Computability Closure: Ten Years Later

Frédéric Blanqui

▶ **To cite this version:**

Computability Closure: Ten Years Later

Frédéric Blanqui

INRIA
LORIA⋆, Campus Scientifique, BP 239
54506 Vandoeuvre-lès-Nancy Cedex, France

Abstract. The notion of computability closure has been introduced for proving the termination of higher-order rewriting with first-order matching by Jean-Pierre Jouannaud and Mitsuhiro Okada in a 1997 draft which later served as a basis for the author's PhD. In this paper, we show how this notion can also be used for dealing with β-normalized rewriting with matching modulo $\beta\eta$ (on patterns *à la* Miller), rewriting with matching modulo some equational theory, and higher-order data types (types with constructors having functional recursive arguments). Finally, we show how the computability closure can easily be turned into a reduction ordering which, in the higher-order case, contains Jean-Pierre Jouannaud and Albert Rubio's higher-order recursive path ordering and, in the first-order case, is equal to the usual first-order recursive path ordering.

1 Introduction

After Jan Willem Klop's PhD thesis on Combinatory Reduction Systems (CRS) [28, 29], the interest in higher-order rewriting, or the combination of λ-calculus and rewriting, was relaunched by Dale Miller and Gopalan Nadathur's work on λ-Prolog [38] and Val Breazu-Tannen's paper on the modularity of confluence for the combination of simply-typed λ-calculus and first-order rewriting [10, 13]. A year later, Dale Miller proved the decidability of unification modulo $\beta\eta$ for "higher-order patterns" [36, 37], and the modularity of termination for simply-typed λ-calculus and first-order rewriting was independently proved by Jean Gallier and Val Breazu-Tannen [11, 12] and Mitsuhiro Okada [40], both using Jean-Yves Girard's technique of reducibility predicates [18–20]. A little bit later, Daniel Dougherty showed, by purely syntactic means (without using reducibility predicates), that these results could be extended to any "stable" set of untyped λ-terms [16, 17], the set of simply-typed λ-terms being stable. We must also mention Zhurab Khasidashvili's new approach to higher-order rewriting with his Expression Reduction Systems (ERS) [27].

Then, in 1991, two important papers were published on this subject, both introducing a new approach to higher-order rewriting: Tobias Nipkow's Higher-order Rewrite Systems (HRS) [39, 33], and Jean-Pierre Jouannaud and Mitsuhiro Okada's Executable Higher-Order Algebraic Specification Languages [22,

---

⋆ UMR 7503 CNRS-INPL-INRIA-Nancy2-UHP

23]. Tobias Nipkow's approach is based on Dale Miller's result: the simply-typed λ-calculus, which is confluent and terminating, is used as a framework for encoding higher-order rewriting. He extends to this framework the Critical Pair Lemma. Jean-Pierre Jouannaud and Mitsuhiro Okada's approach can be seen as a typed version of CRS's (restricted to first-order matching). They proved that termination is modular for the combination of simply-typed λ-calculus, a non-duplicating[1] terminating first-order rewrite system, and an higher-order rewrite system which definition follows a "general schema" extending primitive recursion. Later, Vincent van Oostrom and Femke van Raamsdonk compared CRS's and HRS's [46] and developed an axiomatized framework subsuming them [47, 49].

The combination of β-reduction and rewriting is naturally used in dependent type systems and proof assistants implementing the proposition-as-type and proof-as-object paradigm [6]. In these systems, two propositions equivalent modulo β-reduction and rewriting are considered as equivalent (*e.g.* $P(2+2)$ and $P(4)$). This is essential for enabling users to formalize large proofs with many computations, as recently shown by Georges Gonthier and Benjamin Werner's proof of the Four Color Theorem in the Coq proof assistant. However, checking the correctness of user proofs requires to check the equivalence of two terms. Hence, the necessity to have termination criteria for the combination of β-reduction and a set R of higher-order rewrite rules.

For proving the correctness of the general schema, Jean-Pierre Jouannaud and Mitsuhiro Okada used Jean-Yves Girard's technique of reducibility predicates. Roughly speaking, since proving the (strong) β-normalization by induction on the structure of terms does not work directly, one needs to prove a stronger predicate. In 1967, William Tait introduced a "convertibility predicate" for proving the weak normalization of some extension of Kurt Gödel's system T [43]. Later, in 1971, Jean-Yves Girard introduced "reducibility predicates" (called *computability predicates* in the following) for proving the weak and strong normalization of the polymorphic λ-calculus [18, 19]. This technique can be applied to (higher-order) rewriting by proving that every function symbol is computable, that is, that every function call is computable whenever its arguments so are.

This naturally leads to the following question: which operations preserve computability? Indeed, from a set of such operations, one can define the *computability closure* of a term t, written $CC(t)$, as the set of terms that are computable whenever t so is. Then, to get normalization, it suffices to check that, for every rule $f\boldsymbol{l} \to r \in R$, r belongs to the computability closure of $\boldsymbol{l}$. Examples of computability-preserving operations are: application, function calls on arguments smaller than $\boldsymbol{l}$ in some well-founded ordering $>$, etc. Jean-Pierre Jouannaud and Mitsuhiro Okada introduced this notion in a 1997 draft which served as a basis for [8, 9]. In this paper, we show how this notion can be extended for dealing with β-normalized rewriting with matching modulo $\beta\eta$ on patterns *à la* Miller and matching modulo some equational theory.

---

[1] $l \to r$ is non-duplicating if no variable has more occurrences in r than it has in l.

Another way to prove the termination of R is to find a decidable well-founded rewrite relation containing R. A well known such relation in the first-order case is the recursive path ordering [41, 14] which well-foundedness was initially based on Kruskal theorem [30]. The first attempts made for generalizing this ordering to the higher-order case were not able to orient system T [31, 32, 26]. Finally, in 1999, Jean-Pierre Jouannaud and Albert Rubio succeeded in finding such an ordering [25] by using computability-based techniques again, hence providing the first well-foundedness proof of RPO not based on Kruskal theorem. This ordering was later extended to the calculus of constructions by Daria Walukiewicz [50, 51].

Although the computability closure on one hand, and the recursive path ordering on the other hand, share the same computability-based techniques, there has been no precise comparison between these two termination criteria. In [51], one can find examples of rules that are accepted by one criterion but not the other. And Jean-Pierre Jouannaud and Albert Rubio themselves use the notion of computability closure for strengthening HORPO.

In this paper, we explore the relations between both criteria. We start from the trivial remark that the computability closure itself provides us with an ordering: let t CR($>$) u if $t = f\boldsymbol{t}$ and $u \in \mathrm{CC}_>(\boldsymbol{t})$, where $\mathrm{CC}_>$ is the computability closure built by using a well-founded relation $>$ for comparing the arguments between function calls. Proving the well-foundedness of this ordering simply consists in proving that the computability closure is correct, which can be done by induction on $>$. Then, we remark that the function mapping $>$ to CR($>$) is monotone wrt inclusion. Thus, it admits a least fixpoint which is a well-founded ordering. We prove that this fixpoint contains HORPO and is equal to RPO in the first-order case.

2   Terms and types

We consider simply-typed λ-terms with curried constants. See [2] for details about typed λ-calculus. For rewriting, we follow the notations of Nachum Dershowitz and Jean-Pierre Jouannaud's survey [15].

Let $\mathcal{B}$ be a set of *base types*. The set $\mathbb{T}$ of *simple types* is inductively defined as usual: $T \in \mathbb{T} = B \in \mathcal{B} \mid T \Rightarrow T$.

Let $\mathcal{X}$ be a set of *variables* and $\mathcal{F}$ be a set of *function symbols* disjoint from $\mathcal{X}$. We assume that every $a \in \mathcal{X} \cup \mathcal{F}$ is equipped with a type $\tau_a \in \mathbb{T}$. The sets $\mathcal{T}^T$ of *terms of type T* are inductively defined as follows:

- If $a \in \mathcal{X} \cup \mathcal{F}$, then $a \in \mathcal{T}^{\tau_a}$.
- If $x \in \mathcal{X}$ and $t \in \mathcal{T}^U$, then $\lambda x t \in \mathcal{T}^{\tau_x \Rightarrow U}$.
- If $v \in \mathcal{T}^{T \Rightarrow U}$ and $t \in \mathcal{T}^T$, then $vt \in \mathcal{T}^U$.

As usual, we assume that, for all type T, the set of variables of type T is infinite and consider terms up to α-conversion (type-preserving renaming of bound variables). Let $\mathrm{FV}(t)$ be the set of variables *free* in t. Let $\boldsymbol{t}$ denote a sequence of terms $t_1, \ldots, t_n$ of length $n = |\boldsymbol{t}| \geq 0$.

Let $\tau(t)$ denote the type of a term t. In the following, writing $t : T$ or t^T means that $\tau(t) = T$.

The set $\mathrm{Pos}(t)$ of positions in a term t is defined as usual as words on $\{1, 2\}$. Let $t|_p$ be the subterm of t at position $p \in \mathrm{Pos}(t)$, and $t[u]_p$ be the term obtained by replacing in t its subterm at position $p \in \mathrm{Pos}(t)$ by u.

A term is *algebraic* if it contains no abstraction and no subterm of the form xt. A term t is *linear* if no variable free in t occurs more than once in t.

The β-reduction is the closure by context of the relation $(\lambda xt)u \rightarrow_\beta t_x^u$ where t_x^u denotes the higher-order substitution of x by u in t.

A *rewrite rule* is a pair of terms $l \rightarrow r$ such that l is of the form $f\boldsymbol{l}$, $\mathrm{FV}(r) \subseteq \mathrm{FV}(l)$ and $\tau(l) = \tau(r)$. Given a set R of rewrite rules, let $\rightarrow_R$ be the closure by context and substitution of R. Hence, matching is modulo α-conversion (but α-conversion is needed only for left-hand sides having abstractions). A rule $l \rightarrow r$ is linear (resp. algebraic) if both l and r are linear (resp. algebraic).

Given a relation $\rightarrow$ on terms, let $\leftarrow$, $\rightarrow^=$ and $\rightarrow^*$ be its inverse, its reflexive closure and its reflexive and transitive closure respectively. Let also $\rightarrow(t) = \{t' \in \mathcal{T} \mid t \rightarrow t'\}$ be the set of reducts of t, and $\mathrm{SN}(\rightarrow)$ (resp. $\mathrm{SN}^T(\rightarrow)$) be the set of terms (resp. of type T) that are strongly normalizable wrt $\rightarrow$. Our aim is to prove the termination (strong normalization, well-foundedness) of $\rightarrow = \rightarrow_\beta \cup \rightarrow_R$.

Given a relation $>$, let $>_{\mathrm{lex}}$, $>_{\mathrm{mul}}$ and $>_{\mathrm{prod}}$ respectively denote the lexicographic, multiset and product extensions of $>$. Note that all these extensions are well-founded whenever $>$ is well-founded.

3 Computability

In this section, we remind the notion of computability predicate introduced by William Tait [43, 44] and extended by Jean-Yves Girard with the notion of *neutral[2] term* [19, 20]. Every type is interpreted by a set of *computable* terms of that type. Since computability is defined so as to imply strong normalization, the latter is obtained by proving that every term is computable.

In the following, we assume given a set R of rewrite rules.

Definition 1 (Reducibility candidates). *A term is* neutral *if it is of the form $x\boldsymbol{v}$ or of the form $(\lambda xt)u\boldsymbol{v}$. Let $\rightarrow = \rightarrow_\beta \cup \rightarrow_R$. A reducibility candidate for the type T is a set P of terms such that:*

(1) $P \subseteq \mathrm{SN}^T(\rightarrow)$.

(2) P is stable by $\rightarrow$: $\rightarrow(P) \subseteq P$.

(3) If $t : T$ is neutral and $\rightarrow(t) \subseteq P$, then $t \in P$.

Let $\mathcal{Q}_R^T$ be the set of all reducibility candidates for the type T, and $\mathcal{I}_R$ be the set of functions I from $\mathcal{B}$ to $2^{\mathcal{T}}$ such that, for all $\mathsf{B} \in \mathcal{B}$, $I(\mathsf{B}) \in \mathcal{Q}_R^{\mathsf{B}}$. Given an interpretation of base types $I \in \mathcal{I}_R$, we define an interpretation $[\![T]\!]_R^I \in \mathcal{Q}_R^T$ for every type T as follows:

– $[\![\mathsf{B}]\!]_R^I = I(\mathsf{B})$,

---

[2] simple in [19].

$- \ [\![T \Rightarrow U]\!]_R^I = \{ v \in \mathrm{SN}^{T \Rightarrow U} \mid \forall t \in [\![T]\!]_R^I, \ vt \in [\![U]\!]_R^I \}.$

One can check that SN^T is a reducibility candidate for T.

We now check that the interpretation of a type is a reducibility candidate.

Lemma 1. *If $I \in \mathcal{I}_R$ then, for all type T, $[\![T]\!]_R^I \in \mathcal{Q}_R^T$.*

Proof. We proceed by induction on T. The lemma is immediate for $T \in \mathcal{B}$. Assume now that $[\![T]\!]_R^I \in \mathcal{Q}_R^T$ and $[\![U]\!]_R^I \in \mathcal{Q}_R^U$. We prove that $[\![T \Rightarrow U]\!]_R^I \in \mathcal{Q}_R^{T \Rightarrow U}$.

(1) $[\![T \Rightarrow U]\!]_R^I \subseteq \mathrm{SN}^{T \Rightarrow U}$ by definition.

(2) Let $v \in [\![T \Rightarrow U]\!]_R^I$, $v' \in \to(v)$ and $t \in [\![T]\!]_R^I$. We must prove that $v't \in [\![U]\!]_R^I$. This follows from the facts that $[\![U]\!]_R^I \in \mathcal{Q}_R^U$, $vt \in [\![U]\!]_R^I$ and $v't \in \to(vt)$.

(3) Let $v^{T \Rightarrow U}$ be a neutral term such that $\to(v) \subseteq [\![T \Rightarrow U]\!]_R^I$ and $t \in [\![T]\!]_R^I$. We must prove that $vt \in [\![U]\!]_R^I$. Since v is neutral, vt is neutral too. Since $[\![U]\!]_R^I \in \mathcal{Q}_R^U$, it suffices to prove that $\to(vt) \subseteq [\![U]\!]_R^I$. Since $[\![T]\!]_R^I \in \mathcal{Q}_R^T$, $t \in \mathrm{SN}$ and we can proceed by induction on t with $\to$ as well-founded ordering. Let $w \in \to(vt)$. Since v is neutral, either $w = v't$ with $v' \in \to(v)$, or $w = vt'$ with $t' \in \to(t)$. In the former case, $w \in [\![U]\!]_R^I$ since $v' \in [\![T \Rightarrow U]\!]_R^I$. In the latter case, we conclude by induction hypothesis on t'. $\qquad\square$

Finally, we come to the definition of computability.

Definition 2 (Computability). *Let I be the base type interpretation such that $I(\mathsf{B}) = \mathrm{SN}^{\mathsf{B}}$. A term $t : T$ is* computable *if $t \in [\![T]\!]_R^I$.*

In the following, we drop the superscript I in $[\![T]\!]_R^I$.

We do not know how to prove that computability is stable by subterm before proving that every term is computable. However, since, on base types, computability is equivalent to strong normalization, the subterms of base type of a computable term are computable. This is in particular the case for the arguments of base type of a function symbol:

Definition 3 (Accessibility). *For all $f : \boldsymbol{T} \Rightarrow \mathsf{B}$, let $\mathrm{Acc}(f) = \{ i \mid T_i \in \mathcal{B} \}$ be the set of* accessible *arguments of f.*

We now prove some properties of computable terms.

Lemma 2 (Computability properties).

(C1) If t, u and t_x^u are computable, then $(\lambda x t)u$ is computable.

(C2) If every symbol is computable, then every term is computable.

(C3) If $f\boldsymbol{t}$ is computable and $i \in \mathrm{Acc}(f)$, then t_i is computable.

(C4) A term $f\boldsymbol{t} : \mathsf{B}$ is computable whenever $\boldsymbol{t}$ are computable and every head-reduct of $f\boldsymbol{t}$ is computable.

(C5) A symbol $f : \boldsymbol{T} \Rightarrow \mathsf{B}$ is computable if every head-reduct of $f\boldsymbol{t}$ is computable whenever $\boldsymbol{t} : \boldsymbol{T}$ are computable.

(C6) A symbol f is computable if, for every rule $f\boldsymbol{l} \to r \in R$ and substitution σ, $r\sigma$ is computable whenever $\boldsymbol{l}\sigma$ are computable.

Proof. (C1) Since $(\lambda xt)u$ is neutral, it suffices to prove that every reduct is computable. We proceed by induction on (t, u) with $\to_{\mathrm{prod}}$ as well-founded ordering (t and u are computable). Assume that $(\lambda xt)u \to v$. If $v = t_x^u$, then t' is computable by assumption. Otherwise, $v = (\lambda xt')u$ with $t \to t'$, or $v = (\lambda xt)u'$ with $u \to u'$. In both cases, we can conclude by induction hypothesis.

(C2) First note that the identity substitution is computable since variables are computable (they are neutral and irreducible). We then prove that, for every term t and computable substitution θ, $t\theta$ is computable, by induction on t.

- Assume that $t = f \in \mathcal{F}$. Then, $t\theta = f$ is computable by assumption.
- Assume that $t = x \in \mathcal{X}$. Then, $t\theta = x\theta$ is computable by assumption.
- Assume that $t = \lambda xu$. Then, $t\theta = \lambda xu\theta$. Let $v : V$ computable. We must prove that $t\theta v$ is computable. By induction hypothesis, $u\theta_x^v$ is computable. Since $u\theta$ and v are computable too, by (C1), $t\theta$ is computable.
- Assume that $t = u^{V\Rightarrow T}v$. Then, $t\theta = u\theta v\theta$. By induction hypothesis, $u\theta$ and $v\theta$ are computable. Thus, $t\theta$ is computable.

(C3) By definition of the interpretation of base types.

(C4) By definition of the interpretation of base types, it suffices to prove that every reduct of $f\boldsymbol{t}$ is computable. We prove it by induction on $\boldsymbol{t}$ with $\to_{\mathrm{prod}}$ as well-founded ordering ($\boldsymbol{t}$ are computable). Head-reducts are computable by assumption. For non-head-reducts, this follows by induction hypothesis.

(C5) By definition of the interpretation of arrow types and (C4).

(C6) After (C5), it suffices to prove that every head-reduct of $f\boldsymbol{t}$ is computable whenever $\boldsymbol{t}$ are computable. Let t' be a head-reduct of $f\boldsymbol{t}$. Then, there is $l \to r \in R$ and σ such that $\boldsymbol{t} = \boldsymbol{l}\sigma$ and $t' = r\sigma$. Thus, t' is computable. $\square$

4 Computability closure

After the properties (C2) and (C6), we are left to prove that, for every rule $f\boldsymbol{l} \to r \in R$, $r\sigma$ is computable whenever $\boldsymbol{l}\sigma$ are computable. This naturally leads us to find a set $\mathrm{CC}^f(\boldsymbol{l})$ of terms t such that $t\sigma$ is computable whenever $\boldsymbol{l}\sigma$ are computable: the computability closure of $\boldsymbol{l}$ wrt f.

We can include $\boldsymbol{l}$ and close this set with computability-preserving operations like applying a term to another or taking the accessible argument of a function call.

We can also include variables distinct from $\mathrm{FV}(\boldsymbol{l})$ and allow abstraction on them by strengthening the property to prove as follows: for all $t \in \mathrm{CC}^f(\boldsymbol{l})$, $t\sigma$ is computable whenever $\boldsymbol{l}\sigma$ are computable and σ is computable on $\mathrm{FV}(t) \setminus \mathrm{FV}(l)$.

Now, to allow function calls, the idea is to introduce a precedence on function symbols and a well-founded ordering $>$ on function arguments.

So, we assume given a quasi-ordering $\geq_{\mathcal{F}}$ on $\mathcal{F}$ which strict part $>_{\mathcal{F}} = \geq_{\mathcal{F}} \setminus \leq_{\mathcal{F}}$ is well-founded. Let $\simeq_{\mathcal{F}} = \geq_{\mathcal{F}} \cap \leq_{\mathcal{F}}$ be its associated equivalence relation.

We also assume that every symbol f is equipped with a *status* $\text{stat}_f \in \{\text{lex}, \text{mul}\}$, such that $\text{stat}_f = \text{stat}_g$ whenever $f \simeq_{\mathcal{F}} g$, defining how the arguments of f must be compared: lexicographically (from left to right, or from right to left) or by multiset.

Definition 4 (Status relation). *The* status relation *associated to a relation $>$ is the relation $(f, \boldsymbol{t}) >_{\text{stat}} (g, \boldsymbol{u})$ such that $f >_{\mathcal{F}} g$ or $f \simeq_{\mathcal{F}} g$ and $\boldsymbol{t} >_{\text{stat}_f} \boldsymbol{u}$.*

Note that the status relation $>_{\text{stat}}$ is well-founded whenever $>$ so is.
We now formalize the notion of computability closure.

Definition 5. *A function* CC *mapping every $f^{\boldsymbol{T} \Rightarrow \mathsf{B}}$ and $\boldsymbol{l}^{\boldsymbol{T}}$ to a set of terms $\text{CC}^f(\boldsymbol{l})$ is a* computability closure *if, for all $f^{\boldsymbol{T} \Rightarrow \mathsf{B}}$, $\boldsymbol{l}^{\boldsymbol{T}}$, $r \in \text{CC}^f(\boldsymbol{l})$ and θ, $r\theta$ is computable whenever $\boldsymbol{l}\theta$ are computable and θ is computable on $\mathcal{X} \setminus \text{FV}(\boldsymbol{l})$.*

We now check that the computability of symbols, hence the termination of $\to_\beta \cup \to_R$ by (C2), can be obtained by using a computability closure.

Lemma 3. *If CC is a computability closure and, for all rule $f\boldsymbol{l} \to r \in R$, $r \in \text{CC}^f(\boldsymbol{l})$, then every symbol is computable.*

Proof. It follows from (C6) and the fact that $\text{FV}(r) \subseteq \text{FV}(\boldsymbol{l})$. $\qquad\qquad\square$

Fig. 1. Higher-order computability closure

$$(\text{arg}) \quad l_i \in \text{CC}^f_>(\boldsymbol{l})$$

$$(\text{decomp-symb}) \quad \frac{g\boldsymbol{u} \in \text{CC}^f_>(\boldsymbol{l}) \quad i \in \text{Acc}(g)}{u_i \in \text{CC}^f_>(\boldsymbol{l})}$$

$$(\text{prec}) \quad \frac{f >_{\mathcal{F}} g}{g \in \text{CC}^f_>(\boldsymbol{l})}$$

$$(\text{call}) \quad \frac{f \simeq_{\mathcal{F}} g^{\boldsymbol{U} \Rightarrow U} \quad \boldsymbol{u}^{\boldsymbol{U}} \in \text{CC}^f_>(\boldsymbol{l}) \quad \boldsymbol{l} >^{f\boldsymbol{l}}_{\text{stat}_f} \boldsymbol{u}}{g\boldsymbol{u} \in \text{CC}^f_>(\boldsymbol{l})}$$

$$(\text{app}) \quad \frac{u^{V \Rightarrow T} \in \text{CC}^f_>(\boldsymbol{l}) \quad v^V \in \text{CC}^f_>(\boldsymbol{l})}{uv \in \text{CC}^f_>(\boldsymbol{l})}$$

$$(\text{var}) \quad \frac{x \notin \text{FV}(\boldsymbol{l})}{x \in \text{CC}^f_>(\boldsymbol{l})}$$

$$(\text{lam}) \quad \frac{u \in \text{CC}^f_>(\boldsymbol{l}) \quad x \notin \text{FV}(\boldsymbol{l})}{\lambda x u \in \text{CC}^f_>(\boldsymbol{l})}$$

We now present a computability closure similar to the one introduced in [8, 9] except that the relation $>$ used for comparing arguments in recursive calls is replaced by an abstract family of relations $(>^l)_{l \in \mathcal{T}}$. We then prove the correctness of this abstract computability closure under some condition.

Definition 6 (Closure-compatibility). *A relation $\succ$ is closure-compatible with a family of relations $(>^l)_{l \in \mathcal{T}}$ if, for all l and θ, $t\theta \succ u\theta$ whenever $t >^l u$, $t\theta$ and $u\theta$ are computable, and θ is computable on $\mathcal{X} \setminus \mathrm{FV}(l)$.*

Note that any relation stable by substitution $>$ is closure-compatible with itself (the constant family equal to $>$). This is in particular the case of the restriction of the subterm ordering $>$ defined by $t > u$ if u is a subterm of t and $\mathrm{FV}(u) \subseteq \mathrm{FV}(t)$.

Lemma 4. *Let $> = (>^l)_{l \in \mathcal{T}}$ be a family of relations. The function $\mathrm{CC}_>$ defined in Figure 1 is a computability closure whenever there exists a well-founded relation on computable terms $\succ$ that is closure-compatible with $>$.*

Proof. We proceed by induction, first on $(f, \boldsymbol{l}\theta)$ with $\succ_{\mathrm{stat}}$ as well-founded ordering (H1), and second, by induction on $\mathrm{CC}_>^f(\boldsymbol{l})$ (H2).

(arg) $l_i\theta$ is computable by assumption.

(decomp-symb) By (H2), $g\boldsymbol{u}\theta$ is computable. Thus, after (C3), $u_i\theta$ is computable.

(prec) By (H1), g is computable.

(call) By (H2), $\boldsymbol{u}\theta$ are computable. Since $\boldsymbol{l} >^{f\boldsymbol{l}}_{\mathrm{stat}_f} \boldsymbol{u}$, $\succ$ is closure-compatible with $>$, $\boldsymbol{l}\theta$ and $\boldsymbol{u}\theta$ are computable, and θ is computable on $\mathcal{X} \setminus \mathrm{FV}(\boldsymbol{l})$, we have $\boldsymbol{l}\theta \succ_{\mathrm{stat}_f} \boldsymbol{u}\theta$. Therefore, by (H1), $g\boldsymbol{u}\theta$ is computable.

(app) By (H2), $u\theta$ and $v\theta$ are computable. Thus, $u\theta v\theta$ is computable.

(var) Since $x \in \mathcal{X} \setminus \mathrm{FV}(\boldsymbol{l})$, $x\theta$ is computable by assumption.

(lam) Wlog we can assume that $x \notin \mathrm{codom}(\theta)$. Thus, $(\lambda x u)\theta = \lambda x u\theta$. Let $v : \tau_x$ computable. After (C1), $(\lambda x u\theta)v$ is computable if $u\theta$, v and $u\theta_x^v$ are computable. We have v computable by assumption and $u\theta$ and $u\theta_x^v$ computable by (H2). $\qquad \square$

5 β-normalized rewriting with matching modulo $\beta\eta$

In this section, we show how the notion of computability closure can be extended to deal with HRS's [39]. This extends our previous results on CRS's and HRS's [5]. This computability closure approach seems simpler than the technique of "neutralization" introduced by Jean-Pierre Jouannaud and Albert Rubio in [24]. However, the comparison between both approaches remains to be done.

In HRS's, rewrite rules are of base type, rule left-hand sides are patterns *à la* Miller [37], and rewriting is defined on terms in β-normal η-long form as follows: $t \Rightarrow_R u$ if there are $p \in \mathrm{Pos}(t)$, $l \to r \in R$ and σ in β-normal η-long form such that $t|_p = l\sigma{\downarrow}_\beta{\uparrow}_\eta$ and $u = t[r\sigma{\downarrow}_\beta{\uparrow}_\eta]_p$.

We are going to consider a slightly more general notion of rewriting: β-*normalized rewriting with matching modulo $\beta\eta$*, defined as follows: $t \rightarrow_{R,\beta\eta} u$ if there are $p \in \mathrm{Pos}(t)$, $l \rightarrow r \in R$ and σ in β-normal form such that $t|_p$ is in β-normal form, $t|_p =_{\beta\eta} l\sigma$ and $u = t[r\sigma]_p$. Furthermore, we do not assume that rules are of base type. However, in this case, one can check that, on terms in β-normal η-long form, $\Rightarrow_R \,\subseteq\, \rightarrow_{R,\beta\eta}\rightarrow_\beta^*$.

Matching modulo $\beta\eta$ is necessary when a rule left-hand side contains abstractions. Consider for instance the left-hand side $l = D\lambda x(\sin(Fx))$. With matching modulo α-conversion only, the term $t = D\lambda x(\sin u)$ matches p only if u is of the form vx. In particular, $D\lambda x(\sin x)$ does not match p. Yet, if one substitutes F by λxu in l, then one gets $D(\lambda x(\sin((\lambda xu)x)))$ which β-reduces to t.

Take now $l = D\lambda x(Fx)$. With matching modulo α-conversion only, the term $t = Du$ matches l only if u is of the form λxv. In particular, $(D\sin)$ does not match l. Yet, if one substitutes F by u in l, then one gets $D\lambda x(ux)$ which η-reduces to t since $x \notin \mathrm{FV}(u)$ (by definition of higher-order substitution).

Higher-order patterns are terms in β-normal η-long form which free variables are applied to terms η-equivalent to distinct bound variables. Hence, if l is a pattern, t and σ are in β-normal form and $l\sigma =_{\beta\eta} t$, then $l\sigma \rightarrow_{\beta_0}^* =_\eta t$, where $\rightarrow_{\beta_0}$ is the restriction of $\rightarrow_\beta$ to redexes of the form $(\lambda xt)x$, that is, $(\lambda xt)x \rightarrow_{\beta_0} t$ [37].

Now, for proving the termination of $\rightarrow_\beta \cup \rightarrow_{R,\beta\eta}$ (hence the termination of the HRS rewrite relation $\Rightarrow_R$), it suffices to adapt the notion of computability by replacing $\rightarrow_R$ by $\rightarrow_{R,\beta\eta}$. One can check that all the proofs of the computability properties are still valid except the one for (C6) for which we give a new proof:

Lemma 5 (C6). *A symbol f is computable if, for every rule $fl \rightarrow r \in R$ and substitution σ, $r\sigma$ is computable whenever $l\sigma$ are computable.*

Proof. After (C5), for proving that $f : \boldsymbol{T} \Rightarrow \mathsf{B}$ is computable, it suffices to prove that every head-reduct of $f\boldsymbol{t}$ is computable whenever $\boldsymbol{t} : \boldsymbol{T}$ are computable. Let t' be a head-reduct of $f\boldsymbol{t}$. Then, $f\boldsymbol{t}$ is in β-normal form and there are $fl \rightarrow r \in R$ and σ such that $fl\sigma \leftarrow_{\beta_0}^* =_\eta f\boldsymbol{t}$ and $t' = r\sigma$. To conclude, it suffices to check that $l\sigma$ are computable.

To this end, we prove that computability is preserved by η-reduction, η-expansion and β_0-expansion. Let t be a computable term and let u be a term obtained from t by η-reduction, η-expansion or β_0-expansion. We prove that u is computable when u is of base type. If u is not of base type then, by applying it to computable terms of appropriate types, we get a term of base type. On base types, computability is equivalent to strong normalization. Thus, it suffices to prove that every reduct of u is strongly normalizable. In each case, we proceed by induction on t with $\rightarrow$ as well-founded ordering (t is computable).

- β_0-expansion: $t \leftarrow_{\beta_0} u$. If $u \rightarrow_\beta u'$ then either $u' = t$ is computable or, by confluence of β and since β_0 makes no duplication, there is t' such that $t \rightarrow_\beta t' \leftarrow_{\beta_0}^* u'$. Now, if $u \rightarrow_R u'$ then, since R-redexes are in β-normal form, the β_0-redex is either above the R-redex or at a disjoint position. Thus, there is u' such that $t \rightarrow_R t' \leftarrow_{\beta 0} u'$. In both cases, we can conclude by induction hypothesis.

– η-reduction: $t \to_\eta u$. If $u \to_\beta u'$ then, by postponement of η wrt β ($\to_\eta \to_\beta \subseteq \to_\beta^+ \to_\eta^*$), there is t' such that $t \to_\beta^+ t' \to_\eta^* u'$. Now, if $u \to_R u'$ then, since R-redexes are in β-normal form, either the η-redex is a β-redex and $t \to_\beta u \to_R t' = u'$, or there is t' such that $t \to_R t' \to_\eta^* u'$. In both cases, we can conclude by induction hypothesis.

– η-expansion: $t \leftarrow_\eta u$. If $u \to_\beta u'$ then either $u' = t$ is computable or, by confluence of $\beta\eta$, there is t' such that $t \to_\beta t' \leftarrow_\eta^* u'$. Now, if $u \to_R u'$ then, since R-redexes are in β-normal form, there is t' such that $t \to_R t' \leftarrow_\eta^* u'$. In both cases, we can conclude by induction hypothesis. □

By property (C2) and Lemma 4, it follows that $\to \; = \; \to_\beta \cup \to_{R,\beta\eta}$ is well-founded if, for all rule $f\boldsymbol{l} \to r \in R$, $r \in \mathrm{CC}_>^f(\boldsymbol{l})$.

Fig. 2. Decomposition rules for higher-order patterns

$$(\text{decomp-lam}) \quad \frac{\lambda y u \in \mathrm{CC}_>^f(\boldsymbol{l}) \quad y \notin \mathrm{FV}(\boldsymbol{l})}{u \in \mathrm{CC}_>^f(\boldsymbol{l})}$$

$$(\text{decomp-app-left}) \quad \frac{u y \in \mathrm{CC}_>^f(\boldsymbol{l}) \quad y \notin \mathrm{FV}(\boldsymbol{l}) \cup \mathrm{FV}(u)}{u \in \mathrm{CC}_>^f(\boldsymbol{l})}$$

Now, for dealing with patterns *à la* Miller, we also need to add new decomposition rules in the computability closure.

Lemma 6. *The function* $\mathrm{CC}_>$ *defined by the rules of Figure 1 and 2 is a computability closure whenever there exists a well-founded relation on computable terms that is closure-compatible with* $>$.

Proof. We extend the proof of Lemma 4 with the new decomposition rules.

(decomp-lam) Let θ' be the restriction of θ to $\mathrm{dom}(\theta) \backslash \{y\}$. Wlog, we can assume that $y \notin \mathrm{codom}(\theta)$. Hence, $(\lambda y u)\theta' = \lambda y u \theta'$. Now, since $\mathrm{dom}(\theta) \subseteq \mathrm{FV}(u) \backslash \mathrm{FV}(\boldsymbol{l})$, $\mathrm{dom}(\theta') \subseteq \mathrm{FV}(\lambda y u) \backslash \mathrm{FV}(\boldsymbol{l})$. Thus, by (H2), $\lambda y u \theta'$ is computable. Since $y\theta$ is computable, $(\lambda y u\theta')y\theta$ is computable. Thus, by β-reduction, $u\theta'^{y\theta}_y$ is computable too. Finally, since $y \notin \mathrm{dom}(\theta') \cup \mathrm{codom}(\theta')$, $u\theta'^{y\theta}_y = u\theta$.

(decomp-app-left) Let $v : \tau_y$ computable. Since $\mathrm{dom}(\theta) \subseteq \mathrm{FV}(u) \backslash \mathrm{FV}(\boldsymbol{l})$ and $y \notin \mathrm{FV}(\boldsymbol{l})$, $\mathrm{dom}(\theta_y^v) = \mathrm{dom}(\theta) \cup \{y\} \subseteq \mathrm{FV}(uy) \backslash \mathrm{FV}(\boldsymbol{l})$. Thus, by (H2), $(uy)\theta_y^v = u\theta_y^v v$ is computable. Since $y \notin \mathrm{FV}(u)$, $u\theta_y^v = u\theta$. Thus, $u\theta$ is computable. □

6 Matching modulo some equational theory

In this section, we show how the notion of computability closure can be used for proving the termination of the combination of β-reduction and rewriting with matching modulo some equational theory E [48, 21].

To this end, we assume that E is a symmetric set of *rules*, that is, $l \to r \in E$ iff $r \to l$ in E. By definition of rewrite rules (see Section 2), this implies that, for all $l \to r \in E$, r is of the form $g\boldsymbol{r}$ and $\mathrm{FV}(l) = \mathrm{FV}(r)$. This includes associativity and commutativity but excludes collapsing rules like $x + 0 \to x$ and erasing rules like $x \times 0 \to 0$.

Then, rewriting with matching modulo can be defined as follow: $t \to_{R,E} u$ if there are $p \in \mathrm{Pos}(t)$, $l \to r \in R$ and σ such that $t|_p \to_E^* l\sigma$ and $u = t[r\sigma]_p$.

Rewriting with matching modulo E is different from rewriting modulo E which is $\to_E^* \to_R$. The point is that, with matching modulo E, no E-step takes place above $t|_p$ when one rewrites a term t at some position $p \in \mathrm{Pos}(t)$.

Hence, correct an error in [4] (Theorem 6) where it is claimed that $\to_\beta \cup \to_E^* \to_R$ is terminating. What is in fact proved in [4] is the termination of $\to_\beta \cup \to_{E_1}^* \to_{R_1} \cup \to_{R_\omega, E_\omega}$ where E_1 and R_1 (resp. E_ω and R_ω) are the first-order (resp. higher-order) parts of E and R respectively.

For proving the termination of $\to_\beta \cup \to_{R,E}$, it suffices to adapt computability by replacing $\to_R$ by $\to_{R,E}$. One can check that all the proofs of computability properties are still valid except the one for (C6) for which we give a new proof:

Lemma 7 (C6). *Let E be a symmetric set of* rules. *Assume that $\succ$ is a well-founded relation on computable terms closure-compatible with $>$ and that, for all rule $f\boldsymbol{l} \to g\boldsymbol{r} \in E$, $\boldsymbol{r} \in \mathrm{CC}_>^f(\boldsymbol{l})$. Then, f is computable if, for every rule $f\boldsymbol{l} \to r \in R$ and substitution σ, $r\sigma$ is computable whenever $\boldsymbol{l}\sigma$ are computable.*

Proof. By Lemma 4, $\mathrm{CC}_>$ is a computability closure. After (C5), for proving that $f : \boldsymbol{T} \Rightarrow \mathsf{B}$ is computable, it suffices to prove that every head-reduct of $f\boldsymbol{t}$ is computable whenever $\boldsymbol{t} : \boldsymbol{T}$ are computable. Let t' be a head-reduct of $f\boldsymbol{t}$. Then, there is $g\boldsymbol{l} \to r \in R$ and σ such that $f\boldsymbol{t} \to_E^* g\boldsymbol{l}\sigma$ and $t' = r\sigma$. By definition of computability closure, $\boldsymbol{l}\sigma$ are computable since $\boldsymbol{t}$ are computable (induction on the number of E-steps). Therefore, $r\sigma$ is computable. $\qquad\square$

By property (C2) and Lemma 4, it follows that $\to \; = \; \to_\beta \cup \to_{R,E}$ is well-founded if moreover, for all rule $f\boldsymbol{l} \to r \in R$, $r \in \mathrm{CC}_>^f(\boldsymbol{l})$.

7 Higher-order data types

Until now, we used the subterm ordering in (call). But this ordering is not strong enough to handle recursive definitions on higher-order data types, *i.e.* data types with constructors having functional recursive arguments. Consider for instance a type P representing processes with a sequence operator $; : \mathsf{P} \Rightarrow \mathsf{P} \Rightarrow \mathsf{P}$ and a data-dependent choice operator $\Sigma : (\mathsf{D} \Rightarrow \mathsf{P}) \Rightarrow \mathsf{P}$. Then, in the following simplification rule [45]:

$$(\Sigma P); x \to \Sigma \lambda y (Py; x)$$

the term Py is not a subterm of ΣP.

In this section, we describe an extension of the computability closure to handle such definitions. It is based on the interpretation of "positive" higher-order data types introduced by Nax Paul Mendler in 1987 [34, 35].

As usual, the set $\mathrm{Pos}(T)$ of *positions in a type T* is defined as words on $\{1, 2\}$. The sets $\mathrm{Pos}^+(T)$ and $\mathrm{Pos}^-(T)$ of *positive and negative positions* respectively are inductively defined as follows:

- $\mathrm{Pos}^\delta(\mathsf{B}) = \{\varepsilon\}$.
- $\mathrm{Pos}^\delta(T \Rightarrow U) = 1 \cdot \mathrm{Pos}^{-\delta}(T) \cup 2 \cdot \mathrm{Pos}^\delta(U)$.

Let $\mathrm{Pos}(\mathsf{B}, T)$ be the positions of the occurrences of B in T. A base type B *occurs only positively* (resp. *negatively*) in a type T if $\mathrm{Pos}(\mathsf{B}, T) \subseteq \mathrm{Pos}^+(T)$ (resp. $\mathrm{Pos}(\mathsf{B}, T) \subseteq \mathrm{Pos}^-(T)$).

Nax Paul Mendler showed that the combination of β-reduction and reduction rules for a "case" or "match" construction does not terminate if a data type B has a constructor having an argument in the type of which B occurs negatively (we say that B is not positive). Take for instance $c : (\mathsf{B} \Rightarrow \mathsf{N}) \Rightarrow \mathsf{B}$, $f : \mathsf{B} \Rightarrow (\mathsf{B} \Rightarrow \mathsf{N})$ together with the rule $f(cx) \to_R x$. Then, by taking $\omega = \lambda x f x x : \mathsf{B} \Rightarrow \mathsf{N}$, we have $\omega(c\omega) \to_\beta f(c\omega)(c\omega) \to_R \omega(c\omega) \to_\beta \ldots$

He also showed that the set of all reducibility candidates is a complete lattice for inclusion and that, if B is positive, then one can build an interpretation of B as the fixpoint of a monotone functional on reducibility candidates, in which the reduction rules for the case construction are safe. In this case, we can say that every argument of a constructor is accessible. We extend this notion of accessibility to every (defined or undefined) function symbol as follows.

Definition 7 (Accessible arguments). *For every $f^{\boldsymbol{T} \Rightarrow \mathsf{B}} \in \mathcal{F}$, let $\mathrm{Acc}(f) = \{i \le |\boldsymbol{T}| \mid \mathrm{Pos}(\mathsf{B}, T_i) \subseteq \mathrm{Pos}^+(T_i)\}$.*

In our example, we have $\mathrm{Pos}(\mathsf{P}, \mathsf{D} \Rightarrow \mathsf{P}) = \{2\} = \mathrm{Pos}^+(\mathsf{D} \Rightarrow \mathsf{P})$ and $\mathrm{Pos}(\mathsf{P}, \mathsf{P}) = \{\varepsilon\} = \mathrm{Pos}^+(\mathsf{P})$. Thus, $\mathrm{Acc}(\Sigma) = \{1\}$ and $\mathrm{Acc}(;) = \{1, 2\}$.

We now define the functional the least fixpoint of which will provide the interpretation of base types.

Lemma 8. *The function $F_R^I(\mathsf{B}) = \{t \in \mathrm{SN}^B \mid \forall f^{\boldsymbol{T} \Rightarrow \mathsf{B}} \boldsymbol{t}, \, t \to^* f\boldsymbol{t} \Rightarrow \forall i \in \mathrm{Acc}(f), \, t_i \in [\![T_i]\!]_R^I\}$ is a monotone function on $\mathcal{I}_R$.*

Proof. We first prove that $P = F_R^I(\mathsf{B}) \in \mathcal{Q}_R^\mathsf{B}$.

(1) $P \subseteq \mathrm{SN}^\mathsf{B}$ by definition.

(2) Let $t \in P$, $t' \in \to(t)$, $f : \boldsymbol{T} \Rightarrow \mathsf{B}$ and $\boldsymbol{t}$ such that $t' \to^* f\boldsymbol{t}$. We must prove that $\boldsymbol{t} \in [\![\boldsymbol{T}]\!]_R$. It follows from the facts that $t \in P$ and $t \to^* f\boldsymbol{t}$.

(3) Let t^B neutral such that $\to(t) \subseteq P$. Let $f^{\boldsymbol{T} \Rightarrow \mathsf{B}}$, $\boldsymbol{t}$ such that $t \to^* f\boldsymbol{t}$ and $i \in \mathrm{Acc}(f)$. We must prove that $t_i \in [\![T_i]\!]_R$. Since t is neutral, $t \ne f\boldsymbol{t}$. Thus, there is $t' \in \to(t)$ such that $t' \to^* f\boldsymbol{t}$. Since $t' \in P$, $t_i \in [\![T_i]\!]_R$.

For the monotony, let $\le^+ = \le$ and $\le^- = \ge$. Let $I \le J$ iff, for all B, $I(\mathsf{B}) \subseteq J(\mathsf{B})$. We first prove that $[\![T]\!]_R^I \subseteq^\delta [\![T]\!]_R^J$ whenever $I \le J$ and $\mathrm{Pos}(\mathsf{B}, T) \subseteq \mathrm{Pos}^\delta(T)$, by induction on T.

– Assume that $T = C \in \mathcal{B}$. Then, $\delta = +$, $[\![T]\!]_R^I = I(C)$ and $[\![T]\!]_R^I = J(C)$. Since $I(C) \subseteq J(C)$, $[\![T]\!]_R^I \subseteq [\![T]\!]_R^J$.
– Assume that $T = U \Rightarrow V$. Then, $\mathrm{Pos}(\mathsf{B}, U) \subseteq \mathrm{Pos}^{-\delta}(U)$ and $\mathrm{Pos}(\mathsf{B}, V) \subseteq \mathrm{Pos}^{\delta}(V)$. Thus, by induction hypothesis, $[\![U]\!]_R^I \subseteq^{-\delta} [\![U]\!]_R^J$ and $[\![V]\!]_R^I \subseteq^{\delta} [\![V]\!]_R^J$. Assume that $\delta = +$. Let $t \in [\![T]\!]_R^I$ and $u \in [\![U]\!]_R^J$. We must prove that $tu \in [\![V]\!]_R^J$. Since $[\![U]\!]_R^I \supseteq [\![U]\!]_R^J$, $tu \in [\![V]\!]_R^I$. Since $[\![V]\!]_R^I \subseteq [\![V]\!]_R^J$, $tu \in [\![V]\!]_R^J$. It works similarly for $\delta = -$.

Assume now that $I \leq J$. We must prove that, for all B, $F_R^I(\mathsf{B}) \subseteq F_R^J(\mathsf{B})$. Let $\mathsf{B} \in \mathcal{B}$ and $t \in F_R^I(\mathsf{B})$. We must prove that $t \in F_R^J(\mathsf{B})$. First, we have $t \in \mathrm{SN}^{\mathsf{B}}$ since $t \in F_R^I(\mathsf{B})$. Assume now that $t \rightarrow^* f^{\boldsymbol{T} \Rightarrow \mathsf{B}} \boldsymbol{t}$ and let $i \in \mathrm{Acc}(f)$. We must prove that $t_i \in [\![T_i]\!]_R^J$. Since $t \in F_R^I(\mathsf{B})$, $t_i \in [\![T_i]\!]_R^I$. Since $i \in \mathrm{Acc}(f)$, $\mathrm{Pos}(\mathsf{B}, T_i) \subseteq \mathrm{Pos}^+(T_i)$ and $[\![T_i]\!]_R^I \subseteq [\![T_i]\!]_R^J$. $\square$

Definition 8 (Computability). *Let I_R be the least fixpoint of F_R. A term $t : T$ is computable if $t \in [\![T]\!]_R^{I_R}$.*

In the following, we drop the superscript I_R in $[\![T]\!]_R^{I_R}$.

One can check that all the proofs of computability properties are still valid except the one for (C4) for which we give a new proof:

Lemma 9 (C4). *A term $f\boldsymbol{t} : \mathsf{B}$ is computable whenever $\boldsymbol{t}$ are computable and every head-reduct of $f\boldsymbol{t}$ is computable.*

Proof. We first need to prove that $f\boldsymbol{t}$ is SN. This follows from the previous proof of (C4). Assume now that $f\boldsymbol{t} \rightarrow^* g\boldsymbol{u}$ and $i \in \mathrm{Acc}(g)$. We prove that u_i is computable by induction on $\boldsymbol{t}$ with $\rightarrow_{\mathrm{prod}}$ as well-founded ordering ($\boldsymbol{t}$ are computable). If $f\boldsymbol{t} = g\boldsymbol{u}$, then $u_i = t_i$ is computable by assumption. Otherwise, $f\boldsymbol{t} \rightarrow v \rightarrow^* g\boldsymbol{u}$. If v is a head-reduct of $f\boldsymbol{t}$, then v and u_i are computable. Otherwise, we conclude by induction hypothesis. $\square$

The least fixpoint of F_R is reachable by transfinite iteration from the smallest element of $\mathcal{I}_R$. This provides us with an ordering that can handle definitions on higher-order data types.

Definition 9 (Size ordering). *For all $\mathsf{B} \in \mathcal{B}$ and $t \in [\![\mathsf{B}]\!]_R$, let the size of t be the smallest ordinal $o_R^{\mathsf{B}}(t) = \mathfrak{a}$ such that $t \in F_R^{\mathfrak{a}}(\emptyset)(\mathsf{B})$, where $F_R^{\mathfrak{a}}$ is the transfinite $\mathfrak{a}$-iteration of F_R. Let $\succeq_R$ be the union of all the relations $\succeq_R^T$ inductively defined on $[\![T]\!]_R$ as follows:*
– $t \succeq_R^{\mathsf{B}} u$ if $o_R^{\mathsf{B}}(t) \geq o_R^{\mathsf{B}}(u)$.
– $t \succeq_R^{T \Rightarrow U} u$ if, for all $v \in [\![T]\!]_R$, $tv \succeq_R^U uv$.

In our example, we have $[\![\mathsf{P}]\!]_R = \{ t \in \mathrm{SN}^P \mid \forall f^{\boldsymbol{T} \Rightarrow \mathsf{P}} \boldsymbol{t}, t \rightarrow^* f\boldsymbol{t} \Rightarrow \forall i \in \mathrm{Acc}(f), t_i \in [\![T_i]\!]_R \}$. Since $\mathrm{Acc}(\Sigma) = \{1\}$, if $\Sigma P \in [\![\mathsf{P}]\!]_R$ then, for all $d \in [\![\mathsf{D}]\!]_R$, $Pd \in [\![\mathsf{P}]\!]_R$ and $o_R^{\mathsf{P}}(Pd) < o_R^{\mathsf{P}}(\Sigma P)$.

We immediately check that the size ordering is well-founded.

Lemma 10. *$\succeq_R$ is a well-founded quasi-ordering containing $\rightarrow$.*

Proof. The relation $\succeq_R$ is the union of pairwise disjoint relations. Hence, it suffices to prove that each one is transitive and well-founded. We proceed by induction on T. For $T \in \mathcal{B}$, this is immediate. Assume now that $(t_i)_{i \in \mathbb{N}}$ is an increasing sequence for $\succ_R^{T \Rightarrow U}$. Since variables are computable, let $x \in [\![T]\!]_R$. By definition of $\succ_R^{T \Rightarrow U}$, $(t_i x)_{i \in \mathbb{N}}$ is an increasing sequence for $\succ_R^U$. $\qquad\square$

Fig. 3. Accessibility ordering

$$(>\text{base}) \quad \frac{i \in \text{Acc}(g) \quad \boldsymbol{b} \in \mathcal{X} \setminus \text{FV}(l)}{g^{A \Rightarrow B} \boldsymbol{a}^A >^l a_i^{B \Rightarrow B} \boldsymbol{b}^B}$$

$$(>\text{lam}) \quad \frac{a >^l bx \quad x \in \mathcal{X} \setminus (\text{FV}(b) \cup \text{FV}(l))}{\lambda x a > b}$$

$$(>\text{red}) \quad \frac{a >^l b \quad b \to_\beta c}{a >^l c}$$

$$(>\text{trans}) \quad \frac{a >^l b \quad b >^l c}{a >^l c}$$

We now define some relation strong enough for capturing definitions on higher-order data types and with which $\succ_R$ is closure-compatible.

Lemma 11. $\succ_R$ *is closure-compatible with the family* $(>^l)_{l \in \mathcal{T}}$ *defined Figure 3.*

Proof. We prove that $a\theta \succ_R b\theta$ whenever $a >^l b$, $a\theta$ and $b\theta$ are computable, and θ is computable on $\mathcal{X} \setminus \text{FV}(l)$.

($>$base) By definition of I_R, $o_R(g\boldsymbol{a}\theta) = \mathfrak{a} + 1$ and $a_i\theta \in [\![\boldsymbol{B} \Rightarrow \mathsf{B}]\!]_R^{I_R^{\mathfrak{a}}}$. Since $\boldsymbol{b} \in \mathcal{X} \setminus \text{FV}(l)$ and θ is computable on $\mathcal{X} \setminus \text{FV}(l)$, $\boldsymbol{b}\theta$ are computable. Therefore, $a_i\theta\boldsymbol{b}\theta \in I_R^{\mathfrak{a}}(B)$ and $a_R(g\boldsymbol{a}\theta) > \mathfrak{a} \geq o_R(a_i\theta\boldsymbol{b}\theta)$.

($>$lam) Let $w : \tau_x$ computable. Wlog we can assume that $x \notin \text{dom}(\theta) \cup \text{codom}(\theta)$. Hence, $(\lambda x a)\theta = \lambda x a\theta$. We must prove that $(\lambda x a\theta)w \succ_R b\theta w$. By β-reduction, $(\lambda x a\theta)w \succeq_R a\theta_x^w$. By induction hypothesis, $a\theta_x^w \succ_R (bx)\theta_x^w$. Since $x \notin \text{FV}(b) \cup \text{dom}(\theta) \cup \text{codom}(\theta)$, $(bx)\theta_x^w = b\theta w$.

($>$red) By induction hypothesis and since $\to_\beta \subseteq \succeq_R$.

($>$trans) By induction hypothesis and transitivity of $\succ_R$. $\qquad\square$

By property (C2) and Lemma 4, it follows that $\to = \to_\beta \cup \to_R$ is well-founded if, for all rule $f\boldsymbol{l} \to r \in R$, $r \in \text{CC}_>^f(\boldsymbol{l})$.

Note that we could strengthen the definition of $(>^l)_{l \in \mathcal{T}}$ by taking in ($>$base), when $l = f\boldsymbol{l}$, $\boldsymbol{b} \in \text{CC}_>^f(\boldsymbol{l})$ instead of $\boldsymbol{b} \in \mathcal{X} \setminus \text{FV}(\boldsymbol{l})$, making the definitions of $>$ and $\text{CC}_>$ mutually dependent. See [7] for details.

8 The recursive computability ordering

We now show how the computability closure can be turned into a well-founded ordering containing the monomorphic version of Jean-Pierre Jouannaud and Albert Rubio's higher-order recursive path ordering [25].

Indeed, consider the relation $\mathrm{CR}(>) = \{(f\boldsymbol{l}, r) \mid r \in \mathrm{CC}^f_>(\boldsymbol{l}), \mathrm{FV}(r) \subseteq \mathrm{FV}(\boldsymbol{l}), \tau(f\boldsymbol{l}) = \tau(r)\}$ made of all the rules which right-hand side is in the computability closure of its left-hand side. After (C2) and Lemma 3, $\to_\beta \cup \to_{\mathrm{CR}(>)}$ is well-founded whenever $>$ is well-founded and stable by substitution. Hence, $\mathrm{CR}(>)$ is itself well-founded and stable by substitution whenever $>$ is well-founded and stable by substitution.

We now observe that the function mapping $>$ to $\mathrm{CR}(>)$ is monotone wrt inclusion. It has therefore a least fixpoint that is stable by substitution and which closure by context is well-founded when combined with $\to_\beta$.

Lemma 12. *The function mapping $>$ to the relation $\mathrm{CR}(>) = \{(f\boldsymbol{l}, r) \mid r \in \mathrm{CC}^f_>(\boldsymbol{l}), \mathrm{FV}(r) \subseteq \mathrm{FV}(\boldsymbol{l}), \tau(f\boldsymbol{l}) = \tau(r)\}$ is monotone wrt inclusion on the set of well-founded relations stable by substitution.*

Proof. Assume that $>_1 \subseteq >_2$. One can prove by induction on $(f\boldsymbol{l}, r) \in \mathrm{CR}(>_1)$ that $(f\boldsymbol{l}, r) \in \mathrm{CR}(>_2)$. In the (call) case, we use the fact that the function mapping $>$ to $>_{\mathrm{stat}}$ is monotone wrt inclusion.

Now, assume that $>$ is well-founded and stable by substitution. After (C2) and Lemma 3, $\to_\beta \cup \to_{\mathrm{CR}(>)}$ is well-founded. Thus, $\mathrm{CR}(>)$ is well-founded. Now, one can check that $\mathrm{CR}(>)$ is stable by substitution whenever $>$ is stable by substitution. $\square$

Definition 10. *Let the* weak higher-order recursive computability (quasi-) ordering $>_{\mathrm{whorco}}$ *be the least fixpoint of* CR, *and the* higher-order recursive computability (quasi-) ordering $>_{\mathrm{horco}}$ *be the closure by context of* $>_{\mathrm{whorco}}$.

In Figure 4, we give an inductive presentation of $>_{\mathrm{horco}}$ obtained by replacing $u \in \mathrm{CC}^f_>(\boldsymbol{l})$ by $f\boldsymbol{l} > u$ in Figure 1, and adding a rule (cont) for the closure by context and a rule (rule) for the conditions on rules.

Strictly speaking, $>_{\mathrm{horco}}$, like $>_{\mathrm{horpo}}$, is not a quasi-ordering. One needs to take its transitive closure to get a quasi-ordering. On the other hand, one can check that $>_{\mathrm{whorco}}$ is transitive, hence is a true quasi-ordering (note that, if $t >_{\mathrm{whorco}} u$, then t is of the form $f\boldsymbol{t}$).

Moreover, since $>_{\mathrm{whorco}}$ is not closed by context, it is better suited for proving the termination of rewrite systems by using the dependency pair method [1, 42, 3].

We now would like to compare this ordering with the monomorphic version of $>_{\mathrm{horpo}}$ which definition is reminded in Figure 5. To this end, we need to slightly strengthen the definition of computability closure by replacing $>$ by its closure by context $\to_>$, and by adding the following deduction rule:

$$(\mathrm{red}) \quad \frac{u \in \mathrm{CC}^f_>(\boldsymbol{l}) \quad u > v}{v \in \mathrm{CC}^f_>(\boldsymbol{l})}$$

Fig. 4. Higher-order computability ordering

$$(\text{cont}) \quad \frac{t >_{\text{whorco}} u \quad p \in \text{Pos}(C)}{C[t]_p >_{\text{horco}} C[u]_p}$$

$$(\text{rule}) \quad \frac{t^T > u^U \quad \text{FV}(u) \subseteq \text{FV}(t) \quad T = U}{t >_{\text{whorco}} u}$$

$$(\text{arg}) \quad f\boldsymbol{l} > l_i$$

$$(\text{decomp-symb}) \quad \frac{f\boldsymbol{l} > g\boldsymbol{u} \quad i \in \text{Acc}(g)}{f\boldsymbol{l} > u_i}$$

$$(\text{prec}) \quad \frac{f >_{\mathcal{F}} g}{f\boldsymbol{l} > g}$$

$$(\text{call}) \quad \frac{f \simeq_{\mathcal{F}} g^{\boldsymbol{U} \Rightarrow U} \quad f\boldsymbol{l} > \boldsymbol{u}^{\boldsymbol{U}} \quad \boldsymbol{l} \ (>_{\text{whorco}})_{\text{stat}_f} \ \boldsymbol{u}}{f\boldsymbol{l} > g\boldsymbol{u}}$$

$$(\text{app}) \quad \frac{f\boldsymbol{l} > u^{V \Rightarrow T} \quad f\boldsymbol{l} > v^V}{f\boldsymbol{l} > uv}$$

$$(\text{var}) \quad \frac{x \notin \text{FV}(\boldsymbol{l})}{f\boldsymbol{l} > x}$$

$$(\text{lam}) \quad \frac{f\boldsymbol{l} > u \quad x \notin \text{FV}(\boldsymbol{l})}{f\boldsymbol{l} > \lambda xu}$$

One can check that all the properties are preserved. More details can be found in [7]. Hence, we get the following additional deduction rules for $>_{\text{whorco}}$:

$$(\text{call}) \quad \frac{f \simeq_{\mathcal{F}} g^{\boldsymbol{U} \Rightarrow U} \quad f\boldsymbol{l} > \boldsymbol{u}^{\boldsymbol{U}} \quad \boldsymbol{l} \ (>_{\text{horco}})_{\text{stat}_f} \ \boldsymbol{u}}{f\boldsymbol{l} > g\boldsymbol{u}}$$

$$(\text{red}) \quad \frac{f\boldsymbol{l} > u \quad u >_{\text{horco}} v}{f\boldsymbol{l} > v}$$

We now prove that $>_{\text{horpo}}$ is included in the transitive closure of $>_{\text{horco}}$.

Lemma 13. $>_{\text{horpo}} \subseteq >_{\text{horco}}^+$.

Proof. Note that $\text{FV}(u) \subseteq \text{FV}(t)$ and $T = U$ whenever $t^T >_{\text{horpo}} u^U$ ($>_{\text{horpo}}$ is a set of rules).

We first prove the property (*): $f\boldsymbol{t} > v$ whenever $t_j >_{\text{horco}}^* v$ or $f\boldsymbol{t} >_{\text{horco}}^+ v$. Assume that $t_j >_{\text{horco}}^* v$. By (arg), $f\boldsymbol{t} > t_j$. Thus, by (red), $f\boldsymbol{t} > v$. Assume now that $f\boldsymbol{t} >_{\text{horco}} u >_{\text{horco}}^* v$. By (red), it suffices to prove that $f\boldsymbol{t} > u$. There are two cases:

− $f\boldsymbol{t} = f\boldsymbol{a}t_k\boldsymbol{b}$, $u = f\boldsymbol{a}t_k'\boldsymbol{b}$ and $t_k >_{\text{horco}} t_k'$. We conclude by (call).

Fig. 5. HORPO [25]

$$P(f, \boldsymbol{t}, u) = f\boldsymbol{t} >_{\text{horpo}} u \vee (\exists j)\ t_j \geq_{\text{horpo}} u$$

$$(1) \quad \frac{t_i \geq_{\text{horpo}} u}{f^{\boldsymbol{T} \Rightarrow T} \boldsymbol{t}^{\boldsymbol{T}} >_{\text{horpo}} u^T}$$

$$(2) \quad \frac{f >_{\mathcal{F}} g \quad P(f, \boldsymbol{t}, \boldsymbol{u})}{f^{\boldsymbol{T} \Rightarrow T} \boldsymbol{t}^{\boldsymbol{T}} >_{\text{horpo}} g^{\boldsymbol{U} \Rightarrow T} \boldsymbol{u}^{\boldsymbol{U}}}$$

$$(3) \quad \frac{f \simeq_{\mathcal{F}} g \quad \text{stat}_f = \text{mul} \quad \boldsymbol{t}\ (>_{\text{horpo}})_{\text{stat}_f}\ \boldsymbol{u}}{f^{\boldsymbol{T} \Rightarrow T} \boldsymbol{t}^{\boldsymbol{T}} >_{\text{horpo}} g^{\boldsymbol{U} \Rightarrow T} \boldsymbol{u}^{\boldsymbol{U}}}$$

$$(4) \quad \frac{f \simeq_{\mathcal{F}} g \quad \text{stat}_f = \text{lex} \quad \boldsymbol{t}\ (>_{\text{horpo}})_{\text{stat}_f}\ \boldsymbol{u} \quad P(f, \boldsymbol{t}, \boldsymbol{u})}{f^{\boldsymbol{T} \Rightarrow T} \boldsymbol{t}^{\boldsymbol{T}} >_{\text{horpo}} g^{\boldsymbol{U} \Rightarrow T} \boldsymbol{u}^{\boldsymbol{U}}}$$

$$(5) \quad \frac{P(f, \boldsymbol{t}, \boldsymbol{u})}{f^{\boldsymbol{T} \Rightarrow T} \boldsymbol{t} >_{\text{horpo}} \boldsymbol{u}^{T}}$$

$$(6) \quad \frac{\{t_1, t_2\}\ (>_{\text{horpo}})_{\text{mul}}\ \{u_1, u_2\}}{t_1^{U \Rightarrow T} t_2^{U} >_{\text{horpo}} u_1^{V \Rightarrow T} u_2^{V}}$$

$$(7) \quad \frac{t >_{\text{horpo}} u}{\lambda x t >_{\text{horpo}} \lambda x u}$$

– $f\boldsymbol{t} = f\boldsymbol{l}\boldsymbol{b}$, $u = r\boldsymbol{b}$ and $f\boldsymbol{l} >_{\text{whorco}} r$. One can check that $f\boldsymbol{l}t > rt$ whenever $f\boldsymbol{l} > r$.

We now prove the theorem by induction on $>_{\text{horpo}}$.

(1) By induction hypothesis, $t_i >_{\text{horco}}^{*} u$. By (arg), $f\boldsymbol{t} > t_i$. Since $t_i >_{\text{horpo}} u$ and $f\boldsymbol{t} >_{\text{horpo}} u$, $f\boldsymbol{t} \to t_i$ is a rule. Thus, $f\boldsymbol{t} >_{\text{whorco}} t_i$ and, by (red), $f\boldsymbol{t} >_{\text{whorco}} u$.

(2) By induction hypothesis, for all i, $f\boldsymbol{t} >_{\text{horco}}^{+} u_i$ or $t_j >_{\text{horco}}^{*} u_i$. Hence, by (*), $f\boldsymbol{t} > \boldsymbol{u}$. By (prec), $f\boldsymbol{t} > g$. Thus, by (app), $f\boldsymbol{t} > g\boldsymbol{u}$. Since $f\boldsymbol{t} \to g\boldsymbol{u}$ is a rule, $f\boldsymbol{t} >_{\text{whorco}} g\boldsymbol{u}$.

(3) By induction hypothesis, $\boldsymbol{t}\ (>_{\text{horco}}^{+})_{\text{mul}}\ \boldsymbol{u}$. Hence, by (*), $f\boldsymbol{t} > \boldsymbol{u}$. Thus, by (call), $f\boldsymbol{t} > g\boldsymbol{u}$. Since $f\boldsymbol{t} \to g\boldsymbol{u}$ is a rule, $f\boldsymbol{t} >_{\text{whorco}} g\boldsymbol{u}$.

(4) By induction hypothesis, $\boldsymbol{t}\ (>_{\text{horco}}^{+})_{\text{stat}_f}\ \boldsymbol{u}$ and, for all i, $f\boldsymbol{t} >_{\text{horco}}^{+} u_i$ or $t_j >_{\text{horco}}^{*} u_i$. Hence, by (*), $f\boldsymbol{t} > \boldsymbol{u}$. Thus, by (call), $f\boldsymbol{t} > g\boldsymbol{u}$. Since $f\boldsymbol{t} \to g\boldsymbol{u}$ is a rule, $f\boldsymbol{t} >_{\text{whorco}} g\boldsymbol{u}$.

(5) By induction hypothesis, for all i, $f\boldsymbol{t} >_{\text{horco}}^{+} u_i$ or $t_j >_{\text{horco}}^{*} u_i$. Hence, by (*), $f\boldsymbol{t} > u_i$ for all i. Thus, by (app), $f\boldsymbol{t} > \boldsymbol{u}$. Since $(f\boldsymbol{t}, \boldsymbol{u})$ is a rule, $f\boldsymbol{t} >_{\text{whorco}} \boldsymbol{u}$.

(6) For typing reasons, $(t_1, u1)\ (>_{\text{horpo}})_{\text{prod}}\ (t_2, u_2)$. Thus, by induction hypothesis, $(t_1, u_1)\ (>_{\text{horco}}^{+})_{\text{prod}}\ (t_2, u_2)$. Hence, by (cont) and transitivity, $t_1 t_2 >_{\text{horco}}^{+} u_1 u_2$.

(7) By induction hypothesis, $t >^+_{\text{horco}} u$. Thus, by (cont), $\lambda x t >^+_{\text{horco}} \lambda x u$. $\qquad\square$

We observe that, if (6) were restricted to $(t_1 >_{\text{horpo}} u_1 \wedge t_2 = u_2) \vee (t_1 = u_1 \wedge t_2 >_{\text{horpo}} u_2)$, then we would get $>_{\text{horpo}} \subseteq >_{\text{horco}}$, since this is the only case requiring transitivity.

Note that $>_{\text{horco}}$ can be extended with the accessibility ordering defined in Figure 3. The details can be found in [7].

Finally, we remark that, when restricted to first-order terms, the recursive computability ordering is equal to the usual first-order recursive path ordering [41, 14], the subterm rule being simulated by (arg) and (red).

Lemma 14. *The relation defined in Figure 4 by the rules (arg), (decomp-symb), (call) and the rule:*

$$(\text{prec-app}) \quad \frac{f >_{\mathcal{F}} g^{\boldsymbol{U} \Rightarrow U} \quad f\boldsymbol{l} > \boldsymbol{u}^{\boldsymbol{U}}}{f\boldsymbol{l} > g\boldsymbol{u}}$$

is equal to the usual first-order recursive path ordering.

9 Conclusion

We show through various extensions how powerful is the notion of computability closure introduced by Jean-Pierre Jouannaud and Mitsuhiro Okada. In particular, we show how it can easily be turned into a well-founded ordering containing Jean-Pierre Jouannaud and Albert Rubio's higher-order recursive path ordering. This provides a simple way to extend this ordering to richer type disciplines. However, its definition as the closure by context of another relation is not completely satisfactory, all the more so since one wants to combine it with the accessibility ordering. We should therefore try to find a new definition of HORPO that nicely integrates the notions of computability closure and accessibility ordering in order to capture definitions on higher-order data types (data types with constructors having functional recursive arguments).

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. H. Barendregt. Lambda calculi with types. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of logic in computer science*, volume 2. Oxford University Press, 1992.
3. F. Blanqui. Higher-order dependency pairs. In *Proceedings of the 8th International Workshop on Termination, 2006*.
4. F. Blanqui. Rewriting modulo in Deduction modulo. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 2706, 2003.
5. F. Blanqui. Termination and confluence of higher-order rewrite systems. In *Proceedings of the 11th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 1833, 2000.

6. F. Blanqui. Definitions by rewriting in the Calculus of Constructions. *Mathematical Structures in Computer Science*, 15(1):37–92, 2005.

7. F. Blanqui. (HO)RPO revisited. Research Report 5972, INRIA, 2006.

8. F. Blanqui, J.-P. Jouannaud, and M. Okada. The Calculus of Algebraic Constructions. In *Proceedings of the 10th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 1631, 1999.

9. F. Blanqui, J.-P. Jouannaud, and M. Okada. Inductive-data-type Systems. *Theoretical Computer Science*, 272:41–68, 2002.

10. V. Breazu-Tannen. Combining algebra and higher-order types. In *Proceedings of the 3rd IEEE Symposium on Logic in Computer Science*, 1988.

11. V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic strong normalization. In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 372, 1989.

12. V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic strong normalization. *Theoretical Computer Science*, 83(1):3–28, 1991.

13. V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic confluence. *Information and Computation*, 114(1):1–29, 1994.

14. N. Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.

15. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 6. North-Holland, 1990.

16. D. Dougherty. Adding algebraic rewriting to the untyped lambda calculus. In *Proceedings of the 4th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 488, 1991.

17. D. Dougherty. Adding algebraic rewriting to the untyped lambda calculus. *Information and Computation*, 101(2):251–267, 1992.

18. J.-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse et son application à l'élimination des coupures dans l'analyse et la théorie des types. In J. Fenstad, editor, *Proc. of the 2nd Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1971.

19. J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmetique d'ordre supérieur*. PhD thesis, Université Paris VII, France, 1972.

20. J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1988.

21. J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal on Computing*, 15(4):1155–1194, 1986.

22. J.-P. Jouannaud and M. Okada. Executable higher-order algebraic specification languages. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, 1991.

23. J.-P. Jouannaud and M. Okada. Abstract Data Type Systems. *Theoretical Computer Science*, 173(2):349–391, 1997.

24. J.-P. Jouannaud and A. Rubio. Higher-order orderings for normal rewriting. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 4098, 2006.

25. J.-P. Jouannaud and A. Rubio. The Higher-Order Recursive Path Ordering. In *Proceedings of the 14th IEEE Symposium on Logic in Computer Science*, 1999.

26. J.-P. Jouannaud and A. Rubio. A recursive path ordering for higher-order terms in eta-long beta-normal form. In *Proceedings of the 7th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 1103, 1996.

27. Z. Khasidashvili. Expression Reduction Systems. In *Proc. of I. Vekua Institute of Applied Mathematics*, volume 36, 1990.

28. J. W. Klop. *Combinatory Reduction Systems*. PhD thesis, Utrecht Universiteit, The Netherlands, 1980. Published as Mathematical Center Tract 129.

29. J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993.

30. J. B. Kruskal. Well-quasi-ordering, the tree theorem, and vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95:210–225, 1960.

31. C. Loria-Saenz and J. Steinbach. Termination of combined (rewrite and λ-calculus) systems. In *Proceedings of the 3rd International Workshop on Conditional and Typed Rewriting Systems*, Lecture Notes in Computer Science 656, 1992.

32. O. Lysne and J. Piris. A termination ordering for higher order rewrite systems. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 914, 1995.

33. R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(2):3–29, 1998.

34. N. P. Mendler. Recursive types and type constraints in second order lambda calculus. In *Proceedings of the 2nd IEEE Symposium on Logic in Computer Science*, 1987.

35. N. P. Mendler. Inductive types and type constraints in the second-order lambda calculus. *Annals of Pure and Applied Logic*, 51(1-2):159–172, 1991.

36. D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. In *Proceedings of the International Workshop on Extensions of Logic Programming*, Lecture Notes in Computer Science 475, 1989.

37. D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

38. D. Miller and G. Nadathur. An overview of λProlog. In *Proceedings of the 5th International Conference on Logic Programming*, MIT Press, 1988.

39. T. Nipkow. Higher-order critical pairs. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, 1991.

40. M. Okada. Strong normalizability for the combined system of the typed lambda calculus and an arbitrary convergent term rewrite system. In *Proceedings of the 1989 International Symposium on Symbolic and Algebraic Computation*, ACM Press.

41. D. A. Plaisted. A recursively defined ordering for proving termination of term rewriting systems. Technical report, University of Illinois, Urbana-Champaign, United States, 1978.

42. M. Sakai and K. Kusakari. On new dependency pair method for proving termination of higher-order rewrite systems. In *Proceedings of the 1st International Workshop on Rewriting in Proof and Computation, 2001*.

43. W. W. Tait. Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967.

44. W. W. Tait. A realizability interpretation of the theory of species. In R. Parikh, editor, *Proceedings of the 1972 Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, 1975.

45. J. van de Pol. Termination proofs for higher-order rewrite systems. In *Proceedings of the 1st International Workshop on Higher-Order Algebra, Logic and Term Rewriting*, Lecture Notes in Computer Science 816, 1993.

46. V. van Oostrom. Development closed critical pairs. In *Proceedings of the 2nd International Workshop on Higher-Order Algebra, Logic and Term Rewriting*, Lecture Notes in Computer Science 1074, 1995.

47. V. van Oostrom. *Confluence for Abstract and Higher-Order Rewriting*. PhD thesis, Vrije Universiteit Amsterdam, The Netherlands, 1994.

48. G. Peterson and M. Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM*, 28(2):233–264, 1981.

49. F. van Raamsdonk. *Confluence and Normalization for Higher-Order Rewriting*. PhD thesis, Vrije University Amsterdam, The Netherlands, 1996.

50. D. Walukiewicz-Chrząszcz. *Termination of Rewriting in the Calculus of Constructions*. PhD thesis, Warsaw University, Poland and Université d'Orsay, France, 2003.

51. D. Walukiewicz-Chrząszcz. Termination of rewriting in the Calculus of Constructions. *Journal of Functional Programming*, 13(2):339–414, 2003.