A type-based termination criterion for dependently-typed higher-order rewrite systems

Frédéric Blanqui

LORIA & INRIA
615 rue du Jardin Botanique, BP 101, 54602 Villers-lès-Nancy, France
http://www.loria.fr/~blanqui/ - blanqui@loria.fr

Abstract. Several authors devised type-based termination criteria for ML-like languages allowing non-structural recursive calls. We extend these works to general rewriting and dependent types, hence providing a powerful termination criterion for the combination of rewriting and β-reduction in the Calculus of Constructions.

1 Introduction

The Calculus of Constructions (CC) [13] is a powerful type system allowing polymorphic and dependent types. It is the basis of several proof assistants (Coq, Lego, Agda, ...) since it allows one to formalize the proofs of higher-order logic. In this context, it is essential to allow users to define functions and predicates in the most convenient way and to be able to decide whether a term is a proof of some proposition, and whether two terms/propositions are equivalent w.r.t. user definitions. As exemplified in [16,10], a promising approach is rewriting. To this end, we need powerful criteria to check the termination of higher-order rewrite-based definitions combined with β-reduction.

In [10], we proved that such a combination is strongly normalizing if, on the one hand, first-order rewrite rules are strongly normalizing and non-duplicating and, on the other hand, higher-order rewrite rules satisfy a termination criterion based on the notion of computability closure and similar to primitive recursion. However, many rewrite systems do not satisfy these conditions, as division[1] on natural numbers *nat* for instance:

$$\begin{array}{lrcl}
(1) & -\ x\ 0 & \rightarrow & x \\
(2) & -\ 0\ x & \rightarrow & 0 \\
(3) & -\ (sx)\ (sy) & \rightarrow & -\ x\ y \\[4pt]
(4) & /\ 0\ x & \rightarrow & 0 \\
(5) & /\ (sx)\ y & \rightarrow & s\ (/\ (-\ x\ y)\ y)
\end{array}$$

Hughes *et al* [20], Xi [26], Giménez *et al* [18,5] and Abel [2] devised termination criteria able to treat such examples by exploiting the way inductive types are usually interpreted [23]. Take for instance the addition on Brouwer's ordinals *ord* whose constructors are $0 : ord$, $s : ord \Rightarrow ord$ and $lim : (nat \Rightarrow ord) \Rightarrow ord$:

[1] $(/\ x\ y)$ is the lower integer part of $\frac{x}{y+1}$.

$$\begin{array}{lrcl}
(1) & +\ 0\ x & \rightarrow & x \\
(2) & +\ (sx)\ y & \rightarrow & s\ (+\ x\ y) \\
(3) & +\ (lim\ f)\ y & \rightarrow & lim\ ([x : nat](+\ (f\ x)\ y))
\end{array}$$

The usual computability-based technique for proving the termination of this function is to interpret ord as the fixpoint of the following monotone function φ on the powerset of the set of strongly normalizing terms $\mathcal{SN}$ ordered by inclusion:

$$\varphi(X) = \{t \in \mathcal{SN} \mid t \rightarrow^* su \Rightarrow u \in X; t \rightarrow^* limf \Rightarrow \forall u \in \mathcal{SN}, fu \in X\}$$

The fixpoint of φ, $[\![ord]\!]$, can be reached by transfinite iteration and every $t \in [\![ord]\!]$ is obtained after a smallest ordinal $o(t)$ of iterations, the order of t. This naturally defines an ordering: $t > u$ iff $o(t) > o(u)$, with which $lim\ f > fu$ for all $u \in \mathcal{SN}$.

Now, applying this technique to nat, we can easily check that $o(-tu) \leq o(t)$ and thus allow the recursive call with $-xy$ in the definition of $/$ above. We proceed by induction on $o(t)$, knowing that $-tu$ is computable (*i.e.* belongs to $[\![nat]\!]$) iff all its reducts are computable:

- If $-tu$ matches rule (1) then $o(-tu) = o(t)$.
- If $-tu$ matches rule (2) then $o(-tu) = 0 \leq o(t)$.
- If $-tu$ matches rule (3) then $t = st'$ and $u = su'$. By induction hypothesis, $o(-t'u') \leq o(t')$. Thus, $o(-tu) = 1 + o(-t'u') \leq 1 + o(t') = o(t)$.
- If $-tu$ matches no rule then $o(-tu) = 0 \leq o(t)$.

The idea of the previously cited authors is to add that size/index/stage information to the syntax in order to prove this automatically. Instead of a single type nat, they consider a family of types $\{nat^{\mathfrak{a}}\}_{\mathfrak{a}\in\omega}$ (higher-order types require ordinals bigger than ω), each type $nat^{\mathfrak{a}}$ being interpreted by the set obtained after $\mathfrak{a}$ iterations of the function φ for nat. For first-order data types, $\mathfrak{a}$ can be seen as the maximal number of constructors at the top of a term. Finally, they define a decidable type system in which $-$ (defined by *fixpoint/cases* constructions in their work) can be typed by $nat^{\alpha} \Rightarrow nat^{\beta} \Rightarrow nat^{\alpha}$, where α and β are size variables, meaning that the order of $-tu$ is not greater than the order of t.

This can also be interpreted as a way to automatically prove theorems on the size of the result of a function w.r.t. the size of its arguments with applications to complexity and resource bound certification, and compilation optimization (*e.g.* array bound checks elimination and vector-based memoisation).

In this paper, we extend this technique to the full Calculus of Algebraic Constructions [10] whose type conversion rule depends on user definitions, and to general rewrite-based definitions (including rewriting modulo equational theories treated elsewhere [7]) instead of definitions only based on $fixpoint/cases$ constructions. However, several questions remain unanswered (*e.g.* subject reduction, matching on defined symbol, type inference) and left for future work.

We allow a richer size algebra than the one in [20,5,2] but we do not allow existential size variables and do not take into account conditionals as it can be done in Xi's work [26]. Note however that Xi is interested in the call-by-value normalization of closed simply-typed λ-terms with first-order data types, while we are interested in the strong normalization of the open terms of CAC.

The paper is organized as follows. Section 2 introduces the Calculus of Algebraic Constructions with Size Annotations (CACSA). Section 3 presents the termination criterion with some examples. Section 4 gives some elements of the termination proof (more details can be found in [9]). Finally, Section 5 proposes an extension (whose justification is ongoing) for capturing more definitions.

2 The Calculus of Algebraic Constructions with Size Annotations

CC is the full Pure Type System with set of *sorts* $\mathcal{S} = \{\star, \Box\}$ and axiom $\star : \Box$ [4]. The sort $\star$ is intended to be the universe of types and propositions, while $\Box$ is intended to be the universe of predicate types. Let $\mathcal{X}$ be the set of variables.

The Calculus of Algebraic Constructions (CAC) [10] is an extension of CC with a set $\mathcal{F}$ of function or predicate *symbols* defined by a set $\mathcal{R}$ of (higher-order) rewrite rules [15,22] whose left hand-sides are built from symbols and variables only. Every $x \in \mathcal{X} \cup \mathcal{F}$ is equipped with a sort s_x. We denote by $\mathcal{DF}$ the set of *defined* symbols, that is, the set of symbols f with a rule $f\boldsymbol{l} \rightarrow r \in \mathcal{R}$, and by $\mathcal{CF}$ the set $\mathcal{F} \setminus \mathcal{DF}$ of *constant* symbols. We add a superscript s to restrict these sets to objects of sort s.

Now, we assume given a first-order term algebra $\mathcal{A} = T(\mathcal{H}, \mathcal{Z})$, called the algebra of *size expressions*, built from a set $\mathcal{H}$ of *size symbols* of fixed arity and a set $\mathcal{Z}$ of *size variables*. Let $\mathcal{V}(t)$ be the set of size variables occurring in a term t. We assume that $\mathcal{H} \cap \mathcal{F} = \mathcal{Z} \cap \mathcal{X} = \emptyset$, $T(\mathcal{H}, \emptyset) \neq \emptyset$ and $\mathcal{A}$ is equipped with a quasi-ordering $\leq_{\mathcal{A}}$ stable by size substitution (if $a \leq_{\mathcal{A}} b$ then, for all size substitution φ, $a\varphi \leq_{\mathcal{A}} b\varphi$) such that $(\mathcal{A}, \leq_{\mathcal{A}})$ has a well-founded model $(\mathfrak{A}, \leq_{\mathfrak{A}})$:

Definition 1 (Size model). *A* pre-model *of $\mathcal{A}$ is given by a set $\mathfrak{A}$, an ordering $\leq_{\mathfrak{A}}$ on $\mathfrak{A}$ and a function $h_{\mathfrak{A}}$ from $\mathfrak{A}^n$ to $\mathfrak{A}$ for every n-ary size symbol $h \in \mathcal{H}$. A* size valuation *is a function ν from $\mathcal{Z}$ to $\mathfrak{A}$, naturally extended to a function on $\mathcal{A}$. A pre-model is a* model *if $a \leq_{\mathcal{A}} b$ implies $a\nu \leq_{\mathfrak{A}} b\nu$, for all size valuation ν. Such a model is* well-founded *if $>_{\mathfrak{A}}$ is well-founded.*

The Calculus of Algebraic Constructions with Size Annotations (CACSA) is an extension of CAC where constant predicate symbols are annotated by size expressions. The terms of CACSA are defined by the following grammar rule:

$$t ::= s \mid x \mid C^a \mid f \mid [x:t]t \mid (x:t)t \mid tt$$

where $C \in \mathcal{CF}^{\Box}$, $f \in \mathcal{F} \setminus \mathcal{CF}^{\Box}$ and $a \in \mathcal{A}$. We denote by $\mathcal{T}_{\mathcal{A}}(\mathcal{F}, \mathcal{X})$ the set of terms built from $\mathcal{F}$, $\mathcal{X}$ and $\mathcal{A}$. A product $(x:T)U$ with $x \notin \mathrm{FV}(U)$ is written $T \Rightarrow U$. We now assume that rewrite rules are built from annotated terms not containing size variables. Hence, if $t \rightarrow t'$ then, for all size substitution φ, $t\varphi \rightarrow t'\varphi$.

We also assume that every symbol f is equipped with a closed type $\tau_f = (\boldsymbol{x} : \boldsymbol{T})U$ with no size variable if $s_f = \Box$ (size variables are implicitly universally quantified otherwise), and $|\boldsymbol{l}| \leq |\boldsymbol{x}|$ if $f\boldsymbol{l} \to r \in \mathcal{R}$, a set $\mathrm{Mon}^+(f) \subseteq A_f = \{1, \ldots, |\boldsymbol{x}|\}$ of *monotone arguments* and a disjoint set $\mathrm{Mon}^-(f) \subseteq A_f$ of *anti-monotone arguments*. For a size symbol h, $\mathrm{Mon}^+(h)$ (resp. $\mathrm{Mon}^-(h)$) is taken to be the arguments in which $h_{\mathfrak{A}}$ is monotone (resp. anti-monotone).

Fig. 1. Typing rules

$$\text{(ax)} \qquad \vdash \star : \Box$$

$$\text{(size)} \qquad \frac{\vdash \tau_C : \Box}{\vdash C^a : \tau_C} \qquad (C \in \mathcal{CF}^\Box,\ a \in \mathcal{A})$$

$$\text{(symb)} \qquad \frac{\vdash \tau_f : s_f}{\vdash f : \tau_f \varphi} \qquad (f \notin \mathcal{CF}^\Box)$$

$$\text{(var)} \qquad \frac{\Gamma \vdash T : s_x}{\Gamma, x : T \vdash x : T} \qquad (x \notin \mathrm{dom}(\Gamma))$$

$$\text{(weak)} \qquad \frac{\Gamma \vdash t : T \quad \Gamma \vdash U : s_x}{\Gamma, x : U \vdash t : T} \qquad (x \notin \mathrm{dom}(\Gamma))$$

$$\text{(prod)} \qquad \frac{\Gamma \vdash U : s \quad \Gamma, x : U \vdash V : s'}{\Gamma \vdash (x : U)V : s'}$$

$$\text{(abs)} \qquad \frac{\Gamma, x : U \vdash v : V \quad \Gamma \vdash (x : U)V : s}{\Gamma \vdash [x : U]v : (x : U)V}$$

$$\text{(app)} \qquad \frac{\Gamma \vdash t : (x : U)V \quad \Gamma \vdash u : U}{\Gamma \vdash tu : V\{x \mapsto u\}}$$

$$\text{(sub)} \qquad \frac{\Gamma \vdash t : T \quad \Gamma \vdash T' : s}{\Gamma \vdash t : T'} \qquad (T \leq T')$$

An *environment* Γ is a sequence of variable-term pairs. Let $t \downarrow u$ iff there is v such that $t \to^* v \,{}^*\!\!\leftarrow u$, with $\to^*$ the reflexive and transitive closure of $\to = \to_\beta \cup \to_{\mathcal{R}}$. The typing rules of CACSA are given in Figure 1 and its subtyping rules in Figure 2. There are two differences with CAC. First, there is a new rule (size) for typing constant predicate symbols with size annotations, while the usual rule (symb) for typing symbols is restricted to the other symbols. Second, in CAC, the condition for (sub) is not $T \leq T'$ but $T \downarrow T'$. Note that, if $\to$ is confluent then $\downarrow$ is equivalent to $\leq$ without the subtyping rule (size).

Subtyping is necessary since size annotations are upper bounds. For instance, in an *if-then-else* expression, the *then*-branch does not need to exactly have the same type as the *else*-branch. Instead of subtyping, Xi uses singleton types, existential size variables and refinement types.

The way the subtyping relation is defined is due to Chen [12]. Replacing (red), (exp) and (refl) by $T \le U$ if $T \downarrow U$ would not allow us to prove that (trans) can be eliminated, which is essential for proving the compatibility of subtyping with the product construction (if $(x:U)V \le (x:U')V'$ then $U' \le U$ and $V \le V'$), which in turn enables one to prove that β preserves typing. Another consequence is that subtyping is decidable when applied on weakly normalizing terms. We refer the reader to [9] for more details on the meta-theory of our type system.

Fig. 2. Subtyping rules

(refl) $$T \le T$$

(size) $$C^a\boldsymbol{t} \le C^b\boldsymbol{t} \qquad (C \in \mathcal{CF}^\Box,\ a \le_{\mathcal{A}} b)$$

(prod) $$\frac{U' \le U \quad V \le V'}{(x:U)V \le (x:U')V'}$$

(red) $$\frac{T' \le U'}{T \le U} \qquad (T \to^* T',\ U' \ {}^*\!\!\leftarrow U)$$

(exp) $$\frac{T' \le U'}{T \le U} \qquad (T \ {}^*\!\!\leftarrow T',\ U' \to^* U)$$

(trans) $$\frac{T \le U \quad U \le V}{T \le V}$$

In this paper, we make two important assumptions:

(1) $\beta \cup \mathcal{R}$ is confluent. This is the case for instance if $\mathcal{R}$ is confluent and left-linear. Finding other sufficient conditions is an open problem.

(2) $\mathcal{R}$ preserves typing: if $l \to r \in \mathcal{R}$ and $\Gamma \vdash l\sigma : T$ then $\Gamma \vdash r\sigma : T$. Finding sufficient conditions with subtyping and dependent types does not seem easy. We leave the study of this problem for future work. With dependent or polymorphic symbols, requiring the existence of Γ and T such that $\Gamma \vdash l : T$ and $\Gamma \vdash r : T$ leads to non left-linear rules. In [10], we give general conditions avoiding the non-linearities implied by requiring l to be well-typed.

3 Constructor-based systems

We now study the case of CACSA's whose size algebra at least contains the following expressions:

$$a ::= \alpha \mid sa \mid \infty \mid \ \ldots$$

Following [5], when there is no other symbol, the ordering $\le_{\mathcal{A}}$ on size expressions is defined as the smallest congruent quasi-ordering $\le$ such that, for all a,

$a < sa$ and $a \leq \infty$, and size expressions are interpreted in $\mathfrak{A} = \Omega + 1$, where Ω is the first uncountable ordinal, by taking $s_{\mathfrak{A}}(\mathfrak{a}) = \mathfrak{a} + 1$ if $\mathfrak{a} < \Omega$, $s_{\mathfrak{A}}(\Omega) = \Omega$ and $\infty_{\mathfrak{A}} = \Omega$.

One can easily imagine other size expressions like $a + b$, $max(a, b)$, ...

We now define the sets of positive and negative positions in a term, which will enforce monotonicity and anti-monotonicity properties respectively. Then, we define the set of admissible inductive types.

Definition 2 (Positive and negative positions). *The set of positions (words over $\{L, R, S\}$) in a term t is inductively defined as follows:*

- $\mathrm{Pos}(s) = \mathrm{Pos}(x) = \mathrm{Pos}(f) = \{\varepsilon\}$ *(empty word)*
- $\mathrm{Pos}((x : u)v) = \mathrm{Pos}([x : u]v) = \mathrm{Pos}(uv) = L.\mathrm{Pos}(u) \cup R.\mathrm{Pos}(v)$
- $\mathrm{Pos}(C^a) = \{\varepsilon\} \cup S.\mathrm{Pos}(a)$

Let $\mathrm{Pos}(x, t)$ *($x \in \mathcal{F} \cup \mathcal{X} \cup \mathcal{Z}$) be the set of positions of the free occurrences of x in t. The set of* positive positions *in t,* $\mathrm{Pos}^+(t)$*, and the set of* negative positions *in t,* $\mathrm{Pos}^-(t)$*, are simultaneously defined by induction on t:*

- $\mathrm{Pos}^\delta(s) = \mathrm{Pos}^\delta(x) = \{\varepsilon \mid \delta = +\}$
- $\mathrm{Pos}^\delta((x : U)V) = L.\mathrm{Pos}^{-\delta}(U) \cup R.\mathrm{Pos}^\delta(V)$
- $\mathrm{Pos}^\delta([x : U]v) = R.\mathrm{Pos}^\delta(v)$
- $\mathrm{Pos}^\delta(tu) = L.\mathrm{Pos}^\delta(t)$ *if* $t \neq f\boldsymbol{t}$
- $\mathrm{Pos}^\delta(f\boldsymbol{t}) = \{L^{|\boldsymbol{t}|} \mid \delta = +\} \cup \bigcup\{L^{|\boldsymbol{t}|-i}R.\mathrm{Pos}^{\varepsilon\delta}(t_i) \mid \varepsilon \in \{-, +\}, i \in \mathrm{Mon}^\varepsilon(f)\}$
- $\mathrm{Pos}^\delta(C^a\boldsymbol{t}) = \mathrm{Pos}^\delta(C\boldsymbol{t}) \cup \{L^{|\boldsymbol{t}|}S \mid \delta = +\}.\mathrm{Pos}^\delta(a)$.

where $\delta \in \{-, +\}$, $-+ = -$ and $-- = +$ (usual rules of signs).

Definition 3 (Constructor-based system). *We assume given a* precedence *$\leq_\mathcal{F}$ on $\mathcal{F}$ and that every $C \in \mathcal{CF}^\square$ with $C : (\boldsymbol{z} : \boldsymbol{V})\star$ is equipped with a set* $\mathrm{Cons}(C)$ *of* constructors*, that is, a set of constant symbols $f : (\boldsymbol{y} : \boldsymbol{U})C^a\boldsymbol{v}$ equipped with a set* $\mathrm{Acc}(f) \subseteq A_f$ *of* accessible *arguments such that:*

- *If there are $D \simeq_\mathcal{F} C$ such that* $\mathrm{Pos}(D, U_j) \neq \emptyset$ *then there is $\alpha \in \mathcal{Z}$ such that $\mathcal{V}(\tau_f) = \{\alpha\}$ and $a = s\alpha$.*
- *For all $j \in$* $\mathrm{Acc}(c)$*:*
  - *For all $D >_\mathcal{F} C$,* $\mathrm{Pos}(D, U_j) = \emptyset$*.*
  - *For all $D \simeq_\mathcal{F} C$ and $p \in \mathrm{Pos}(D, U_j)$, $p \in \mathrm{Pos}^+(U_j)$ and $U_j|_p = D^\alpha$.*
  - *For all $p \in \mathrm{Pos}(\alpha, U_j)$, $p = qS$, $U_j|_q = D^\alpha$ and $D \simeq_\mathcal{F} C$.*
  - *For all $x \in \mathrm{FV}^\square(U_j)$, there is ι_x with $v_{\iota_x} = x$ and* $\mathrm{Pos}(x, U_j) \subseteq \mathrm{Pos}^+(U_j)$*.*
- *For all $F \in \mathcal{DF}^\square$ and $F\boldsymbol{l} \to r \in \mathcal{R}$:*
  - *For all $G >_\mathcal{F} F$,* $\mathrm{Pos}(G, r) = \emptyset$*.*
  - *For all $i \in \mathrm{Mon}^\delta(F)$, $l_i \in \mathcal{X}^\square$ and* $\mathrm{Pos}(l_i, r) \subseteq \mathrm{Pos}^\delta(r)$*.*
  - *For all $x \in \mathrm{FV}^\square(r)$, there is κ_x with $l_{\kappa_x} = x$.*

The positivity conditions are usual. The restrictions on a and α are also present in [5,2]. Section 5 proposes more general conditions. The conditions involving ι and κ mean that we restrict our attention to *small* inductive types for which predicate variables are parameters. See [6] for details about inductive types and weak/strong elimination.

An example is the inductive-recursive type $T : \star$ with constructors $v : nat \Rightarrow T$, $f : (list\ T) \Rightarrow T$ and $\mu : \neg\neg T \Rightarrow T$ (first-order terms with continuations), where $list : \star \Rightarrow \star$ is the type of polymorphic lists ($\mathrm{Mon}^+(list) = \{1\}$), $\neg : \star \Rightarrow \star$ ($\mathrm{Mon}^-(\neg) = \{1\}$) is defined by the rule $\neg\ A \rightarrow A \Rightarrow \bot$, and $\bot = (A : \star)A$.

We now give general conditions for rewrite rules to preserve strong normalization, based on the fundamental notion of *computability closure*. The computability closure of a term t is a set of terms that can be proved computable whenever t is computable. If, for every rule $f\boldsymbol{l} \rightarrow r$, r belongs to the computability closure of $\boldsymbol{l}$, then rules preserve computability, hence strong normalization.

In [10], the computability closure is inductively defined as a typing relation $\vdash_{\mathrm{c}}$ similar to $\vdash$ except for the (symb) case which is replaced by two new cases: (symb$^<$) for symbols strictly smaller than f, and (symb$^=$) for symbols equivalent to f whose arguments are structurally smaller than $\boldsymbol{l}$.

Here, (symb$^=$) is replaced by a new case for symbols equivalent to f whose arguments have, from typing, sizes smaller than those of $\boldsymbol{l}$. For comparing sizes, one can use metrics, similar to Dershowitz and Hoot's termination functions [14].

Definition 4 (Ordering on symbol arguments). *For every symbol $f : (\boldsymbol{x} : \boldsymbol{T})U$, we assume given two well-founded domains, $(D_f^{\mathcal{A}}, >_f^{\mathcal{A}})$ and $(D_f^{\mathfrak{A}}, >_f^{\mathfrak{A}})$, and two functions $\zeta_f^{\mathcal{A}} : \mathcal{A}^n \rightarrow D_f^{\mathcal{A}}$ and $\zeta_f^{\mathfrak{A}} : \mathfrak{A}^n \rightarrow D_f^{\mathfrak{A}}$ ($n = |\boldsymbol{x}|$) such that $(D_f^X, >_f^X) = (D_g^X, >_g^X)$ ($X \in \{\mathcal{A}, \mathfrak{A}\}$) whenever $f \simeq_{\mathcal{F}} g$, and we define:*

- *$a_f^i = a$ if $T_i = C^a\boldsymbol{v}$, and $a_f^i = \infty$ otherwise.*
- *$(f, \varphi) >^{\mathcal{A}} (g, \psi)$ iff $f >_{\mathcal{F}} g$ or $f \simeq_{\mathcal{F}} g$ and $\zeta_f^{\mathcal{A}}(\boldsymbol{a}_f\varphi) >_f^{\mathcal{A}} \zeta_g^{\mathcal{A}}(\boldsymbol{a}_g\psi)$.*
- *$(f, \nu) >^{\mathfrak{A}} (g, \mu)$ iff $f >_{\mathcal{F}} g$ or $f \simeq_{\mathcal{F}} g$ and $\zeta_f^{\mathfrak{A}}(\boldsymbol{a}_f\nu) >_f^{\mathfrak{A}} \zeta_g^{\mathfrak{A}}(\boldsymbol{a}_g\mu)$.*

Then, we assume that $>^{\mathcal{A}}$ is decidable and that $(f, \varphi) >^{\mathcal{A}} (g, \psi)$ implies $(f, \varphi\nu) >^{\mathfrak{A}} (g, \psi\nu)$ for all ν.

A simple metric is given by assigning a *status* to every symbol, that is, a non-empty sequence of multisets of positive integers, describing a simple combination of lexicographic and multiset comparisons. Given a set D and a status ζ of arity n (biggest integer occurring in it), we define $[\![\zeta]\!]_D$ on D^n as follows:

- $[\![M_1 \ldots M_k]\!]_D(\boldsymbol{x}) = ([\![M_1]\!]_D^m(\boldsymbol{x}), \ldots, [\![M_k]\!]_D^m(\boldsymbol{x}))$
- $[\![\{i_1, \ldots, i_p\}]\!]_D^m(\boldsymbol{x}) = \{x_{i_1}, \ldots, x_{i_p}\}$ (multiset)

Now, take $\zeta_f^X = [\![\zeta_f]\!]_X$, $D_f^X = \zeta_f^X(X^n)$ and $>_f^X = ((>_X)_{\mathrm{mul}})_{\mathrm{lex}}$.

For building the computability closure, one must start from the variables of the left hand-side. However, one cannot take any variable since, *a priori*, not every subterm of a computable term is computable. To this end, based on the interpretation of constant predicate symbols, we introduce the following notion:

Definition 5 (Accessibility). *We say that $u : U$ is a-*accessible *in $t : T$, written $t : T \rhd_a u : U$, iff $t = f\boldsymbol{u}$, $f \in \mathrm{Cons}(C)$, $f : (\boldsymbol{y} : \boldsymbol{U})C^{s\alpha}\boldsymbol{v}$, $|\boldsymbol{u}| = |\boldsymbol{y}|$, $u = u_j$, $j \in \mathrm{Acc}(f)$, $T = C^{s\alpha\varphi}\boldsymbol{v}\gamma$, $U = U_j\gamma\varphi$, $\boldsymbol{y}\gamma = \boldsymbol{u}$, $\alpha\varphi = a$ and $\mathrm{Pos}(\alpha, \boldsymbol{u}) = \emptyset$.*

A constructor $c : (\boldsymbol{y} : \boldsymbol{U})C^a\boldsymbol{v}$ *is* finitely branching[2] *iff, for all* $j \in \mathrm{Acc}(c)$, *either* $\mathrm{Pos}(\alpha, U_j) = \emptyset$ *or there exists* D *such that* $U_j = D^\alpha\boldsymbol{u}$. *We say that* $u : U$ *is* strongly a-accessible *in* $t : T$*, written* $t : T \mathrel{\rhd\!\!\!\rhd}_a u : U$*, iff* $t : T \rhd_a u : U$*,* f *is a finitely branching constructor and* $\mathrm{Pos}(\alpha, U_j) \neq \emptyset$.

We say that $u : U$ *is* $*$-accessible modulo φ *in* $t : T$*, written* $t : T \gg_\varphi u : U$*, iff either* $t : T\varphi = u : U$ *and* $\varphi|_{\mathcal{V}(T)}$ *is a renaming,*[3] *or* $t : T\varphi \mathrel{\rhd\!\!\!\rhd}^* \rhd_\epsilon u : U$ *for some* $\epsilon \in \mathcal{Z}$.

This seems to restrict matching to constructors as in ML-like languages. However, one can prove that, for first-order data types, computability is equivalent to strong normalization [9]. Thus, every argument of a first-order symbol can be declared as accessible, and matching on defined first-order function symbols is possible. Meanwhile, it may be uneasy to find for these symbols output sizes and measures satisfying all the constraints required for subject reduction and recursive calls. More research has to be done on this subject.

Definition 6 (Termination criterion). *For every rule* $f\boldsymbol{l} \to r \in \mathcal{R}$ *with* $f : (\boldsymbol{x} : \boldsymbol{T})U$ *and* $\boldsymbol{x}\gamma = \boldsymbol{l}$*, we assume given a size substitution* φ*. The* computability closure *for this rule is given by the type system of Figure 3 on the set of terms* $\mathcal{T}_\mathcal{A}(\mathcal{F}', \mathcal{X}')$ *where* $\mathcal{F}' = \mathcal{F} \cup \mathrm{dom}(\Gamma)$*,* $\mathcal{X}' = \mathcal{X} \setminus \mathrm{dom}(\Gamma)$ *and, for all* $x \in \mathrm{dom}(\Gamma)$*,* $\tau_x = x\Gamma$ *and* $x <_\mathcal{F} f$*. The termination conditions are:*

- *Well-typedness: for all* $x \in \mathrm{dom}(\Gamma)$*,* $\vdash_c l_i : T_i\varphi\gamma$.
- *Linearity:* Γ *is linear w.r.t. size variables.*
- *Accessibility: for all* $x \in \mathrm{dom}(\Gamma)$*, there are* i *and* β *such that* $l_i : T_i\gamma \gg_\varphi x : x\Gamma$*,* $T_i = C^\beta\boldsymbol{t}$ *and* $\mathcal{V}(\boldsymbol{t}) = \emptyset$.
- *Computability closure:* $\vdash_c r : U\varphi\gamma$.
- *Positivity: for all* $\alpha \in \mathcal{V}(\boldsymbol{T})$*,* $\mathrm{Pos}(\alpha, U) \subseteq \mathrm{Pos}^+(U)$.
- *Safeness:* γ *is an injection from* $\mathrm{dom}^\Box(\Gamma_f)$ *to* $\mathrm{dom}^\Box(\Gamma)$.

The positivity condition on the output type of f w.r.t. size variables appears in the previous works on sized types too. It may be extended to more general continuity conditions [20,1]. In [3], Abel gives an example of a function which is not terminating because it does not satisfy such a condition.

As for the safeness condition, it simply says that one cannot do matching or have non-linearities on predicate variables, which is known to lead to non-termination in some cases [19]. It is also part of other works on CC with inductive types [24] and rewriting [25].

The linearity, positivity, safeness and accessibility conditions are decidable. We think that the other conditions are decidable too, under the assumption that the satisfiability of inequality constraints in $\mathcal{A}$ is decidable. To this end, we prove the strong normalization of well-typed terms in Section 4, and describe a type inference algorithm in [9]. In practice, like Xi, we can restrict size expressions to linear arithmetic, for which the satisfiability of inequality constraints is decidable.

[2] Constructors of usual first-order data types are finitely branching.
[3] An injection from a finite subset of $\mathcal{Z}$ to $\mathcal{Z}$.

Fig. 3. Computability closure of $(f\boldsymbol{l} \to r, \Gamma, \varphi)$ with $f : (\boldsymbol{x} : \boldsymbol{T})U$ and $\boldsymbol{x}\gamma = \boldsymbol{l}$

$$\text{(ax)} \quad \frac{}{\vdash_c \star : \square}$$

$$\text{(size)} \quad \frac{\vdash_c \tau_C : \square}{\vdash_c C^a : \tau_C} \quad (C \in \mathcal{CF}^\square)$$

$$\text{(symb)} \quad \frac{\vdash_c \tau_g : s_g \quad (\forall i) \Delta \vdash_c y_i\delta : U_i\psi\delta}{\Delta \vdash_c g\boldsymbol{y}\delta : V\psi\delta} \quad \begin{array}{c}(g \notin \mathcal{CF}^\square,\ g : (\boldsymbol{y} : \boldsymbol{U})V, \\ (g, \psi) <^{\mathcal{A}} (f, \varphi))\end{array}$$

$$\text{(var)} \quad \frac{\Delta \vdash_c T : s_x}{\Delta, x : T \vdash_c x : T} \quad (x \notin \mathrm{dom}(\Delta))$$

$$\text{(weak)} \quad \frac{\Delta \vdash_c t : T \quad \Delta \vdash_c U : s_x}{\Delta, x : U \vdash_c t : T} \quad (x \notin \mathrm{dom}(\Delta))$$

$$\text{(prod)} \quad \frac{\Delta, x : U \vdash_c V : s}{\Delta \vdash_c (x : U)V : s}$$

$$\text{(abs)} \quad \frac{\Delta, x : U \vdash_c v : V \quad \Delta \vdash_c (x : U)V : s}{\Delta \vdash_c [x : U]v : (x : U)V}$$

$$\text{(app)} \quad \frac{\Delta \vdash_c t : (x : U)V \quad \Delta \vdash_c u : U}{\Delta \vdash_c tu : V\{x \mapsto u\}}$$

$$\text{(conv)} \quad \frac{\Delta \vdash_c t : T \quad \Delta \vdash_c T : s \quad \Delta \vdash_c T' : s}{\Delta \vdash_c t : T'} \quad (T \le T')$$

Note that, with polymorphic or dependent function symbols, the well-typedness condition makes the rules non left-linear. For instance, with concatenation on polymorphic list: $app\ A\ (cons\ A'\ x\ l)\ l' \to cons\ A\ x\ (app\ A\ l\ l')$, we need to take $A' = A$. In [10], we proved that, in CAC, this condition can be relaxed by relativizing the previous conditions with the substitution $\{A' \mapsto A\}$. The same technique should apply to CACSA.

We now give some examples satisfying these conditions:

Example 1 (Division on natural numbers). Take $nat : \star$, $0 : nat^0$, $s : nat^\alpha \Rightarrow nat^{s\alpha}$, $- : nat^\alpha \Rightarrow nat^\beta \Rightarrow nat^\alpha$ and $/ : nat^\alpha \Rightarrow nat^\beta \Rightarrow nat^\alpha$.

- For rule (3), take $\zeta_-(\alpha, \beta) = \alpha$, $\Gamma = x : nat^\delta, y : nat^\epsilon$, $\varphi = \{\alpha \mapsto s\delta, \beta \mapsto s\epsilon\}$ and $s <_{\mathcal{F}} -$. By (symb), $\vdash_c x : nat^\delta$ and $\vdash_c y : nat^\epsilon$. By (symb), $\vdash_c -xy : nat^\delta$ since $\zeta_-(\delta, \epsilon) = \delta < \zeta_-(s\delta, s\epsilon) = s\delta$. Thus, by (sub), $\vdash_c -xy : nat^{s\delta}$.
- For rule (5), take $\zeta_/(\alpha, \beta) = \alpha$, $\Gamma = x : nat^\delta, y : nat^\epsilon$, $\gamma = \{p \mapsto sx, q \mapsto y\}$, $\varphi = \{\alpha \mapsto s\delta, \beta \mapsto \epsilon\}$ and $- <_{\mathcal{F}} /$. By (symb), $\vdash_c x : nat^\delta$ and $\vdash_c y : nat^\epsilon$. By (symb), $\vdash_c -xy : nat^\delta$. By (symb), $\vdash_c /(-xy)y : nat^\delta$ since $\zeta_/(\delta, \epsilon) = \delta < \zeta_/(s\delta, \epsilon) = s\delta$. Thus, by (symb), $\vdash_c s(/(-xy)y) : nat^{s\delta}$.

Example 2 (Addition on Brouwer's ordinals). Take $ord : \star$, $0 : nat^0$, $s : nat^\alpha \Rightarrow nat^{s\alpha}$, $lim : (nat \Rightarrow ord^\alpha) \Rightarrow ord^{s\alpha}$ and $+ : nat^\alpha \Rightarrow nat^\beta \Rightarrow nat^\infty$. For rule (3), take $\zeta_+(\alpha,\beta) = \alpha$, $\Gamma = f : nat^\infty \Rightarrow ord^\delta, y : ord^\epsilon$, $\varphi = \{\alpha \mapsto s\delta, \beta \mapsto \epsilon\}$ and $s, lim <_\mathcal{F} +$. By (symb), $\vdash_\mathrm{c} f : nat^\infty \Rightarrow ord^\delta$ and $\vdash_\mathrm{c} y : ord^\epsilon$. Let $\Delta = x : nat^\infty$. By (var), $\Delta \vdash_\mathrm{c} x : nat^\infty$. By (weak), $\Delta \vdash_\mathrm{c} f : nat^\infty \Rightarrow ord^\delta$ and $\Delta \vdash_\mathrm{c} y : ord^\epsilon$. By (app), $\Delta \vdash_\mathrm{c} fx : ord^\delta$. By (symb), $\Delta \vdash_\mathrm{c} +(fx)y : ord^\infty$ since $\zeta_+(\delta,\epsilon) = \delta < \zeta_+(s\delta,\epsilon) = s\delta$. By (abs), $\vdash_\mathrm{c} [x : nat^\infty](+(fx)y) : (x : nat^\infty)ord^\delta$. Thus, by (symb), $\vdash_\mathrm{c} lim([x : nat^\infty](+(fx)y)) : ord^{s\delta}$. This does not enter Xi's framework.

Example 3 (Huet and Hullot's reverse function). Take $list : \star$, $nil : list^0$, $cons : nat^\infty \Rightarrow list^\alpha \Rightarrow list^{s\alpha}$, $rev1 : nat^\infty \Rightarrow list^\infty \Rightarrow nat^\infty$, $rev2 : nat^\infty \Rightarrow list^\beta \Rightarrow list^\beta$ and $rev : list^\alpha \Rightarrow list^\alpha$.

$$
\begin{array}{lrcl}
(1) & rev1\ x\ nil & \rightarrow & x \\
(2) & rev1\ x\ (cons\ y\ l) & \rightarrow & rev1\ y\ l \\[4pt]
(3) & rev2\ x\ nil & \rightarrow & nil \\
(4) & rev2\ x\ (cons\ y\ l) & \rightarrow & rev\ (cons\ x\ (rev\ (rev2\ y\ l))) \\[4pt]
(5) & rev\ nil & \rightarrow & nil \\
(6) & rev\ (cons\ x\ l) & \rightarrow & cons\ (rev1\ x\ l)\ (rev2\ x\ l)
\end{array}
$$

For rule (4), take $\zeta_{rev}(\alpha) = 2\alpha$, $\zeta_{rev2}(\alpha,\beta) = 2\beta+1$, $\Gamma = x : nat^\infty, y : nat^\infty, l : list^\delta$, $\varphi = \{\beta \mapsto \delta+1\}$ and $rev \simeq_\mathcal{F} rev2 >_\mathcal{F} rev1 >_\mathcal{F} cons, nil$. Then, one can check that $\zeta_{rev2}(\infty,\delta+1) = 2\delta+3$ is strictly greater than $\zeta_{rev2}(\infty,\delta) = 2\delta+1$, $\zeta_{rev}(\delta) = 2\delta$ and $\zeta_{rev}(1+\delta) = 2\delta+2$.

4 Termination proof

The termination proof follows the computability-based method of [10]. For lack of space, we just state the most important theorems. See [9] for details.

Let $\mathcal{R}_t$ be the set of possible interpretations for the terms of type t. $\mathcal{R}_s$ is made of sets of strongly normalizable terms. $\mathcal{R}_{(x:T)U}$ is made of the functions from $\mathcal{T} \times \mathcal{R}_T$ to $\mathcal{R}_U$ that are invariant by reduction or size substitution. $\mathcal{R}_t^m$ is the subset of $\mathcal{R}_t$ made of the functions that are monotone (resp. anti-monotone) in their monotone (resp. anti-monotone) arguments.

We first define the interpretation of types. Then, we prove monotonicity properties, the correctness of accessibility w.r.t. computability (accessible subterms of a computable term are computable), the correctness of the computability closure (every term of the computability closure is computable) and the computability of every symbol, hence the strong normalization of every well-typed term.

Definition 7 (Interpretation schema). *A* candidate assignment *is a function ξ from $\mathcal{X}$ to $\bigcup\{\mathcal{R}_t \mid t \in \mathcal{T}\}$. A candidate assignment ξ is a* Γ-assignment, *written $\xi \models \Gamma$, if, for all $x \in \mathrm{dom}(\Gamma)$, $x\xi \in \mathcal{R}_{x\Gamma}$.*

An interpretation *for a symbol $C \in \mathcal{CF}^\square$ is a monotone function I from $\mathfrak{A}$ to $\mathcal{R}^m_{\tau_f}$. An* interpretation *for a symbol $f \notin \mathcal{CF}^\square$ is an element of $\mathcal{R}^m_{\tau_f}$. An* interpretation *for a set $\mathcal{G}$ of predicate symbols is a function which, to every symbol $g \in \mathcal{G}$, associates an interpretation for g.*

The interpretation *of t w.r.t. a candidate assignment ξ, an interpretation I for $\mathcal{F}$, a substitution θ and a valuation ν, $[\![t]\!]^{I,\nu}_{\xi,\theta}$, is defined by induction on t:*

- $[\![t]\!]^{I,\nu}_{\xi,\theta} = \emptyset$ *if t is an object or a sort*
- $[\![F]\!]^{I,\nu}_{\xi,\theta} = I_F$ *if $F \in \mathcal{DF}^\square$*
- $[\![C^a]\!]^{I,\nu}_{\xi,\theta} = I^{a\nu}_C$ *if $C \in \mathcal{CF}^\square$*
- $[\![x]\!]^{I,\nu}_{\xi,\theta} = x\xi$
- $[\![(x:U)V]\!]^{I,\nu}_{\xi,\theta} = \{t \in \mathcal{T} \mid \forall u \in [\![U]\!]^{I,\nu}_{\xi,\theta}, \forall S \in \mathcal{R}_U, tu \in [\![V]\!]^{I,\nu}_{\xi^S_x,\theta^u_x}\}$
- $[\![[x:U]v]\!]^{I,\nu}_{\xi,\theta}(u,S) = [\![v]\!]^{I,\nu}_{\xi^S_x,\theta^u_x}$
- $[\![tu]\!]^{I,\nu}_{\xi,\theta} = [\![t]\!]^{I,\nu}_{\xi,\theta}(u\theta, [\![u]\!]^{I,\nu}_{\xi,\theta})$

where $\theta^u_x = \theta \cup \{x \mapsto u\}$ and $\xi^S_x = \xi \cup \{x \mapsto S\}$. A substitution θ is adapted *to a Γ-assignment ξ and a valuation ν, written $\xi, \theta \models_\nu \Gamma$, if $\mathrm{dom}(\theta) \subseteq \mathrm{dom}(\Gamma)$ and, for all $x \in \mathrm{dom}(\theta)$, $x\theta \in [\![x\Gamma]\!]^{I,\nu}_{\xi,\theta}$.*

We define the interpretation of predicate symbols by induction on $>_\mathcal{F}$. The definition of defined predicate symbols can be found in [10]. We now define the interpretation of constant predicate symbols by transfinite induction on $\mathfrak{a} \in \mathfrak{A}$.

Definition 8 (Interpretation of constant predicate symbols).

- $I^0_C(\boldsymbol{t}, \boldsymbol{S})$[4] *is the set of $u \in \mathcal{SN}$ that never reduces to a term of the form $f\boldsymbol{u}$ with $f \in \mathrm{Cons}(C)$, $f : (\boldsymbol{y}:\boldsymbol{U})C^a\boldsymbol{v}$, $|\boldsymbol{u}| = |\boldsymbol{y}|$ and $\mathrm{Acc}(f) \neq \emptyset$.*
- $I^{\mathfrak{a}+1}_C(\boldsymbol{t}, \boldsymbol{S})$ *is the set of terms $u \in \mathcal{SN}$ such that, if u reduces to a constructor term $f\boldsymbol{u}$ with $f : (\boldsymbol{y}:\boldsymbol{U})C^{s\alpha}\boldsymbol{v}$ then, for all $j \in \mathrm{Acc}(f)$, $u_j \in [\![U_j]\!]^{I,\nu}_{\xi,\theta}$ with $y\xi = S_{\iota_y}$, $\boldsymbol{y}\theta = \boldsymbol{u}$ and $\alpha\nu = \mathfrak{a}$.*
- $I^\mathfrak{b}_C = \bigwedge_{\tau_C}(\{I^\mathfrak{a}_C \mid \mathfrak{a} < \mathfrak{b}\})$ *if $\mathfrak{b}$ is a limit ordinal.*

For $t \in I^\Omega_C(\boldsymbol{S})$, let $o_{C(\boldsymbol{S})}(t)$ be the smallest ordinal $\mathfrak{a}$ such that $t \in I^\mathfrak{a}_C(\boldsymbol{S})$.

The interpretation is well defined thanks to the assumptions made on constructors, and the following properties of the interpretation schema:

Lemma 1 (Monotonicity). *Let $\leq^+ = \leq$; $\leq^- = \geq$; $\xi \leq_x \xi'$ iff $x\xi \leq x\xi'$ and, for all $y \neq x$, $y\xi = y\xi'$; $I \leq_f I'$ iff $I_f \leq I'_f$ and, for all $g \neq f$, $I_g = I'_g$; $\nu \leq_\alpha \nu'$ iff $\alpha\nu \leq_\mathfrak{A} \alpha\nu'$ and, for all $\beta \neq \alpha$, $\beta\nu = \beta\nu'$. Assume that $\Gamma \vdash t : T$ and $\xi, \xi' \models \Gamma$.*

- *If $\xi \leq_x \xi'$ and $\mathrm{Pos}(x,t) \subseteq \mathrm{Pos}^\delta(t)$ then $[\![t]\!]^{I,\nu}_{\xi,\theta} \leq^\delta [\![t]\!]^{I,\nu}_{\xi',\theta}$.*
- *If $I \leq_f I'$ and $\mathrm{Pos}(f,t) \subseteq \mathrm{Pos}^\delta(t)$ then $[\![t]\!]^{I,\nu}_{\xi,\theta} \leq^\delta [\![t]\!]^{I',\nu}_{\xi,\theta}$.*
- *If $\nu \leq_\alpha \nu'$ and $\mathrm{Pos}(\alpha,t) \subseteq \mathrm{Pos}^\delta(t)$ then $[\![t]\!]^{I,\nu}_{\xi,\theta} \leq^\delta [\![t]\!]^{I,\nu'}_{\xi,\theta}$.*
- *If $\Gamma \vdash T \leq T' : s$, $T, T' \in \mathcal{WN}$, $[\![t]\!] = [\![t']\!]$ whenever $t \to t'$, then $[\![T]\!]^{I,\nu}_{\xi,\theta} \leq [\![T']\!]^{I,\nu}_{\xi,\theta}$.*

[4] In the following, we do not write $\boldsymbol{t}$ since the interpretation does not depend on it.

Theorem 1 (Accessibility correctness). *If $t : T \gg_\varphi u : U$, $T = C^\beta \boldsymbol{t}$, $\mathcal{V}(\boldsymbol{t}) = \emptyset$ and $t\sigma \in [\![T]\!]^\mu_{\xi,\sigma}$ then there is ν such that $\beta\varphi\nu \le \beta\mu$ and $u\sigma \in [\![U]\!]^\nu_{\xi,\sigma}$.*

Theorem 2 (Correctness of the computability closure). *Let $(f\boldsymbol{l} \to r, \Gamma, \varphi) \in \mathcal{R}$, $f : (\boldsymbol{x} : \boldsymbol{T})U$ and $\boldsymbol{x}\gamma = \boldsymbol{l}$. Assume that, for all $(g,\mu) <^{\mathfrak{A}} (f, \varphi\nu)$, $g \in [\![\tau_g]\!]^\mu$. If $\Delta \vdash_c t : T$ and $\xi, \sigma \models_\nu \Gamma, \Delta$ then $t\sigma \in [\![T]\!]^\nu_{\xi,\sigma}$.*

Proof. By induction on $\Delta \vdash_c t : T$. We only detail the case (symb). Since $(g,\psi) <^{\mathcal{A}} (f,\varphi)$, $(g,\psi\nu) <^{\mathfrak{A}} (f,\varphi\nu)$. Hence, by assumption, $g \in [\![\tau_g]\!]^{\psi\nu}$. Now, by induction hypothesis, $\boldsymbol{y}\delta\sigma \in [\![\boldsymbol{U}\psi\delta]\!]^\nu_{\xi,\sigma}$. By candidate substitution, there exists η such that $[\![\boldsymbol{U}\psi\delta]\!]^\nu_{\xi,\sigma} = [\![\boldsymbol{U}\psi]\!]^\nu_{\eta,\delta\sigma}$. By size substitution, $[\![\boldsymbol{U}\psi]\!]^\nu_{\eta,\delta\sigma} = [\![\boldsymbol{U}]\!]^{\psi\nu}_{\eta,\delta\sigma}$. Therefore, $g\boldsymbol{y}\delta\sigma \in [\![V]\!]^{\psi\nu}_{\eta,\delta\sigma} = [\![V\psi\delta]\!]^\nu_{\xi,\sigma}$. □

Lemma 2 (Computability of symbols). *For all f and μ, $f \in [\![\tau_f]\!]^\mu$.*

Proof. Assume that $\tau_f = (\boldsymbol{x} : \boldsymbol{T})U$ with U distinct from a product. $f \in [\![\tau_f]\!]^\mu$ iff, for all η, θ such that $\eta, \theta \models_\mu \Gamma_f$, $f\boldsymbol{x}\theta \in [\![U]\!]^\mu_{\eta,\theta}$. We prove it by induction on $((f,\mu),\theta)$ with $(>^{\mathfrak{A}}, \to)_{\text{lex}}$ as well-founded ordering. □

Theorem 3 (Termination). *$\beta \cup \mathcal{R}$ is well-founded on well-typed terms.*

5 Towards another extension: sized constructors

By definition, constructors are restricted to types of the form $(\boldsymbol{y} : \boldsymbol{U})C^{s\alpha}\boldsymbol{v}$ with every occurrence of a type $D \simeq_{\mathcal{F}} C$ in $\boldsymbol{U}$ of the form D^α (this is so in [5,2] too). However, some functions need more general size annotations [17]:

Example 4 (Paulson's normalization procedure of if-expressions). By taking the types $expr : \star$, $at : expr^0$, $if : expr^\alpha \Rightarrow expr^\beta \Rightarrow expr^\gamma \Rightarrow expr^{(\alpha+1)(\beta+\gamma+3)}$ and $nm : expr^\alpha \Rightarrow expr^\alpha$, one can prove the termination conditions for the rules:

$$\begin{array}{lrcl}
(1) & nm\ at & \to & at \\
(2) & nm\ (if\ at\ y\ z) & \to & if\ at\ (nm\ y)\ (nm\ z) \\
(3) & nm\ (if\ (if\ u\ v\ w)\ y\ z) & \to & nm\ (if\ u\ (nm\ (if\ v\ y\ z))\ (nm\ (if\ w\ y\ z)))
\end{array}$$

For rule (3), take $\zeta_{nm}(\alpha) = \alpha$, $\Gamma = u : expr^\alpha, v : expr^\beta, w : expr^\gamma, y : expr^\delta, z : expr^\epsilon$, $\upsilon = (\alpha+1)(\beta+\gamma+3)(\delta+\epsilon+3)$, $\varphi = \{\alpha \mapsto \upsilon\}$ and $nm >_{\mathcal{F}} at, if$. Then, one can check that υ is strictly greater than $(\beta+1)(\delta+\epsilon+3)$, $(\gamma+1)(\delta+\epsilon+3)$ and $(\alpha+1)((\beta+1)(\delta+\epsilon+3)+(\gamma+1)(\delta+\epsilon+3)+3)$.

The conditions on constructors imply also that non-recursive arguments are of size ∞ (*i.e.* undefined). So, there is no way to give different sizes to the terms of a non-recursive type. Yet, it may be very useful as shown by the type *blist* in the following example.

Example 5 (Quick sort). Take $bool : \star$, $true : bool^\infty$, $false : bool^\infty$, $blist : \star$, $pair : list^\alpha \Rightarrow list^\beta \Rightarrow blist^{max(\alpha,\beta)}$ or $pair : list^\alpha \Rightarrow list^\alpha \Rightarrow blist^\alpha$, $fst : blist^\alpha \Rightarrow list^\alpha$, $snd : blist^\alpha \Rightarrow list^\alpha$, $\le : nat^\infty \Rightarrow nat^\infty \Rightarrow bool^\infty$, $pivot : nat^\infty \Rightarrow list^\alpha \Rightarrow blist^\alpha$, $qs : list^\infty \Rightarrow list^\infty \Rightarrow list^\infty$ and $qsort : list^\infty \Rightarrow list^\infty$.

$$\begin{array}{lll} & (3)\quad \leq\ 0\ x \rightarrow true & \\ (1)\ fst\ (pair\ x\ y) \rightarrow x & (4)\quad \leq\ (s\ x)\ 0 \rightarrow false & (6)\ \ if\ true\ x\ y \rightarrow x \\ (2)\ snd\ (pair\ x\ y) \rightarrow y & (5) \leq\ (s\ x)\ (s\ y) \rightarrow \leq\ x\ y & (7)\ if\ false\ x\ y \rightarrow y \end{array}$$

$$\begin{array}{lrl} (8) & pivot\ x\ nil \rightarrow & pair\ nil\ nil \\ (9) & pivot\ x\ (cons\ y\ l) \rightarrow & if\ (\leq\ y\ x)\ (pair\ (cons\ y\ u)\ v)\ (pair\ u\ (cons\ y\ v)) \\ & & \text{where } u = fst\ (pivot\ x\ l) \text{ and } v = snd\ (pivot\ x\ l) \end{array}$$

$$\begin{array}{lrl} (10) & qs\ nil\ l \rightarrow & l \\ (11) & qs\ (cons\ x\ l)\ l' \rightarrow & qs\ u\ (cons\ x\ (qs\ v\ l')) \\ & & \text{where } u = fst\ (pivot\ x\ l) \text{ and } v = snd\ (pivot\ x\ l) \end{array}$$

$$\begin{array}{lrl} (12) & qsort\ l \rightarrow & qs\ l\ nil \end{array}$$

For rule (11), take $\zeta_{qs}(\alpha, \beta) = \alpha$, $\Gamma = x : nat^\infty, l : list^\delta, l' : list^\epsilon$, $\varphi = \{\alpha \mapsto s\delta, \beta \mapsto \epsilon\}$ and $qs >_\mathcal{F} pivot >_\mathcal{F} cons, pair, fst, snd$. By (symb), $\vdash_c x : nat^\infty$, $\vdash_c l : list^\delta$ and $\vdash_c l' : list^\epsilon$. By (symb), $\vdash_c pivot\ x\ l : blist^\delta$. By (symb), $\vdash_c u : list^\delta$ and $\vdash_c v : list^\delta$. By (symb), $\vdash_c qs\, v\, l' : list^\infty$. By (symb), $\vdash_c cons\, x\, (qs\, v\, l') : list^{s\infty}$. By (sub), $\vdash_c cons\, x\, (qs\, v\, l') : list^\infty$. Thus, by (symb), $\vdash_c qs\, u\, (cons\, x\, (qs\, v\, l')) : list^\infty$ since $\zeta_{qs}(\delta, \infty) = \delta < \zeta_{qs}(s\delta, \epsilon) = s\delta$.

Therefore, we naturally come to the following more general conditions, whose justification is ongoing.

Definition 9 (Sized constructors). *A type C is* non-recursive *if, for all constructor $f : (\boldsymbol{y} : \boldsymbol{U})C^\alpha \boldsymbol{v}$ and $j \in \mathrm{Acc}(f)$, no $D \simeq C$ occurs in U_j. The first, third and fourth conditions of Definition 3 are replaced by the following ones:*

- *For all $j \in \mathrm{Acc}(f)$, $D \simeq_\mathcal{F} C$ and $p \in \mathrm{Pos}(D, U_j)$, $p \in \mathrm{Pos}^+(U_j)$ and $U_j|_p = D^\alpha$ for some $\alpha <_\mathcal{A} a$ ($\alpha \leq_\mathcal{A} a$ if C is non-recursive).*
- *For all $j \in \mathrm{Acc}(f)$, $\alpha \in \mathcal{V}(U_j)$ and $p \in \mathrm{Pos}(\alpha, U_j)$, there is $D \simeq_\mathcal{F} C$ and $q \in \mathrm{Pos}(D, U_j)$ such that $p = qS$.*

Note however that it still does not allow us to take $qs : list^\alpha \Rightarrow list^\beta \Rightarrow list^{\alpha+\beta}$ and thus $qsort : list^\alpha \Rightarrow list^\alpha$ since too much information is lost by taking $pair : list^\alpha \Rightarrow list^\beta \Rightarrow blist^{max(\alpha,\beta)}$. A solution would be to take $pair : list^\alpha \Rightarrow list^\beta \Rightarrow blist^{\langle\alpha,\beta\rangle}$ with $\langle\alpha, \beta\rangle$ interpreted as a pair of ordinals, and to say that $pivot$ has type $nat^\infty \Rightarrow list^\alpha \Rightarrow blist^{\langle\beta,\gamma\rangle}$ for some β and γ such that $\beta + \gamma = \alpha$, as it can be done in [26].

Another interest of Xi's framework is to take into account the semantics of conditional statements:

Example 6 (Mc Carthy's "91" function). Mc Carthy's "91" function f is defined by the following equations: $f(x) = f(f(x + 11))$ if $x \leq 100$, and $f(x) = x - 10$ otherwise. In fact, f is equal to the function F such that $F(x) = 91$ if $x \leq 100$, and $F(x) = x - 10$ otherwise. A way to formalize this in CACSA would be to use conditional rewrite rules:

$$\begin{array}{llll} (1)\ f\ x & \rightarrow & f\ (f\ (+\ x\ 11)) & \text{if } \leq\ x\ 100 = true \\ (2)\ f\ x & \rightarrow & -\ x\ 10 & \text{if } \leq\ x\ 100 = false \end{array}$$

and take $f : nat^{\alpha} \Rightarrow nat^{F(\alpha)}$ and $\zeta_f^X(x) = max(0, 101 - x)$ as measure function, as it can be done in Xi's framework. Then, by taking into account the rewrite rule conditions, one could prove that, if $\Gamma = x : nat^{\delta}$ and $\leq\ x\ 100 = true$, then $\delta \leq 100$, $\zeta_f(\delta + 11) < \zeta_f(\delta)$ and $\zeta_f(F(\delta)) < \zeta_f(\delta)$.

6 Conclusion

The notion of computability closure, first introduced in [11] and further extended to higher-order pattern-matching [8], higher-order recursive path ordering [21,25], type-level rewriting[10] and rewriting modulo equational theories [7], shows to be essential for extending to rewriting and dependent types type-based termination criteria for (polymorphic) λ-calculi with inductive types and case analysis [20,26,5,2]. In contrast with what is suggested in [5], this notion, which is expressed as a sub-system of the whole type system (see Figure 3), allows pattern-matching and does not suffer from limitations one could find in systems relying on external guard predicates for recursive definitions.

We allow a richer size algebra than the one in [20,5,2] but do not allow existential size variables and conditional rewriting that are essential for capturing some size-preserving properties or some definitions as it can be done in [26]. Such extensions should allow us to subsume Xi's work completely.

Some questions also need further research. In particular, matching on defined symbols and decidability of type-checking. For type-checking, we believe that it is decidable if solving inequations in $\mathcal{A}$ is decidable. We already have preliminary results in this direction [9].

We made two important assumptions that also need further research. First, the confluence of $\beta \cup \mathcal{R}$, which is still an open problem when $\mathcal{R}$ is confluent, terminating and non left-linear. Second, the preservation of typing under rewriting for which we need to find decidable sufficient conditions.

We also assume that users provide appropriate sized types for function symbols and then check by our technique that the rewrite rules defining these function symbols are compatible with their types. An important extension would be to infer these types. Works in this direction already exist for ML-like languages.

Finally, by combining rewriting and subtyping in CC, this work may also be seen as an important step towards a better integration of membership equational logic and dependent type systems. Following [21,25], we also think that it can serve as a basis for a higher-order extension of the General Path Ordering [14].

Acknowledgments. I thank Ralph Matthes for having invited me in Münich in February 2002, Andreas Abel for his useful comments on a previous version of this work (his technical report [2] and the discussions we had in Münich were the starting point of the present work), and the anonymous referees too.

References

1. A. Abel. Termination and productivity checking with continuous types. In *Proc. of TLCA'03*, LNCS 2701.

2. A. Abel. Termination checking with types. Technical Report 0201, Ludwig Maximilians Universität, München, Germany, 2002.
3. A. Abel. Termination checking with types, 2003. Submitted to ITA.
4. H. Barendregt. Lambda calculi with types. In S. Abramski, D. Gabbay, and T. Maibaum, editors, *Handbook of logic in computer science*, volume 2. Oxford University Press, 1992.
5. G. Barthe, M. J. Frade, E. Giménez, L. Pinto, and T. Uustalu. Type-based termination of recursive definitions. *Mathematical Structures in Computer Science*, 14(1):97–141, 2004.
6. F. Blanqui. Inductive types in the Calculus of Algebraic Constructions. In *Proc. of TLCA'03*, LNCS 2701.
7. F. Blanqui. Rewriting modulo in Deduction modulo. In *Proc. of RTA'03*, LNCS 2706.
8. F. Blanqui. Termination and confluence of higher-order rewrite systems. In *Proc. of RTA'00*, LNCS 1833.
9. F. Blanqui. A type-based termination criterion for dependently-typed higher-order rewrite systems. Draft. 38 pages. `http://www.loria.fr/~blanqui/`.
10. F. Blanqui. Definitions by rewriting in the Calculus of Constructions, 2003. To appear in Mathematical Structures in Computer Science.
11. F. Blanqui, J.-P. Jouannaud, and M. Okada. Inductive-data-type Systems. *Theoretical Computer Science*, 272:41–68, 2002.
12. G. Chen. *Subtyping, Type Conversion and Transitivity Elimination.* PhD thesis, Université Paris VII, France, 1998.
13. T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76(2–3):95–120, 1988.
14. N. Dershowitz and C. Hoot. Natural termination. *Theoretical Computer Science*, 142(2):179–207, 1995.
15. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, vol. B, chap. 6. North-Holland, 1990.
16. G. Dowek and B. Werner. Proof normalization modulo. In *Proc. of TYPES'98*, LNCS 1657.
17. J. Giesl. Termination of nested and mutually recursive algorithms. *Journal of Automated Reasoning*, 19(1):1–29, 1997.
18. E. Giménez. Structural recursive definitions in type theory. In *Proc. of ICALP'98*, LNCS 1443.
19. R. Harper and J. Mitchell. Parametricity and variants of Girard's J operator. *Information Processing Letters*, 70:1–5, 1999.
20. J. Hughes, L. Pareto, and A. Sabry. Proving the correctness of reactive systems using sized types. In *Proc. of POPL'96.*
21. J.-P. Jouannaud and A. Rubio. The Higher-Order Recursive Path Ordering. In *Proc. of LICS'99.*
22. J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Comp. Science*, 121:279–308, 1993.
23. N. P. Mendler. *Inductive Definition in Type Theory.* PhD thesis, Cornell University, United States, 1987.
24. M. Stefanova. *Properties of Typing Systems.* PhD thesis, Katholiecke Universiteit Nijmegen, The Netherlands, 1998.
25. D. Walukiewicz-Chrzaszcz. Termination of rewriting in the Calculus of Constructions. *Journal of Functional Programming*, 13(2):339–414, 2003.
26. H. Xi. Dependent types for program termination verification. *Journal of Higher-Order and Symbolic Computation*, 15(1):91–131, 2002.