differential *λ*-Categories

In this section we provide two examples of differential *λ*-category: (i) **MRel**, which is the co-Kleisli category of the functor *Mf* (*−*) over the-autonomous category **Rel** of sets and relations [[11](#_bookmark32),[5](#_bookmark26)]; (ii) the category **MFin**, which is the co-Kleisli of the functor *Mf* (*−*) over the autonomous category of finiteness spaces and finitary

relations [[7](#_bookmark25)].

* 1. *Relational Semantics*

We provide here a direct definition of the category **MRel**:

* + - The objects of **MRel** are all the sets.
    - A morphism from *A* to *B* is a relation from *Mf* (*A*) to *B*, in other words,

**MRel**(*A, B*) = *P*(*Mf* (*A*) *× B*).

* + - The identity of *A* is the relation Id*A* = *{*([*a*]*, a*) *| a ∈ A}∈* **MRel**(*A, A*).
    - The composition of *s ∈* **MRel**(*A, B*) and *t ∈* **MRel**(*B, C*) is defined by:

*t◦s* = *{*(*m, c*) *| ∃*(*m*1*, b*1)*,...,* (*mk, bk*) *∈ s* such that

*m* = *m*1 *... mk* and ([*b*1*,..., bk*]*, c*) *∈ t}.*

**Theorem 5.1** *The category* **MRel** *is a differential λ-category.*

**Proof. (Outline)** The fact that **MRel** is a ccc is proved, for instance, in [[5](#_bookmark26)]. The categorical product [7](#_bookmark14) (&) is the disjoint union (*∪*˙ ) and *∅* is the terminal object. Given *si ∈* **MRel**(*A, Bi*) (for *i* = 1*,* 2), the corresponding morphism *⟨s*1*, s*2*⟩ ∈* **MRel**(*A, B*1 &*B*2) is given by: *⟨s*1*, s*2*⟩* = *{*(*m,* (*i, a*)) *|* (*m, a*) *∈ si,* for *i* = 1*,* 2*}*.

We consider the canonical bijection between *Mf* (*A*1) *× Mf* (*A*2) and *Mf* (*A*1 & *A*2) as an equality, hence we will still denote by (*m*1*, m*2) the corresponding element of *Mf* (*A*1 &*A*2). Given two sets *A, B* we have *A⇒B* = *Mf* (*A*) *× B* and:

ev*AB* = *{*(([(*m, b*)]*, m*)*, b*) *| m ∈ Mf* (*A*) and *b ∈ B}∈* **MRel**((*A⇒B*)&*A, B*) *.*

Given any set *A* and any morphism *s ∈* **MRel**(*A* & *B, C*), there is exactly one morphism Λ(*s*) = *{*(*p,* (*m, b*)) *|* ((*p, m*)*, b*) *∈ s} ∈* **MRel**(*A, B ⇒ C*) such that ev*BC ◦ ⟨*Λ(*s*)*,* Id*B⟩* = *s*.

**MRel** is a Cartesian closed left-additive category since the homsets can be en- dowed with the following additive structure (**MRel**(*A, B*)*, ∪, ∅*).

Finally, given *f ∈* **MRel**(*A, B*) we can define its derivative as follows:

*D*(*f* ) = *{*(([*a*]*, m*)*, b*) *|* (*m* [*a*]*, b*) *∈ f}∈* **MRel**(*A*&*A, B*)*.*

It is not difficult to check that *D*(*−*) satisfies (D1-7) and (D-curry). *2*

Thus, the operationcan be directly defined in **MRel** as follows:

*f g* = *{*((*m*1 *m*2*, m*)*, b*) *|* (*m*1*, a*) *∈ g,* ((*m*2*,m* [*a*])*, b*) *∈ f}* : *C* &*A → B.*

* 1. *Finiteness Spaces Semantics*

We provide here a brief account of [[7](#_bookmark25)], in order to give a direct presentation of the Cartesian closed category of finiteness spaces and finitary relations. All the categorical constructions are tightly related to the corresponding ones in **MRel**.

Let *X* be a set and *a, b ⊆ X*. We say that *a, b* are *orthogonal*, written *a⊥b*, if

*a ∩ b* is a finite set. Given *F ⊆ P*(*X*), we set *F⊥* = *{ b ∈ P*(*X*) *| ∀a ∈F a⊥b }*.

**Definition 5.2** A *ﬁniteness space* is a pair *X* = (*X,* F(*X* )) where *X* is a countable set and F(*X* ) is a subset of *P*(*X*) satisfying F(*X* )*⊥⊥* = F(*X* ). The elements of F(*X* ) are called the *ﬁnitary sets* of *X* .

Given a finiteness space *X* = (*X,* F(*X* )), define !*X* = (*Mf* (*X*)*,* F(!*X* )) where

F(!*X* ) = *{U ⊆ Mf* (*X*) *|* supp(*U* ) *∈* F(*X*)*}*.

7 In this section the symbol *×* is kept to denote the usual set-theoretical Cartesian product.

**Definition 5.3** A *ﬁnitary relation* from *X* to *Y* is a relation *R ⊆ X × Y* such that:

* for all *a ∈* F(*X* ), *R*(*a*) = *{β ∈ Y | ∃α ∈ a* (*α, β*) *∈ R}∈* F(*Y*), and
* for all *β ∈ Y* , *R⊥*(*β*) = *{α ∈ X |* (*α, β*) *∈ R}∈* F(*X* )*⊥*.

The category **MFin** can be directly defined as follows:

* The objects are the finiteness spaces.
* A morphism from *X* to *Y* is a finitary relation from !*X* to *Y*.
* Identities and composition are defined as in **MRel**.

**Theorem 5.4** *The category* **MFin** *is a differential λ-category.*

**Proof. (Outline)** The categorical product *X* &*Y* of the finiteness spaces *X* and *Y* is (*X* &*Y,* F(*X* &*Y*)), where *X* &*Y* = *X∪*˙ *Y* and F(*X* &*Y* ) = *{a∪*˙ *a' | a ∈* F(*X* )*, a' ∈* F(*Y*)*}*. Projections and pairing are precisely like in **MRel**.

The exponential object *X ⇒ Y* = (*Mf* (*X*) *× Y,* F(*X ⇒ Y*)); we refer to [[7](#_bookmark25)] for the precise definition of F(*X ⇒ Y*). The additive structure on homsets, Λ(*−*), ev and *D*(*−*) are defined as in **MRel**. To conclude the proof it is sufficient to check that ev is a finitary morphism and that for every finitary *f* of the correct type we have that Λ(*f* ) and *D*(*f* ) are finitary.

We explicit here the case of *D*(*f* ). We then want to show that for every *f ∈* **MFin**(*X, Y*) we have *D*(*f* ) = *{*(([*α*]*, m*)*, β*) *|* (*m* [*α*]*, β*) *∈ f}∈* **MFin**(*X* &*X, Y*). As a preliminary remark we note that, for every *X* , F(*X* ) and F(*X* )*⊥* contain all finite subsets of *X* and are closed with respect to arbitrary intersections, finite unions and subsets. As in the preceding subsection we will use implicitly the isomorphism

between *Mf* (*X∪*˙ *Y* ) and *Mf* (*X*) *× Mf* (*Y* ).

We start by proving that, given a finitary set *U ∈* F(!(*X* & *X* )), we have (*D*(*f* ))(*U* ) *∈* F(*Y*). By definition of F(!(*X* & *X* )), we have that *π*1(supp(*U* )),

*π*2(supp(*U* )) *∈* F(*X* ), and hence *π*1(supp(*U* )) *∪ π*2(supp(*U* )) *∈* F(*X* ). Now, defin- ing *U'* = *{m*1 *m*2 *|* (*m*1*, m*2) *∈ U}*, we have that *π*1(supp(*U* )) *∪ π*2(supp(*U* )) =

supp(*U'*), hence *U' ∈ F* (!*X* ), and *f* (*U'*) *∈* F(*Y*), *f* being finitary. Since (*D*(*f* ))(*U* ) *⊆*

*f* (*U'*), we have finished.

It remains to show that, for all *β ∈ Y* , we have (*D*(*f* ))*⊥*(*β*) *∈* F(!(*X* & *X* ))*⊥*. Given *U ∈* F(!(*X* &*X* )), we define *U'* = *{m*1 *m*2 *|* (*m*1*, m*2) *∈ U}∈* F(*X* ) as above. We know that *f⊥*(*β*) *∩ U'* is finite, and we have to show that (*D*(*f* ))*⊥*(*β*) *∩ U* is finite. It is easy to see that (*D*(*f* ))*⊥*(*β*) *∩ U* = *{*([*α*]*, m*) *| m* [*α*] *∈ f⊥*(*β*) *∩ U'}*, *m* being a finite multiset. Hence *D*(*f* )*⊥*(*β*) *∩U* is finite, *f⊥*(*β*) *∩U'* being itself finite.*2*

**Remark 5.5** We may have presented another differential *λ*-category of finiteness spaces, which is obtained from **MFin** by considering as objects the *R*-vector spaces (for some field *R*) associated to the finiteness spaces, and as morphisms the contin- uous and linear maps between *R*-vector spaces [[7](#_bookmark25), pag. 20]. However, this category is unnecessarily complicated for our purposes.

# The Resource Calculus

* 1. *Its Syntax*

In this section we present the resource calculus [[4](#_bookmark23)] (using the formalization given in [[12](#_bookmark33)]) and we show that every differential *λ*-category is also a model of its simply typed version. In this calculus there are three syntactical sorts: *resource λ-terms* (Λ*r* ) that are in functional position; *bags* (Λ*b*) that are in argument position and represent multisets of resources, and *sums* that represent the possible results of a computation. A *resource* (Λ(!)) can be linear or intuitionistic, in the latter case it is written with a ! apex. An *expression* (Λ(*b*)) is either a term or a bag.

Formally, we have:

Λ*r* : *M, N, L* ::= *x | λx.M | MP* resource *λ*-terms Λ(!) : *M* (!)*,N* (!) ::= *M | M* ! resources

Λ*b* : *P, Q, R* ::= [*M* (!)*,...,M* (!)] bags

1 *n*

Λ(*b*) : *A, B* ::= *M | P* expressions

Concerning sums, *N⟨*Λ*r⟩* (resp. *N⟨*Λ*b⟩*) denotes the set of finite formal sums of terms (resp. bags), with 0 referring to the neutral element.

*,* *∈ N⟨*Λ*r⟩* *,* *∈ N⟨*Λ*b⟩* *,* *∈ N⟨*Λ(*b*)*⟩* = *N⟨*Λ*r⟩∪N ⟨*Λ*b⟩* sums

Note that *N⟨*Λ(*b*)*⟩* does not denote the *N* -module generated over Λ(*b*) = Λ*r ∪* Λ*b* but rather the union of the two *N* -modules. In other words, sums must be taken only in the same sort.

**Notation 2** *We introduce the following abbreviations:*

* + - *λx.*Σ*iMi* = *λx.*(Σ*iMi*)*,*
    - Σ*i*[*Mi*] *P* = [(Σ*iMi*)] *P,*
    - Σ*i,jMiPj* = (Σ*iMi*)(Σ*jPj*)*,*
    - [*M* ! *,...,M* ! ] *P* = [(Σ*iMi*)!] *P.*

1 *k*

We will write *L→* for *L*1*,..., Lk* and *N→* ! for *N* ! *,...,N* ! . We will also abbreviate

1 *n*

*M⟨L*1*/x⟩··· ⟨Lk/x⟩* in *M⟨L→ /x⟩*. Moreover, given a sequence *L→ i ≤ k* we set *L→ −i* = *L*1*,..., Li−*1*, Li*+1*,..., Lk*.

and an integer 1 *≤*

Every applicative term *MP* can be written in a unique way as *M* [*L→ , N→* !].

The reduction rule generating the *resource calculus* is the following:

where:

*i*=1

(*λx.M* )[*L→ , N→* !] *→R M⟨L→ /x⟩*[Σ*n*

*Ni/x*]

* + - *A*[*N/x*] is the usual substitution of *N* for *x* in *A*. It is extended to sums as in

[ */x*] by linearity in .

* + - * *A⟨N/x⟩* is the *linear substitution* defined inductively as follows:

*y⟨N/x⟩* = ,⎨ *N* if *x* = *y*

, 0 otherwise

(*λy.M* )*⟨N/x⟩* = *λy.M⟨N/x⟩*

(*MP* )*⟨N/x⟩* = *M⟨N/x⟩P* + *M* (*P⟨N/x⟩*)

[*M* ]*⟨N/x⟩* = [*M⟨N/x⟩*] []*⟨N/x⟩* = 0

[*M* !]*⟨N/x⟩* = [*M⟨N/x⟩, M* !] (*P R*)*⟨N/x⟩* = *P⟨N/x⟩ R* + *P R⟨N/x⟩*

It is extended to *⟨* */x⟩* by bilinearity [8](#_bookmark17) in both and .

The operation *M⟨N/x⟩* on resource *λ*-terms is roughly equivalent to the opera- tion *∂S · T* on differential *λ*-terms (cf. Lemma [6.2](#_bookmark18) below).

*∂x*

We introduce the type system *R* characterizing the *simply typed resource calcu-*

*lus*.

## Type System *R*.

Γ(*x*) = *σ* Γ *▶R x* : *σ*

(Rx) Γ*,x* : *σ ▶R M* : *τ*

Γ *▶R λx.M* : *σ → τ*

(R*λ*)

Γ *▶R M* : *σ → τ* Γ *▶R P* : *σ* Γ *▶R MP* : *τ*

(R@) Γ *▶R N* : *σ* Γ *▶R P* : *σ* Γ *▶R* [*N* (!)] *P* : *σ*

(Rb)

Γ *▶R* [] : *σ*

(R[]) Γ *▶R A* : *σ* Γ *▶R* : *σ*  */*= 0 (R+)

Γ *▶R A* + : *σ*

* 1. *Its Semantics*

In this subsection we show that differential *λ*-categories are models also of the simply typed resource calculus. This result is achieved by first translating the resource calculus in the differential *λ*-calculus, and then applying the machinery of Section [4.3](#_bookmark10).

We add the *permutative equality* to differential *λ*-calculus, i.e., we consider dif- ferential *λ*-terms up to the following equivalence D(D *s · u*) *· v* = D(D *s · v*) *· u*. This is useful for the translation since, in the resource calculus, bags are considered as multisets (thus they are equal up to permutation of resources).

**Remark 6.1** It is not difficult to check that in every differential *λ*-category the interpretations of D(D *s · u*) *· v* and D(D *s · v*) *· u* coincide.

We can now easily translate the resource calculus into the differential *λ*-calculus as follows:

* + - *xo* = *x*,
    - (*λx.M* )*o* = *λx.Mo*,
    - (*M* [*L→ , N→* !])*o* = (D*k Mo · L→ o*)(Σ*n No*),

*i*=1

*i*

8 *F* (*A, B*) is extended by bilinearity by setting *F* (Σ*iAi,* Σ*j Bj* )= Σ*i,j F* (*Ai, Bj* ).

where D*k Mo · L→ o* is an abbreviation for D(*···* (D *Mo · L→ o*) *·· · · Lo* ). The translation

1 *k*

*i*=1

is then extended to elements in *N⟨*Λ*r⟩* by setting (Σ*n*

*i*=1

*i*

*Mi*)*o* = Σ*n*

*Mo*.

**Lemma 6.2** *Let M, N ∈* Λ*r and x be a variable. Then:*

1. (*M⟨N/x⟩*)*o* = *∂Mo · No,*

*∂x*

1. (*M* [*N/x*])*o* = *Mo*[*No/x*]*.*

**Proof.** (*i*) By structural induction on *M* . The only difficult case is *M ≡ M'*[*L→ , N→* !]. By definition of (*−*)*o* and of linear substitution we have:

((*M'*[*L→ , N→* !])*⟨N/x⟩*)*o* = (*M'⟨N/x⟩*[*L→ , N→* !])*o* + (*M'*([*L→ , N→* !]*⟨N/x⟩*))*o* =

(*M'⟨N/x⟩*[*L→ , N→* !])*o* + (Σ*k*

*j*=1

*M'*[*Lj⟨N/x⟩, L→ −j, N→* !])*o* + (Σ*n*

*M'*[*Ni⟨N/x⟩, L→ , N→* !])*o .*

` ˛¸ x

(1)

` ˛¸ x

*i*=1

(2)

` ˛¸ x

(3)

Let us consider the three addenda separately.

* 1. By definition of the translation (*−*)*o* we have that (*M'⟨N/x⟩*[*L→ , N→* !])*o* =

(D*k* (*M'⟨N/x⟩*)*o · L→ o*)(Σ*n No*). By applying the induction hypothesis, this is equal

to (

*k* ( *∂M'o*

*o → o*

*i*=1 *i*

*o*

*i*=1

D *∂x*

*· N* ) *· L* )(Σ*n*

*Ni* ).

* 1. By definition of the map (*−*)*o* we have (Σ*k M'*[*Lj⟨N/x⟩, L→ −j, N→* !])*o* =

*j*=1

)(Σ

*i*

*k j*=1

Σ

(D*k−*1 (D *M'o ·* (*Lj⟨N/x⟩*)*o*) *· L→ o*

*∂Lo*

*−j*

*n i*=1

*No*). By the induction hypothesis, this

is equal to Σ*k*

(D*k−*1 (D *M'o ·* ( *j · No*)) *· L→ o*

)(Σ*n No*).

*j*=1

*∂x −j*

*i*=1 *i*

* 1. By definition of the map (*−*)*o* we have (Σ*n M'*[*Nj⟨N/x⟩, L→ , N→* !])*o* =

*j*=1

*j*=1

*i*

*n* *j*=1

Σ

(*M'*[*Nj⟨N/x⟩, L→ , N→* !])*o* = Σ*n*

(D*k* (D *M'o ·* (*Nj⟨N/x⟩*)*o*) *· L→ o*)(Σ*n*

*∂No*

*i*=1

*No*). By in-

duction hypothesis, this is equal to Σ*n*

(D*k* (D *M'o ·* ( *j · No*)) *· L→ o*)(Σ*n*

*No*). By

*j*=1

*∂x*

*∂No*

*i*=1 *i*

permutative equality this is equal to Σ*n* (D(D*k M'o · L→ o*) *·* ( *j · No*))(Σ*n No*).

*j*=1 *∂x i*=1 *i*

To conclude the proof it is sufficient to check that *∂* ((D*k M'o · L→ o*)(Σ*n No*)) *· No*

*∂x*

is equal to the sum of (1), (2) and (3).

*i*=1 *i*

(*ii*) By straightforward induction on *M* . *2*

**Proposition 6.3** *For all M ∈* Λ*r we have:*

1. *M →R N implies Mo* →*D No.*
2. Γ *▶R M* : *σ ⇐⇒* Γ *▶D Mo* : *σ*
3. *(Subject Reduction for R)* Γ *▶R M* : *σ and M →R N implies* Γ *▶R N* : *σ.*

**Proof.** (*i*) Let *M ≡* (*λx.M'*)[*L→ , N→* !] and *N ≡ M'⟨L→ /x⟩*[Σ*n Ni/x*]. By def-

*i*=1

inition of (*−*)*o* we have ((*λx.M'*)[*L→ , N→* !])*o* = (D*k* (*λx.M'o*) *· L→ o*)(Σ*n No*) →*D*

*i*=1

*i*

*M'o⟨L→ o/x⟩*[Σ*n No/x*] which is equal to *No* by Lemma [6.2](#_bookmark18).

*i*=1

*i*

* 1. by induction on the length of the proofs of Γ *▶R M* : *σ* and Γ *▶D Mo* : *σ*.
  2. Suppose Γ *▶R M* : *σ* and *M →R N* . Then by (*ii*) we have that Γ *▶D Mo* : *σ*. By (*i*) we know that *Mo* →*D No* and since System *D* enjoys the subject reduction we get Γ *▶D No* : *σ*. We conclude by (*ii*). *2*

**Remark 6.4** The two results above generalize easily to sums of terms (*i.e.*, to elements *∈ N⟨*Λ*r⟩*).

Resource *λ*-terms can be interpreted in any differential *λ*-category trough their translation (*−*)*o*. Indeed, it is sufficient to set *|*Γ *▶R M* : *σ|* = *|*(*Mo*)*σ|*Γ. From this fact, Proposition [6.3](#_bookmark19) and Remarks [6.1](#_bookmark16),[6.4](#_bookmark20) it follows that differential *λ*-categories are models of the simply typed resource calculus.

# Conclusions and Further Works

In this paper we have provided sufficient conditions on Cartesian closed differential categories for being models of the simply typed differential *λ*-calculus. We have also shown that they can also be used for interpreting the simply typed resource calculus, by providing a faithful translation between the two calculi.

However, differential *λ*-categories may also provide general mathematical frame- works in which many models of the *untyped* differential *λ*-calculus may live. This is the case of the category **MRel**, while it is not the case of **MFin** since it does not contain any reflexive object. In a forthcoming paper we will provide suitable conditions on the reflexive objects of a differential *λ*-category for being models of the untyped differential *λ*-calculus (intuitively the retraction should be ‘linear’, in a sense to be specified). We will then show that the reflexive object *D* we have built in [[5](#_bookmark26)] satisfies these conditions.

Another interesting line of research is to characterize categorical models of the differential *λ*-calculus at the level of SMCC’s (symmetric monoidal closed cate- gories). In [[3](#_bookmark28)] Blute *et al.* show that (monoidal) differential categories [[2](#_bookmark27)] give rise to Cartesian differential categories via the co-Kleisli construction. In the same spirit, we would like to provide sufficient and necessary conditions on SMCC’s for giving rise to differential *λ*-categories (indeed, the examples of differential *λ*-categories we gave in Section [5](#_bookmark13) may be generated in such a way).

Notice that, in monoidal frameworks, categorical proofs become often awkward due to the symmetric properties of the tensor product *⊗* . It would be then interest- ing to define a graphical formalism allowing to represent in a pleasant and intuitive way the morphisms of these categories. This formalism could be inspired by differ- ential proofnets or interaction nets [[9](#_bookmark30)], but should satisfy (at least) the following properties: there should be a 1-to-1 correspondence between a morphism and its graphical representation (maybe up to some well chosen equivalence on morphisms); the formalism should not ask for extra properties of the category, like the presence of the operator ` or the dualizing object *⊥*.

# Acknowledgement

Many thanks to Guy McCusker, Michele Pagani, Paolo Tranquilli and the anony- mous reviewers for helpful comments and suggestions.

# References

1. A. Asperti and G. Longo. Categories, types and structures. Category theory for the working computer scientist. M.I.T. Press, 1991.
2. R.F. Blute, J.R.B. Cockett and R.A.G. Seely. *Differential categories*. Math. Struct. in Comp. Sci.,16(6):1049–1083, Cambridge University Press, 2006.
3. R.F. Blute, J.R.B. Cockett and R.A.G. Seely. *Cartesian differential categories*. Theory and Applications of Categories, Vol. 22, No. 23, pp 622-672, 2009.
4. G. Boudol. *The lambda-calculus with multiplicities*. CONCUR’93, pp. 1-6, 1993.
5. A. Bucciarelli, T. Ehrhard and G. Manzonetto. *Not enough points is enough.* CSL’07, pp. 298-312, 2007.
6. T. Ehrhard and L. Regnier. *The differential lambda-calculus.* Theor. Comput. Sci. 309(1-3): 1-41, 2003.
7. T. Ehrhard. *Finiteness spaces.* Math. Struct. in Comp. Sci. 15(4): 615-646, 2005.
8. T. Ehrhard and L. Regnier. *B¨ohm trees, Krivine’s machine and the Taylor expansion of λ-terms*. CiE,

pp. 186-197, 2006.

1. T. Ehrhard and L. Regnier. *Differential interaction nets*. Theor. Comput. Sci. 364(2): 166-195, 2006.
2. T. Ehrhard and L. Regnier. *Uniformity and the Taylor expansion of ordinary λ-terms*. Theor. Comput. Sci. 403(2-3): 347-372, 2008.
3. J.-Y. Girard. *Normal functors, power series and λ-calculus*. APAL. 37(2):129-177, 1988.
4. M. Pagani and P. Tranquilli. *Parallel reduction in resource λ-calculus*. APLAv09, pp. 226-242, 2009.
5. P. Tranquilli. *Nets Between Determinism and Nondeterminism*. PhD thesis, Univ. of Paris 7 and Univ. of Roma 3, 2009.