

A Polymorphic Type System for Lambda Calculus with Constructors

Barbara Petit

LIP - ENS Lyon

46 Allée d'Italie, 69364 Lyon, France

<http://perso.ens-lyon.fr/barbara.petit>

Abstract. We present a Curry-style second-order type system with union and intersection types for the lambda-calculus with constructors of Arbiser, Miquel and Rios, an extension of lambda-calculus with a pattern matching mechanism for variadic constructors. To prove the strong normalisation property for this system, we translate well-typed terms in an auxiliary calculus of case-normal forms using the interpretation method. We finally prove the strong normalisation property for the auxiliary calculus using the standard reducibility method.

Key words: lambda-calculus, polymorphism, pattern matching, strong normalisation

1 The lambda calculus with constructors

1.1 Its syntax

The syntax of the λ -calculus with constructors [1] is defined from two disjoint sets of symbols: *variables* (notation: x, y, z , etc.) and *constructors* (notation: c, d , etc.) It consists of two syntactic categories defined by mutual induction in Fig. 1: terms (notation: t, u, s , etc.) and case binding (notation: θ, ϕ).

<i>Terms :</i>	$t, u, s \triangleq$	x		tu		$\lambda x.t$	(λ -calculus)
				c			(Constructor)
				$\{\theta\} \cdot t$			(Case Construct)
				\boxtimes			(Daimon)
<i>Case Bindings :</i>	$\theta, \phi \triangleq$	$\{c_1 \mapsto u_1; \dots c_n \mapsto u_n\}$					(Case Binding)
		$c_i \neq c_j \text{ for } i \neq j$					

Fig. 1. λ_c -terms and case bindings.

Terms include all the syntactic constructs of the λ -calculus, plus constructors (as constants) with a case construct (similar to the case construct of Pascal) to analyse them. There is also a constant \blackbox (the *Daimon*) representing immediate termination. Case bindings are finite functions from constructors to terms.

Free and bound (occurrences of) variables are defined as usual, taking care that constructors are not variables and thus not subject to α -conversion. The set of free variables (denoted by $\mathcal{FV}(t)$) is defined for the new constructs by

$$\mathcal{FV}(c) = \emptyset \quad \mathcal{FV}(\{\theta\} \cdot t) = \mathcal{FV}(\theta) \cup \mathcal{FV}(t) \quad \mathcal{FV}(\theta) = \cup_{(c \mapsto u) \in \theta} \mathcal{FV}(u)$$

The usual operation of substitution on terms (notation: $t[x := u]$) is defined as expected, taking care of renaming bound variables when needed in order to prevent variable capture. Substitution on case bindings (notation: $\theta[x := u]$) is defined componentwise.

1.2 Its operational semantics

The reduction of $\lambda_{\mathcal{C}}$ is based on the 9 reduction rules given in Fig. 2 among which one can find the β and η reduction rules of the λ -calculus, now called **APPLAM** and **LAMAPP**¹, respectively. Case constructs are propagated through terms via the **CASEAPP**, **CASELAM** and **CASECASE** commutation rules, and ultimately destructed with **CASECONS** reduction. For an explanation of the role and expressiveness of these rules, see [1].

The confluence or non confluence is known for every combination of the 9 reduction rules [1]. In this paper, we shall only consider the following subcalculi, that are all confluent:

- $\mathcal{B}_{\mathcal{C}}$ denotes the $\lambda_{\mathcal{C}}$ -calculus without **APPLAM** rule.
- $\mathcal{D}_{\mathcal{C}}$ is the $\lambda_{\mathcal{C}}$ -calculus without rules **APPDAI**, **LAMDAl** and **CASEDAI**.
- $\lambda_{\eta\blackbox}$ denotes $\lambda_{\mathcal{C}}$ -calculus restricted to rules **APPDAI**, **LAMAPP**, **LAMDAl** and **CASEDAI**.

A term that has no infinite reduction is said to be *strongly normalising*. By extension, a calculus is strongly normalising (notation: *SN*) when all its terms are. It is also known that $\mathcal{B}_{\mathcal{C}}$ is *SN*.

1.3 Values in $\lambda_{\mathcal{C}}$

In $\lambda_{\mathcal{C}}$, we call a *data structure* any term which is of the form $ct_1 \dots t_k$ where c is a constructor and t_1, \dots, t_k ($k \geq 0$) are arbitrary terms. We then call a *value* any term which is a λ -abstraction or a data structure.

We say that a term is *defined* when it has no subterm of the form $\{\theta\} \cdot c$, with $c \notin \text{dom}(\theta)$, and that it is *hereditarily defined* when all its reducts (in any number of steps) are defined. (Intuitively, terms that are not defined contain pattern matching failures and therefore will be rejected by the type system.)

¹ In $\lambda_{\mathcal{C}}$ -calculus, the name of each reduction rule consists of the names of the two constructions that interact for the reduction.

Beta-reduction			
APPLAM	(AL)	$(\lambda x.t)u \rightarrow t\{x \leftarrow u\}$	
APPDAI	(AD)	$\boxtimes u \rightarrow \boxtimes$	
Eta-reduction			
LAMAPP	(LA)	$\lambda x.tx \rightarrow t$	$(x \notin \mathcal{FV}(t))$
LAMDAI	(LD)	$\lambda x.\boxtimes \rightarrow \boxtimes$	
Case propagation			
CASECONS	(CO)	$\{\theta\}.c \rightarrow t$	$((c \mapsto t) \in \theta)$
CASEDAI	(CD)	$\{\theta\}.\boxtimes \rightarrow \boxtimes$	
CASEAPP	(CA)	$\{\theta\}.(tu) \rightarrow (\{\theta\}.t)u$	
CASELAM	(CL)	$\{\theta\}.\lambda x.t \rightarrow \lambda x.\{\theta\}.t$	$(x \notin \mathcal{FV}(\theta))$
Case conversion			
CASECASE	(CC)	$\{\theta\}.\{\phi\}.t \rightarrow \{\theta \circ \phi\}.t$	
		with $\theta \circ \{c_1 \mapsto t_1; \dots; c_n \mapsto t_n\} \equiv c_1 \mapsto \{\theta\}.t_1; \dots; c_n \mapsto \{\theta\}.t_n$	

Fig. 2. Reduction rules for λc .

Proposition 1 *Every defined closed normal term is either \boxtimes , either a value.*

Finally, a term that is both strongly normalising and hereditarily defined is said to be *perfectly normalising*.

2 Type system

2.1 An informal presentation

The type system that we want to define includes the simply typed λ -calculus: the main type construct is the arrow type $T \rightarrow U$ that comes with its usual introduction and elimination rules. To achieve polymorphism, we introduce type variables (written X, Y etc.) and universal type quantification (notation: $\forall X.T$). Instantiation is performed via a subtyping judgement that contains all the rules of system F with subtyping such as presented in [3].

To type-check data structures, we associate a type constant —still written c — to every constructor c . We introduce a *type application* DT for applied structures, so that we can derive $c \vec{t} : c \vec{T}$ from $\vec{t} : \vec{T}$ (see 2.2 for more details on vectorial notations). Nevertheless, the formation of application types has to be restricted to prevent typing non normalising terms. Indeed, unrestricted type application would allow to give the type TT to the term $\delta\delta$, were T is a type for δ ².

For that we distinguish a subclass of *data types* (notation: D, E) that will be the only types on the left-hand side of a type application. In practice this

² Such as $\forall X(X \rightarrow X) \rightarrow \forall X(X \rightarrow X)$.

subclass excludes arrow types and type variables (that could be instantiated by arbitrary types). To still keep the ability to quantify over data-types, we introduce *data type variables* (notation: α, β etc.) and data type quantification.

To encode algebraic types, we add union types. For example, we could define a type of natural numbers with the equation $\mathit{nat} \equiv 0 \cup \mathit{succ}(\mathit{nat})$ (where 0 and succ are constructors³). To distribute arrow among union, we also need intersection types: $(0 \cup \mathit{succ}(\mathit{nat})) \rightarrow T \equiv (0 \rightarrow T) \cap (\mathit{succ}(\mathit{nat}) \rightarrow T)$. This allows to type $\mathit{pred} := \lambda x. \{0 \mapsto 0; \mathit{succ} \mapsto \lambda y. y\} \cdot x$ with type $\mathit{nat} \rightarrow \mathit{nat}$ for instance. By symmetry, we add existential quantifier⁴.

2.2 The formal system

We define a polymorphic type system with union and intersection for both terms and case bindings of λ_C (Fig. 3). It uses two spaces of type variables: *ordinary type variables* and *data type variables*. There are also two kinds of types: namely, *data types* form a syntactic subclass of *ordinary types*.

<i>Types :</i>	$T, U :=$	X	(Ordinary type variable)
		$\alpha \mid c \mid DT$	(Data type)
		$T \rightarrow U$	(Arrow type)
		$T \cup U$	(Union type)
		$T \cap U$	(Intersection type)
		$\forall \alpha. T \mid \forall X. T$	(Universal type)
		$\exists \alpha. T \mid \exists X. T$	(Existential type)
 <i>DataTypes :</i>	 $D, E :=$	 α	 (Data type variable)
		$c \mid DT$	(Data structure)
		$D \cup E$	(Union data type)
		$D \cap E$	(Intersection data type)
		$\forall \alpha. D \mid \forall X. D$	(Universal data type)
		$\exists \alpha. D \mid \exists X. D$	(Existential data type)

Fig. 3. Types of λ_C .

In the following, ν denotes a variable that can be a type variable or a data type variable. We also use a vectorial notation for application or arrow types:

$$\begin{aligned} \vec{T} &:= [] \mid \vec{T}; T \\ c[] &= c \quad ; \quad c(\vec{T}; T) = (c\vec{T})T \\ [] \rightarrow U &= U \quad ; \quad (\vec{T}; T) \rightarrow U = \vec{T} \rightarrow (T \rightarrow U) \end{aligned}$$

³ This would require a fixpoint operator, or a double subtyping rule

⁴ The type system is anyhow not decidable. This paper does not deal with decidability but with correctness.

Typing rules (Fig 4) include the usual introduction and elimination rules of typed λ -calculus for each type operator. Some of them —like the elimination of universal quantifier— are indeed subtyping rules (Fig 5).

The subtyping rule **Data** allows typing constructors with arbitrary arity. A case binding is typed like a function waiting for a constructor of its domain as argument, up to a possible conversion of arrow type into application type: if u has type $T \rightarrow U$, then $\{c \mapsto u\}$ has type $c \rightarrow (T \rightarrow U)$ as well as $cT \rightarrow U$. This is the point that allows **CASEAPP** commutation rule to be well typed. In the same way, the typing rule for a case construct $\{\theta\} \cdot t$ (**case**) allows t to be a function that waits for an arbitrary numbers of argument. This make **CASELAM** well typed.

<p>Case Binding: If $\theta = \{(c_i \mapsto u_i)_{i=0}^n\}$</p> $\mathbf{CB} \frac{(\Gamma \vdash u_i : \vec{A}_i \rightarrow T_i)_{i=0}^n}{\Gamma \vdash \theta : c_{i_0} \vec{A}_{i_0} \rightarrow T_{i_0}}$
<p>Terms:</p> $\mathbf{Init} \frac{}{\Gamma \vdash x : A} (x : A \in \Gamma) \quad \mathbf{False} \frac{}{\Gamma \vdash \mathbf{F} : A} \quad \mathbf{Constr} \frac{}{\Gamma \vdash c : c}$ $\rightarrow \mathbf{intro} \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x. t : A \rightarrow B} \quad \rightarrow \mathbf{elim} \frac{\Gamma \vdash t : A \rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$ $\mathbf{case} \frac{\Gamma \vdash t : \vec{B} \rightarrow A \quad \Gamma \vdash \theta : A \rightarrow T}{\Gamma \vdash \{\theta\} \cdot t : \vec{B} \rightarrow T}$
<p>Shared rules: M is either a term t, either a case binding θ.</p> $\mathbf{Univ} \frac{\Gamma \vdash M : T}{\Gamma \vdash M : \forall \nu. T} \nu \notin \mathcal{TV}(\Gamma) \quad \mathbf{Inter} \frac{\Gamma \vdash M : T \quad \Gamma \vdash M : U}{\Gamma \vdash M : T \cap U}$ $\mathbf{Exist} \frac{\Gamma, x : T \vdash M : U}{\Gamma, x : \exists \nu. T \vdash M : U} \nu \notin \mathcal{TV}(U) \quad \mathbf{Union} \frac{\Gamma, x : T_1 \vdash M : U \quad \Gamma, x : T_2 \vdash M : U}{\Gamma, x : T_1 \cup T_2 \vdash M : U}$ $\mathbf{Subs} \frac{\Gamma \vdash M : T \quad T \preceq U}{\Gamma \vdash M : U}$

Fig. 4. Typing rules

In the following sections, we show that this type system satisfies strong normalisation. We get onto correctness question at section ??.

$\text{Refl} \frac{-}{T \preceq T} \qquad \text{Trans} \frac{T \preceq T_0 \quad T_0 \preceq T'}{T \preceq T'}$
$\text{Arrow} \frac{T' \preceq T \quad U \preceq U'}{T \rightarrow U \preceq T' \rightarrow U'} \qquad \text{App} \frac{D \preceq D' \quad T \preceq T'}{DT \preceq D'T'}$
$\text{UintroL} \frac{-}{U_1 \preceq U_1 \cup U_2} \qquad \text{UintroR} \frac{-}{U_2 \preceq U_1 \cup U_2} \qquad \text{Uelim} \frac{T_1 \preceq U \quad T_2 \preceq U}{T_1 \cup T_2 \preceq U}$
$\text{Nintro} \frac{T \preceq U_1 \quad T \preceq U_2}{T \preceq U_1 \cap U_2} \qquad \text{NelimL} \frac{-}{U_1 \cap U_2 \preceq U_1} \qquad \text{NelimR} \frac{-}{U_1 \cap U_2 \preceq U_2}$
$\text{Vintro} \frac{T \preceq U}{T \preceq \forall \nu. U} \quad \nu \notin \text{TV}(T) \qquad \text{Velim} \frac{-}{\forall X. T \preceq T\{X \leftarrow U\}} \qquad \text{VelimD} \frac{-}{\forall \alpha. T \preceq T\{\alpha \leftarrow D\}}$
$\text{Eintro} \frac{-}{T\{X \leftarrow U\} \preceq \exists X. T} \qquad \text{EintroD} \frac{-}{T\{\alpha \leftarrow D\} \preceq \exists \alpha. T} \qquad \text{Elim} \frac{U \preceq T}{\exists \nu. U \preceq T} \quad \nu \notin \text{TV}(T)$
$\text{Data} \frac{-}{D \preceq T \rightarrow DT} \qquad \text{Constr} \frac{-}{c_1 \vec{T} \cap c_2 \vec{U} \preceq \forall \alpha. \alpha} \quad c_1 \neq c_2$
$\text{App}/\cap \frac{-}{D_1 T_1 \cap D_2 T_2 \preceq (D_1 \cap D_2)(T_1 \cap T_2)} \qquad \text{App}/\forall \frac{-}{\forall \nu. (DT) \preceq (\forall \nu. D)(\forall \nu. T)}$
$\rightarrow/\cap \frac{-}{(T_1 \rightarrow U_1) \cap (T_2 \rightarrow U_2) \preceq (T_1 \cap T_2) \rightarrow (U_1 \cap U_2)} \qquad \rightarrow/\forall \frac{-}{\forall \nu. (T \rightarrow U) \preceq (\forall \nu. T) \rightarrow (\forall \nu. U)}$
$\rightarrow/\cup \frac{-}{(T_1 \rightarrow U_1) \cap (T_2 \rightarrow U_2) \preceq (T_1 \cup T_2) \rightarrow (U_1 \cup U_2)} \qquad \rightarrow/\exists \frac{-}{\forall \nu. (T \rightarrow U) \preceq (\exists \nu. T) \rightarrow (\exists \nu. U)}$
$\text{U-AppR} \frac{-}{D(T_1 \cup T_2) \preceq DT_1 \cup DT_2} \qquad \text{U-AppL} \frac{-}{(D_1 \cup D_2)T \preceq D_1 T \cup D_2 T}$
$\text{E-AppR} \frac{-}{D(\exists \nu. T) \preceq \exists \nu. DT} \quad \nu \notin \text{TV}(D) \qquad \text{E-AppL} \frac{-}{(\exists \nu. D)T \preceq \exists \nu. DT} \quad \nu \notin \text{TV}(T)$
$\text{U-}\forall \frac{-}{\forall \nu. (T \cup U) \preceq (\forall \nu. T) \cup U} \quad \nu \notin \text{TV}(U) \qquad \text{E-}\cap \frac{-}{(\exists \nu. T) \cap U \preceq \exists \nu. (T \cap U)} \quad \nu \notin \text{TV}(U)$

Fig. 5. Sub-typing rules.

3 Case normal forms calculus

In this part, we consider λ_C -calculus up to *case rules* (CASECONS, CASEDAI, CASEAPP, CASELAM and CASECASE). Namely, we define a new calculus —the λ_{CN} -calculus— in which terms are in case-normal form, in order to project λ_C in it later. We write A^{CN} the set of λ_{CN} -terms and A_0^{CN} its subset of closed terms.

3.1 The λ_{CN} -calculus

The grammar of λ_{CN} (Fig. 6) differs of the one of λ_C by case constructs, so that λ_{CN} terms do not have case redex. In front of a case binding, one can only find an unmatched constructor or a variable. Since such a variable can be substituted by any term in A^{CN} , we need to refine substitution. For that, we use the notation of *pseudo case constructs* $\{\theta\} \cdot t$ for any $t \in A^{CN}$, that we define inductively:

$$\begin{aligned} \{\theta\} \cdot x &= \{\theta\} \cdot x & \{\theta\} \cdot c &= \{\theta\} \cdot c & (\text{if } c \in \text{dom}(\theta)) \\ \{\theta\} \cdot \mathbf{X} &= \mathbf{X} & \{\theta\} \cdot c &= u & (\text{if } c \mapsto u \in \theta) \\ \{\theta\} \cdot (tu) &= (\{\theta\} \cdot t) u & \{\theta\} \cdot (\lambda x.t) &= \lambda x. \{\theta\} \cdot t & (\text{with } x \notin \mathcal{FV}(\theta)) \\ \{\theta\} \cdot (\{\phi\} \cdot t) &= \{\theta\tilde{\phi}\} \cdot t \\ \text{where } \theta\tilde{\phi}\{c_1 \mapsto u_1; \dots; c_n \mapsto u_n\} &= \{c_1 \mapsto \{\theta\} \cdot u_1; \dots; c_n \mapsto \{\theta\} \cdot u_n\} \end{aligned}$$

Note that $\{\theta\tilde{\phi}\} \cdot t$ is well defined since $\text{dom}(\theta\tilde{\phi}) = \text{dom}(\phi)$.

Thus we can define the substitution in λ_{CN} , that we write $t\langle u/x \rangle$:

$$\begin{aligned} x\langle u/x \rangle &= u & (\{\theta\} \cdot x)\langle u/x \rangle &= \{\theta\langle u/x \rangle\} \cdot u \\ y\langle u/x \rangle &= y & (\{\theta\} \cdot y)\langle u/x \rangle &= \{\theta\langle u/x \rangle\} \cdot y \\ c\langle u/x \rangle &= c & (\{\theta\} \cdot c)\langle u/x \rangle &= \{\theta\langle u/x \rangle\} \cdot c \\ (\lambda y.t)\langle u/x \rangle &= \lambda y.(t\langle u/x \rangle) & (t_1 t_2)\langle u/x \rangle &= t_1\langle u/x \rangle t_2\langle u/x \rangle \\ \mathbf{X}\langle u/x \rangle &= \mathbf{X} \end{aligned}$$

where $\{c_1 \mapsto t_1; \dots; c_k \mapsto t_k\}\langle u/x \rangle = \{c_1 \mapsto t_1\langle u/x \rangle; \dots; c_k \mapsto t_k\langle u/x \rangle\}$

If a case redex appears when a substitution is performed, it is immediately destructed due to the definition of pseudo case construct. That is why there is no explicit case rule (Fig 6). Although pseudo case constructs and CN-substitution can perform many λ_C -reduction steps at a time, their manipulation remain similar to case constructs an substitution in λ_C :

Lemma 2 For any $t, t', u \in A^{CN}$ and any λ_{CN} -case bindings θ, ϕ ,

$$(\{\theta\} \cdot t)\langle u/x \rangle = \{\theta\langle u/x \rangle\} \cdot t\langle u/x \rangle \quad (1)$$

$$\{\theta\} \cdot (\{\phi\} \cdot t) = \{\theta\tilde{\phi}\} \cdot t \quad (2)$$

$$u\langle t/x \rangle \langle t'/y \rangle = u\langle t'/y \rangle \langle t\langle t'/y \rangle/x \rangle \quad (3)$$

$$t \xrightarrow{CN} t' \Rightarrow t\langle u/x \rangle \xrightarrow{CN}^* t'\langle u/x \rangle \quad (4)$$

Values and *defined* terms are defined in the same way as in λ_C -calculus. We write \mathcal{V} the set of λ_{CN} -values, and PN_0 the set of perfectly normalising closed λ_{CN} -terms. Note that proposition 1 also holds in λ_{CN} .

Terms:	$t, u, s \triangleq x \mid c \mid \boxtimes \mid tu \mid \lambda x.t \mid \{\theta\} \cdot x \mid \{\theta\} \cdot c$ (with $c \notin \text{dom}(\theta)$)
Case Bindings:	$\theta, \phi \triangleq \{c_1 \mapsto u_1; \dots; c_n \mapsto u_n\}$
Reduction rules:	$\beta \quad (\lambda x.t)u \xrightarrow{CN} t\langle u/x \rangle$ $\beta_d \quad \boxtimes u \xrightarrow{CN} \boxtimes$ $\eta \quad \lambda x.tx \xrightarrow{CN} t \quad (x \notin \mathcal{FV}(t))$ $\eta_d \quad \lambda x.\boxtimes \xrightarrow{CN} \boxtimes$

Fig. 6. The λ_{CN} -calculus.

3.2 Projection of λ_c in λ_{CN}

Terms of λ_{CN} are included in Λ . Reciprocally, we can project every λ_c -term t in Λ^{CN} through a translation in case-normal form called *switch* (notation: $\downarrow t$):

$$\begin{aligned} \downarrow x &= x & \downarrow (\lambda x.t) &= \lambda x. \downarrow t \\ \downarrow c &= c & \downarrow (t_1 t_2) &= \downarrow t_1 \downarrow t_2 \\ \downarrow \boxtimes &= \boxtimes & \downarrow (\{\theta\} \cdot t) &= \{\downarrow \theta\} \cdot (\downarrow t) \\ \text{where } \downarrow \{c_1 \mapsto t_1; \dots; c_k \mapsto t_k\} &= \{c_1 \mapsto \downarrow t_1; \dots; c_k \mapsto \downarrow t_k\} \end{aligned}$$

This translation preserves composition, substitution and reduction:

Lemma 3 For any $t, t', u \in \Lambda$ and any case bindings θ, ϕ ,

$$\downarrow (\theta \circ \phi) = (\downarrow \theta \circ \downarrow \phi) \tag{5}$$

$$\downarrow (t[u := x]) = (\downarrow t)\langle \downarrow u/x \rangle \tag{6}$$

$$t \rightarrow t' \Rightarrow \downarrow t \xrightarrow{CN}^* \downarrow t' \tag{7}$$

4 Reducibility Candidates Model

In this section, we will interpret types by *reducibility candidates* [2], that are sets of closed and perfectly normalising λ_{CN} -terms. We refine the usual definition with the notion of *data candidates*.

4.1 Introduction of Reducibility candidates

The definition of reducibility candidates is founded on the notion of neutral terms: a term is said to be *neutral* when it is not a value. We write \mathcal{N}_D the set of defined closed neutral λ_{CN} -terms. A set $S \in \Lambda_0^{CN}$ is a reducibility candidate when it satisfies the following conditions:

- (CR1) Perfect normalisation: $S \subseteq PN_0$
- (CR2) Stability by reduction: $t \in S \Rightarrow \text{Red}_1(t) \subseteq S$
- (CR3) Stability by neutral expansion: $t \in \mathcal{N}_D, \text{Red}_1(t) \subseteq S \Rightarrow t \in S$

We denote by \mathcal{CR} the set of all reducibility candidates, and by **(CR)** the conjunction of **(CR1)**, **(CR2)** and **(CR3)**. Sets that satisfy **(CR1)** and **(CR2)** are called *pre-candidate of reducibility*, and we write \mathcal{PCR} the set of all such pre-candidates. Notice that every reducibility candidates is non empty, since it contains \mathfrak{X} as neutral term, and that PN_0 is in \mathcal{CR} .

We distinguish the subclass of candidates whose the only values are data-structures, and we call them *data candidates*. As suggested by their name, they will be helpful to interpret data types. We write \mathcal{DC} the set of all data candidates.

4.2 Closure properties

A reducibility candidate is closed by reduction and by expansion for neutral terms. As a consequence, it is entirely determined by its values. In the following, we note $Red_n(t)$ the set of terms to which t reduces in n steps. $Red_*(t)$ is the union of all these sets for n in \mathbb{N} .

Lemma 4 *Let S and S' be two reducibility candidates.*

Then $S \cap \mathcal{V} \subseteq S' \cap \mathcal{V}$ iff $S \subseteq S'$, and so $S \cap \mathcal{V} = S' \cap \mathcal{V}$ implies $S = S'$.

This property will allow us to define a data candidate by the set of its values. For that, we will close ordinary sets of terms by **(CR)**, following the presentation of [4]. Given a set X of closed $\lambda_{\mathbf{CN}}$ -terms, we define:

- $X_0 = Red_*(X)$
- $X_{n+1} = X_n \cup \{t \in \mathcal{N}_D / Red_1(t) \subseteq X_n\}$ for every $n \in \mathbb{N}$
- $\overline{X} = \bigcup_{n \in \mathbb{N}} X_n$.

When $X \subseteq PN_0$, we call \overline{X} the *closure* of X , whatis justified by:

Lemma 5 *If $S \subseteq PN_0$, then \overline{S} is the smallest reducibility candidate containing S .*

Lemma 4 says that we can define a reducibility candidate with its values. Now the closure operator allows u to consider a candidate *generated* by a set of values.

Definition 1 *Given $S \subseteq \Lambda_0^{CN}$, $Val(S)$ denotes the intersection of \mathcal{V} and $Red_*(S)$. In particular, if $S \in \mathcal{CR}$ then $Val(S) = S \cap \mathcal{V}$.*

Since all the values of a closed set \overline{S} are already in $S_0 = Red_*(S)$, the following lemma holds:

Lemma 6 (Closure in \mathcal{DC}) .

Let $\mathcal{D} \subseteq \Lambda_0^{CN}$ a set of terms such that $Val(\mathcal{D}) \subseteq \{cu_1 \dots u_n / u_i \in \Lambda_0^{CN}\}$. Then $Val(\overline{\mathcal{D}}) \subseteq \{cu_1 \dots u_n / u_i \in \Lambda_0^{CN}\}$ and $\overline{\mathcal{D}}$ is a data candidate.

This shows that we can also use the closure to define *data candidates*.

Remark 1 *The closure operator permits the construction of the smallest reducibility candidate $\overline{\emptyset}$. Lemma 6 ensures that it is a data candidate.*

4.3 Operations in \mathcal{CR}

Recording that we aim to interpret types by reducibility candidates, we need to define \rightarrow , \cap , \cup , \forall and \exists in \mathcal{CR} . The definitions of \rightarrow and \cap are usual. The stability by union is more tricky, and the application is defined by a closure.

Arrow

Definition 2 (Arrow)

Let \mathcal{A} and \mathcal{B} be two subsets of Λ_0^{CN} . Then

$$\mathcal{A} \rightarrow \mathcal{B} \triangleq \{t \mid \forall u \in \mathcal{A}, tu \in \mathcal{B}\}$$

Proposition 7 If \mathcal{A} and \mathcal{B} are in \mathcal{CR} , so is $\mathcal{A} \rightarrow \mathcal{B}$.

Remark 2 It is sufficient to assume $\mathcal{A} \in \mathcal{PCR}$ and $\mathcal{B} \in \mathcal{CR}$.

Intersection It is well known that Reducibility candidates are stable by intersection. Data-candidates are as well.

Proposition 8 (Intersection)

Reducibility candidates trivially satisfy the following property:

$$\mathcal{A}, \mathcal{B} \in \mathcal{CR} \quad \Rightarrow \quad \mathcal{A} \cap \mathcal{B} \in \mathcal{CR}$$

Moreover, since the equality $\mathcal{V}al(\mathcal{A} \cap \mathcal{B}) = \mathcal{V}al(\mathcal{A}) \cap \mathcal{V}al(\mathcal{B})$ holds for any reducibility candidate \mathcal{A}, \mathcal{B} , stability is also satisfied by data-candidates:

$$\mathcal{A}, \mathcal{B} \in \mathcal{DC} \quad \Rightarrow \quad \mathcal{A} \cap \mathcal{B} \in \mathcal{DC}$$

Union The stability by union of reducibility candidates has long been conjectured as false, and it is in general. Riba proved that a necessary and sufficient condition is that perfectly normalising neutral terms have a *principal reduct* ([4], Cor. 4.12).

Definition 3 (Principal reduct)

Let $t, u \in \Lambda^{CN}$. u is a principal reduct for t when $t \xrightarrow{CN} u$, and

$$\text{for all } v \in \mathcal{V}, \quad t \xrightarrow{CN}^* v \quad \Rightarrow \quad u \xrightarrow{CN}^* v$$

Lemma 9 If $t \in \mathcal{N}_D \cap \mathcal{PN}_0$, then t has a principal reduct.

Corrolary 10 \mathcal{CR} is stable by union

Data application

Definition 4 (Set application) .

Let S and R be two sets of terms. Then we note

$$SR \triangleq \{tu / t \in S \text{ and } u \in R\}$$

Definition 5 (Candidate application) .

For $\mathcal{D} \in \mathcal{DC}$ and $\mathcal{A} \in \mathcal{CR}$, we define

$$\mathcal{D} \cdot \mathcal{A} \triangleq \overline{\mathcal{D}\mathcal{A}}$$

By definition, $\mathcal{D} \cdot \mathcal{A}$ is in \mathcal{CR} . We can even say that it is a *data-candidate*:

Lemma 11 For any $\mathcal{D} \in \mathcal{DC}$ and $\mathcal{A} \in \mathcal{CR}$,

$$\text{val}(\mathcal{D} \cdot \mathcal{A}) = \text{val}(\mathcal{D})\mathcal{A}. \quad (8)$$

As a corrolary, $\mathcal{D} \cdot \mathcal{A} \in \mathcal{DC}$

4.4 Modelling types

Now we have all tools we need to interpret λ_c -types. Since the system includes type variables, they will be interpreted by a reducibility candidate by a *valuation*, ie by a function ρ that matches every data-type variable to a data-candidate, and every type variable to a reducibility candidate.

Definition 6 (Interpretation of types) .

Given a valuation ρ , a type A such that $\mathcal{TV}(A) \subseteq \text{dom}(\rho)$, the interpretation $[A]_\rho$ of A in ρ is the reducibility candidate inductively defined by:

- $[\alpha]_\rho = \rho(\alpha)$
- $[X]_\rho = \rho(X)$
- $[c]_\rho = \overline{\{c\}}$
- $[DT]_\rho = [D]_\rho \cdot [T]_\rho$
- $[A \rightarrow B]_\rho = [A]_\rho \rightarrow [B]_\rho$
- $[A \cap B]_\rho = [A]_\rho \cap [B]_\rho$
- $[\forall \alpha. B]_\rho = \bigcap_{\mathcal{A} \in \mathcal{DC}} [B]_{\rho, \alpha \mapsto \mathcal{A}}$
- $[\forall X. B]_\rho = \bigcap_{\mathcal{A} \in \mathcal{CR}} [B]_{\rho, X \mapsto \mathcal{A}}$
- $[A \cup B]_\rho = [A]_\rho \cup [B]_\rho$
- $[\exists \alpha. B]_\rho = \bigcup_{\mathcal{A} \in \mathcal{DC}} [B]_{\rho, \alpha \mapsto \mathcal{A}}$
- $[\exists X. B]_\rho = \bigcup_{\mathcal{A} \in \mathcal{CR}} [B]_{\rho, X \mapsto \mathcal{A}}$

We check that this interpretation is well defined from types to reducibility candidates, using the properties of \rightarrow, \cdot, \cap and \cup in \mathcal{CR} and \mathcal{DC} :

Proposition 12 For any valuation ρ , any type T and any data-type D ,

$$[T]_\rho \in \mathcal{CR} \text{ and } [D]_\rho \in \mathcal{DC}.$$

This definition of type interpretation gives a very precise notion of data-types:

Lemma 13 *A direct corollary of lemma 11 is that*

$$\text{Val}([cT_1 \dots T_k]_\rho) = \{ct_1 \dots t_n / t_i \in [T_i]_\rho\}.$$

In particular, proposition 1 ensures that $t \in [cT_1 \dots T_k]_\rho$ implies $t \xrightarrow{CN}^ ct_1 \dots t_n$ with $t_i \in [T_i]_\rho$ for all $i \leq n$, or $t \xrightarrow{CN}^* \boxtimes$*

Furthermore, substituting a type variable by a type A amounts to the same reducibility candidate as evaluating this type variable by $[A]$:

Lemma 14 *For any types T, U and any data-type D and any valuation ρ ,*

$$[T\{X \leftarrow U\}]_\rho = [T]_{\rho, X \mapsto [U]_\rho} \quad \text{and} \quad [T\{\alpha \leftarrow D\}]_\rho = [T]_{\rho, \alpha \mapsto [D]_\rho}$$

To derive a type judgement, we may need to type a case binding. So we define the interpretation of case binding types.

Definition 7 (Interpretation of case binding types) .

The interpretation of T , seen as a type for case bindings, in the valuation ρ is noted $\llbracket T \rrbracket_\rho$. It is defined by

$$\llbracket T \rrbracket_\rho \triangleq \{\theta / \lambda x. \{\theta\} \cdot x \in [T]_\rho\}.$$

4.5 Soundness w.r.t. sub-typing

Lemma 15 (Interpretation preserves sub-typing) .

If the sub-typing judgement $T_1 \preceq T_2$ is derivable, then for any valuation ρ such that $\mathcal{TV}(T_1, T_2) \subseteq \text{dom}(\rho)$, $[T_1]_\rho \subseteq [T_2]_\rho$.

5 Strong normalisation

In this part, we will prove the adequacy of our model: if a term has type T , then it is in its interpretation. As a consequence, it also means that it is a perfectly normalising term –since the interpretation of a type is a subset of PN_0 .

5.1 Judgements

Reducibility candidates model deals with closed terms. Nevertheless, proving adequacy lemma by induction requires the use of open terms, with some assumptions on their free variables –namely assumptions that will be guaranteed by a context.

Therefore we use substitutions to close terms, and we extend the interpretation of terms with judgements.

Definition 8 (Closing terms) .

A substitution is given by the following grammar:

$$\sigma := \emptyset \mid (x \mapsto u; \sigma) \quad (u \in \Lambda_0^{CN})$$

Its domain is defined by

$$\text{dom}(\emptyset) = \emptyset. \quad \text{dom}(x \mapsto u; \sigma) = \{x\} \cup \text{dom}(\sigma).$$

The closure t_σ of a term $t \in \Lambda^{CN}$ by a substitution σ such that $\mathcal{FV}(t) \subseteq \text{dom}(\sigma)$ is the closed term defined by:

$$t_\emptyset = t; \quad t_{x \mapsto u; \sigma} = t\langle u/x \rangle_\sigma.$$

We close case bindings in the same way.

Definition 9 (Context satisfied by a substitution) .

Let $\Gamma = x_1 : A_1 \dots x_k : A_k$ be a context and ρ a valuation that evaluates all type variables of Γ . We say that a substitution σ satisfies Γ in ρ (noted $\sigma \in [\Gamma]_\rho$) when for all i , $(x_i)_\sigma \in [A_i]_\rho$.

Definition 10 (Valid judgement) .

Given $\Gamma \vdash t : T$ a judgement, we say that it is valid (noted $\Gamma \vDash t : T$) if for every valuation ρ that interprets $\mathcal{TV}(\Gamma, T)$, and every substitution $\sigma \in [\Gamma]_\rho$,

$$\downarrow t_\sigma \in [T]_\rho.$$

We naturally extend this definition to judgements that type a case binding:
 $\Gamma \vDash \theta : T$ when $\downarrow \theta_\sigma \in \llbracket \top \rrbracket_\rho$ for every valuation ρ and every $\sigma \in [\Gamma]_\rho$

5.2 Some lemmas

First we show some kinds of inversion lemmas for reducibility candidates.

Recall that, if u is perfectly normalising, then $\text{Red}_*(u)$ is a pre-candidate, and thus $\text{Red}_*(u) \rightarrow \mathcal{A} \in \mathcal{CR}$ for all $\mathcal{A} \in \mathcal{CR}$ (remark 2).

Lemma 16 (Inversion of application and abstraction in CR) .

For every $\mathcal{A} \in \mathcal{CR}$, every $t \in \Lambda_0^{CN}$, and every $u \in \text{PN}_0$,

$$tu \in \mathcal{A} \iff t \in \text{Red}_*(u) \rightarrow \mathcal{A} \tag{9}$$

For every $\mathcal{A} \in \mathcal{CR}$, every $\mathcal{P} \in \mathcal{PCR}$, and every $t \in \Lambda^{CN}$,

$$\lambda x.t \in \mathcal{P} \rightarrow \mathcal{A} \iff \text{for all } v \in \mathcal{P}, t\langle v/x \rangle \in \mathcal{A} \tag{10}$$

5.3 Adequacy lemma

Lemma 17 For any term t , any case binding θ , any context Γ and any type T ,

$$\begin{aligned} \Gamma \vdash t : T &\implies \Gamma \vDash t : T \\ \text{and } \Gamma \vdash \theta : T &\implies \Gamma \vDash \theta : T \end{aligned}$$

Corrolary 18 Every well typed λ_C -term has a case-normal form perfectly normalising.

5.4 Strong normalisation of λ_C

It is known that λ_C -calculus without β -reduction (noted \mathcal{B}_C) is strongly normalising ([1], Prop. 2). Nevertheless, the strong normalisation of case-normal forms is not sufficient to conclude the strong normalisation property for the typed λ_C -calculus. Indeed, a non terminating sub-term could be hidden in an unmatched branch of a case-binding that disappears in the CN-form, like in the term $\{\!|c \mapsto I ; d \mapsto \Omega|\!\} \cdot c$, whose CN-form is I .

In order to keep potential infinite derivation, we first expand λ_C -terms (in a way that avoids **CaseCons** and **CaseDai** redexes), before taking their case-normal form.

Definition 11 (*-expansion)

Given $t \in \Lambda$, t^* is the λ_C -term defined inductively by:

$$\begin{aligned} x^* &= x & c^* &= I c & \mathfrak{X}^* &= I \mathfrak{X} & (tu)^* &= t^* u^* & (\lambda x.t)^* &= \lambda x.t^* \\ (\{\!\|\theta\|\!\} \cdot t)^* &= \{\!\|\theta^*\!\!\} \cdot t^* & (c_i \mapsto u_i)_{i=1}^n{}^* &= (c_i \mapsto u_i^*)_{i=1}^n \end{aligned}$$

It is straightforward to check that this translation commutes with substitution and is preserved by reduction. It also preserves typing:

Lemma 19 For any term t , any type T and any context Γ ,

$$\Gamma \vdash t : T \quad \Rightarrow \quad \Gamma \vdash t^* : T$$

Now we combine both $*$ and \downarrow to translate typed λ_C -terms. We use the convention $\downarrow t^* = \downarrow (t^*)$.

Proposition 20 (\downarrow -* preserves β -redexes)

Let $t, t' \in \Lambda$ such that $t \rightarrow_R t'$, with $R \in \{\mathbf{AL}, \mathbf{LA}, \mathbf{CC}\}$. Then

$$\downarrow t^* \xrightarrow[CN]{+} \downarrow t'^*$$

This proposition does not hold for **CaseApp**, **CaseLam**, and **CaseCase** reductions:

Lemma 21 For any $t, t' \in \Lambda$ such that $t \rightarrow_R t'$, with $R \in \{\mathbf{CA}, \mathbf{CL}, \mathbf{CC}\}$,

$$\downarrow t^* \quad \neq \quad \downarrow t'^*$$

Now we can conclude the strong normalisation of non-daimon reduction from the strong normalisation property of \mathcal{B}_C .

Definition 12 (\mathcal{D}_C -calculus)

We note $\xrightarrow[\mathfrak{X}]{}_{\downarrow}$ the reduction relation restricted to $\{\mathbf{CaseDai}, \mathbf{LamDai}, \mathbf{CaseDai}\}$, and \mathcal{D}_C the calculus obtained from λ_C by removing these rules.

Definition 13 Given $t \in \Lambda$, we note

- $|\downarrow t^*|$ the size of the maximal CN-reduction of $\downarrow t^*$.
- $\mu(t)$ the size of the maximal \mathcal{B}_C -reduction of t .
- $\mathfrak{s}(t)$ the couple $(|\downarrow t^*|, \mu(t))$.

Lemma 22 *If $t \in \Lambda$ is typable, then $\mathfrak{s}(t)$ is finite and for every t ,*

$$t \xrightarrow[\mathcal{D}_C]{} t' \quad \Rightarrow \quad \mathfrak{s}(t) < \mathfrak{s}(t')$$

for the lexical order.

Corrolary 23 *Every typable term is strongly normalising for \mathcal{D}_C .*

Lemma 24 *If a term is strongly normalising for \mathcal{D}_C , then it also is for λ_C .*

And finally,

Corrolary 25 *Typed λ_C -calculus enjoys strong normalisation property.*

References

1. Ariel Arbiser, Alexandre Miquel, and Alejandro Ríos. A lambda-calculus with constructors. In *RTA*, volume 4098 of *Lecture Notes in Computer Science*, pages 181–196. Springer, 2006.
2. J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1989.
3. John C. Mitchell. Polymorphic type inference and containment. *Inf. Comput.*, 76(2/3):211–249, 1988.
4. Colin Riba. On the stability by union of reducibility candidates. In *FoSSaCS*, volume 4423 of *Lecture Notes in Computer Science*, pages 317–331. Springer, 2007.