Termination of rewrite relations on λ-terms based on Girard's notion of reducibility

Frédéric Blanqui[a,1]

[a] *Institut National de Recherche en Informatique et Automatique (INRIA), France*

Abstract

In this paper, we show how to extend the notion of reducibility introduced by Girard for proving the termination of β-reduction in the polymorphic λ-calculus, to prove the termination of various kinds of rewrite relations on λ-terms, including rewriting modulo some equational theory and rewriting with matching modulo $\beta\eta$, by using the notion of *computability closure*. This provides a powerful termination criterion for various higher-order rewriting frameworks, including Klop's Combinatory Reductions Systems with simple types and Nipkow's Higher-order Rewrite Systems.

Keywords: termination, rewriting, λ-calculus, types, Girard's reducibility, rewriting modulo, matching modulo $\beta\eta$, patterns *à la* Miller

1. Introduction

This paper addresses the problem of checking the termination of various kinds of rewrite relations on simply typed λ-terms.

First-order rewriting [KB70, DJ90] and λ-calculus [Chu40, Bar84] are two general (Turing-complete) computational frameworks with different strengths and limitations.

The λ-calculus is a language for expressing arbitrary functions based on a few primitives (abstraction over some variable and application of a function to an argument). Computation is done by repeatedly substituting formal arguments by actual ones (β-reduction) [Chu40].

In first-order rewriting, one considers a fixed set of function symbols and a fixed set of term transformation rules. Computation is done by repeatedly substituting the left-hand side of a rule by the corresponding right-hand side [KB70].

Hence, in λ-calculus, there is only one computation rule and it is unconditional while, in rewriting, a computation step occurs only if a term *matches* a pattern (possibly modulo some equational theory).

But first-order rewriting cannot express in a simple way anonymous functions or patterns with bound variables. See for instance the works on Combinatory Logic [CF58], first-order definitions of a substitution operation compatible with α-equivalence [dB78, ACCL91, Kes07] (to cite just a few, for the amount of publications on this subject is very important), or first-order encodings of higher-order rewriting [BKR05].

Rewriting on λ-terms, or higher-order rewriting, aims at unifying these two languages. Several approaches exist like Klop's Combinatory Reduction Systems (CRSs) [Klo80, KvOvR93], Khasidashvili's Expression Reduction Systems (ERSs) [Kha90, GKK05], Nipkow's Higher-order Rewrite Systems (HRSs) [Nip91, MN98], or Jouannaud and Okada's higher-order algebraic specification languages (HALs) [JO91, JO97a]. Van Oostrom and van Raamsdonk studied the relations between CRSs and HRSs [vOvR93] and developed a general framework (HORSs) that subsumes most of the previous approaches [vO94, vR96].

[1] Hosted from July 2012 to August 2013 by the Institute of Software of the Chinese Academy of Sciences, Beijing, China.

In another direction, some researchers introduced calculi where patterns are first-class citizens: van Oostrom's pattern calculus [vO90, KvOdV08], Cirstea and Kirchner's ρ-calculus [CK01a, CK01b], Jay and Kesner's pattern calculus [Jay04, JK09], or some extensions of ML or Haskell [Erw96, Tul10].

In this paper, I will consider HALs with curried symbols (*i.e.* all symbols are of arity 0), that is, arbitrary simply typed λ-terms with curried symbols defined by the combination of rewrite rules and β-reduction. But, as we will see in Section 6.5, our results easily apply to HRSs and simply typed CRSs as well.

My goal is to develop techniques for proving the termination of such a system, *i.e.* the combination of β-reduction and arbitrary user-defined rewrite rules.

For proving the termination of rewrite relations on λ-terms, one can try to extend to λ-calculus techniques developed for first-order rewriting (*e.g.* [LSS92, vdP96, SWS01, JB04, FK12]) or, vice versa, adapt to rewriting techniques developed for λ-calculus (*e.g.* [JO91, Bla04, BR06]).

Since β-reduction does not terminate in general, one usually restricts his attention to some strict subset of the set of all λ-terms, like the set of λ-terms typable in some type system [Bar92] (types were first introduced by logicians as an alternative to the restriction of the comprehension axiom in set theory, and later found important applications in programming languages and compilers).

To prove the termination of β-reduction in typed λ-calculi, there are essentially three techniques:

Direct proof. In the simply-typed λ-calculus, it is possible to prove the termination of β-reduction by induction on the size of the type of the substituted variable [San67, vD80]. For instance, in the reduction sequence $(\lambda x^{A\Rightarrow B} xy)(\lambda y^A z) \to_\beta (\lambda y^A z)y \to_\beta z$, the type in the first reduction step of the substituted variable x is $A \Rightarrow B$ while, in the second reduction step (which is generated by the first one), the type of the substituted variable y is A.

But this technique extends neither to polymorphic types nor to rewriting since, in both cases, the type of the substituted variables may increase:

- With polymorphic types, consider the reduction sequence $(\lambda x^{(\forall\alpha)\alpha\Rightarrow B} xY y)(\Lambda\alpha\lambda y^\alpha z) \to_\beta (\Lambda\alpha\lambda y^\alpha z)Yy \to_\beta (\lambda y^Y z)y \to_\beta z$. In the first reduction step, the type of the substituted variable x is $(\forall\alpha)\alpha \Rightarrow B$ while, in the last reduction step, the type of the substituted variable y is the arbitrary type Y.
- With the rule $\mathsf{K}\ x\ \mathsf{a} \to_\mathcal{R} x$ where $\mathsf{K} : T \to \mathsf{A} \to T$, consider the reduction sequence $(\lambda z \mathsf{K} x z)\mathsf{a} \to_\beta \mathsf{K}x\mathsf{a} \to_\mathcal{R} x$. In the first reduction step, the type of the substituted variable z is A while, in the second reduction step which is generated by the first one, the type of the substituted variable x is the arbitrary type T.

Interpretation. For the simply-typed λ-calculus again, Gandy showed that λI-terms (λ-terms where, in every subterm $\lambda x t$, x has at least one free occurrence in t), can be interpreted by hereditarily monotone functionals on $\mathbb{N}$ [Gan80]. Then, van de Pol showed that there is a transformation from λ-terms to λI-terms that strictly decreases when there is a β-reduction, and extended this to higher-order rewriting and other domains than $\mathbb{N}$ [vdP96]. Finally, Hamana developed a categorical semantics for terms with bound variables [Ham06] based on the work of Fiore, Plotkin and Turi [FPT99], that is complete for termination (which is not the case of van de Pol's interpretations), and extended to higher-order terms the technique of semantic labeling [Ham07] introduced for first-order terms by Zantema [Zan95]. However, Roux showed that its application to β-reduction itself is not immediate since the interpretation of β-reduction is not β-reduction [BR09, Rou11].

Computability. The last technique, not limited to simply-typed λ-calculus, is based on Tait and Girard's notions of computability[2] introduced by Tait [Tai67] for the weak normalization of the simply-typed λ-calculus, and extended by Girard to polymorphic types [Gir71] and strong normalization [Gir72].

[2]In fact, Tait speaks of "convertibility" and Girard of "reducibility". To the best of my knowledge, the expression "computability" is due to Troelstra [Tro73] although Troelstra himself invokes Tait. This notion of computability has to be distinguished from the one of Turing and Church [Tur37]. However, given a Tait-computable λ-term $t : U \Rightarrow V$, the function that maps every Tait-computable λ-term $u : U$ to the normal form of tu is indeed Turing-computable.

There are however relations between these techniques. For instance, van de Pol proved that his interpretations on $\mathbb{N}$ can be obtained from a computability proof by adding information on the length of reductions [vdP96]. Conversely, the author and Roux proved that size-based termination [Gim98, Abe04, BFG$^+$04, Bla04], which is a refinement of computability, can to some extent be seen as an instance of Hamana's higher-order semantic labeling technique [BR09].

In this paper, we will consider a technique based on computability.

Computability has been first used for proving the termination of the combination of β-reduction, in the simply typed or polymorphic λ-calculus, together with a first-order rewrite system that is terminating on first-order terms, by Tannen and Gallier [BTG89, BTG91] and Okada [Oka89] independently. It was noticed later by Dougherty that, with first-order rewriting, a proof can be given that is independent of the proof of termination of β-reduction [Dou91, Dou92], because first-order rewriting cannot create β-redexes (but just duplicate them). But this does not extend to higher-order rewriting or to function symbols with polymorphic types.

In [JO91, JO97a], Jouannaud and Okada extended computability to higher-order rewrite rules following a schema extending Gödel' system T recursion schema on Peano integers [Göd58] to arbitrary first-order data types. This work was then extended to Coquand and Huet's Calculus of Constructions [CH84, CH88] in a series of papers culminating in [BFG97].

In [JO97b], Jouannaud and Okada reformulated this general schema as an inductively defined set called *computability closure*. This notion was then extended with the author to strictly positive inductive types [BJO99, BJO02] and to the Calculus of Algebraic Constructions, that is an extension of the Calculus of Constructions where types equivalent modulo user-defined rewrite rules are identified and function symbols can be given polymorphic and dependent types [Bla05].

In this paper, we provide a new presentation of the notion of computability closure for standard rewriting and show how to extend it for dealing with rewriting modulo some equational theory and higher-order pattern-matching, by providing detailed proofs of results sketched in [Bla03, Bla07]. We do it in a progressive way by showing, step by step, how the notion of computability closure can be extended to cope with new term constructions or new rewriting mechanisms. To avoid unnecessary technicalities related to the type discipline, we do it in the simply typed λ-calculus but this work could be conducted in the Calculus of Algebraic Constructions as well, following the lines of [Bla05].

The paper is organized as follows. In Section 2, we define the set of terms that will be considered, introduce our notations and recall some general results on well-founded relations. In Section 3, we present the different definitions of computability introduced so far and discuss their relations and applicability to rewriting. In Section 4, we show how Girard's definition of computability can be extended to deal with rewriting with matching modulo α-equivalence by introducing the notion of computability closure, and provide a first core definition of such a computability closure. Then follows a number of subsections and sections showing how to extend this core definition to deal with new constructions or more general notions of rewriting: abstraction and bound variables, basic subterms, recursive functions, higher-order subterms, matching on defined symbols, rewriting modulo an equational theory and rewriting with matching modulo $\beta\eta$. We finally explain why our results apply to HRSs and simply typed CRSs as well.

Parts of this work have already been formalized in the Coq proof assistant [Bla13]. See the conclusion for more details about that.

2. Definitions and notations

We first recall some definitions and notations about simply-typed λ-terms, rewriting and well-founded relations. See for instance [DJ90, Bar92, TeR03] for more details.

2.1. Notations for sequences

Given a set A, let A^* be the free monoid generated from A, *i.e.* the set of finite sequences of elements of A or *words* on A. We denote the empty word by ε, word concatenation by juxtaposition, and the length

of a word w by $|w|$. We often denote a word $a_1 \dots a_n$ by $\vec{a}$. A word p is a *prefix* of a word q, written $p \leq q$, if there is r such that $q = pr$. The prefix relation is a partial ordering. We write $p\#q$ if p and q are not comparable or *disjoint*.

2.2. Simple types

We assume given a set $\mathcal{B}$ of *type constants*. As usual, the set $\mathcal{T}$ of (simple) *types* is defined recursively as follows [Chu40]:

- a type constant $B \in \mathcal{B}$ is a type;
- if T and U are types, then $T \Rightarrow U$ is a type.

2.3. Terms

All over the paper, we only consider simply typed λ-terms (terms are always well-typed).

We follow Pottinger's approach [Pot78], that is, we assume that every variable or function symbol comes equipped with a fixed (simple) type and that α-equivalence replaces a variable by another variable of the same type only (assuming an infinite set of variable for each type). Hence, we do not have to consider untyped terms and introduce typing environments (finite map from variables to types) for terms and rules: it is like working in a fixed infinite typing environment.

Let $\mathcal{X}$ be an infinite set of *variables* and $\mathcal{F}$ be a set of *function symbols* disjoint from $\mathcal{X}$, and assume that each variable or function symbol s is equipped with a type $\tau(s)$ so that there is an infinite number of variable of each type.

The family $(L^T)_{T \in \mathcal{T}}$ of the sets of *raw terms of type* T is inductively defined as follows:

- if $s \in \mathcal{X} \cup \mathcal{F}$, then $s \in L^{\tau(s)}$;
- if $x \in \mathcal{X}$, $T \in \mathcal{T}$ and $t \in L^T$, then $\lambda xt \in L^{\tau(x) \Rightarrow T}$;
- if $U, V \in \mathcal{T}$, $t \in L^{U \Rightarrow V}$ and $u \in L^U$, then $tu \in L^V$.

For every type T, the set $\mathcal{L}^T$ of *terms of type* T is the quotient of L^T by type-preserving α-equivalence, that is, $\lambda xt =_\alpha \lambda yu$ only if $\tau(x) = \tau(y)$ [Pot78]. Let $\mathcal{L} = \bigcup_{T \in \mathcal{T}} \mathcal{L}^T$ be the set of all (typed) terms. We write $t : T$ or $\tau(t) = T$ if the α-equivalence class of t belongs to $\mathcal{L}^T$. A relation R on terms *preserves types* if $\tau(t) = \tau(u)$ whenever $(t, u) \in R$, written tRu (*e.g.* α-equivalence).

Note that function symbols do not have to be applied to any argument nor any fixed number of arguments. Hence, f, $\mathsf{f}x$, $\mathsf{f}xy$, etc. are legal terms (as long as they are well-typed). However, in some examples, for convenience, we may use infix notations, like $x + y$ for denoting $+xy$.

Let $\mathrm{FV}(t)$ be the set of variables having a free occurrence in t (*i.e.* not bound by a λ), and $\mathrm{BV}(t)$ be the set of binding variables of a raw term t (*e.g.* $\mathrm{BV}(\lambda xy) = \{x\}$).

A term is *linear* if no variable has more than one *free* occurrence in it.

A term is *algebraic* if it contains no subterm of the form λxt or xt.

A type-preserving relation R on terms is *monotone* if, for all t, u, v, x such that tRu, one has $(\lambda xt)R(\lambda xu)$, $(tv)R(uv)$ whenever tv is well-typed, and $(vt)R(vu)$ whenever vt is well-typed.

2.4. Substitution

A substitution σ is a map from $\mathcal{X}$ to $\mathcal{L}$ such that (1) for all $x \in \mathcal{X}$, $\tau(\sigma(x)) = \tau(x)$, and (2) its *domain* $\mathrm{dom}(\sigma) = \{x \in \mathcal{X} \mid \sigma(x) \neq x\}$ is finite. In particular, we write $\frac{u}{x}$ for the substitution σ such that $\sigma(x) = u$ and $\sigma(y) = y$ if $y \neq x$. Let $\mathrm{FV}(\sigma) = \bigcup\{\mathrm{FV}(\sigma(x)) \mid x \in \mathrm{dom}(\sigma)\}$. A substitution σ is *away from* $X \subseteq \mathcal{X}$ if $(\mathrm{dom}(\sigma) \cup \mathrm{FV}(\sigma)) \cap X = \emptyset$.

Given a term t and a substitution σ, we denote by $t\sigma$ the term obtained by replacing in t each free occurrence of a variable x by $\sigma(x)$ by renaming, if necessary, variables bound in t so that no variable free in $\sigma(x)$ becomes bound [CF58]. Note that substitution preserves typing: $\tau(t\sigma) = \tau(t)$.

A relation R is *stable by substitution* (away from X) if $(t\sigma)R(u\sigma)$ whenever tRu (and σ is away from X). It is a *congruence* if it is an equivalence relation that is monotone and stable by substitution.

2.5. Stable subterm ordering

The notion of sub-raw-term is not compatible with α-equivalence. Instead, we consider the notion of *stable subterm*: $t \trianglelefteq_{\mathsf{s}} u$ if t is a sub-raw-term of u and $\mathrm{FV}(t) \subseteq \mathrm{FV}(u)$. The relation $\trianglelefteq_{\mathsf{s}}$ is a partial ordering stable by substitution. Let $\triangleleft_{\mathsf{s}}$ be its strict part and $\trianglerighteq_{\mathsf{s}}$ (resp. $\triangleright_{\mathsf{s}}$) be the inverse of $\trianglelefteq_{\mathsf{s}}$ (resp. $\triangleleft_{\mathsf{s}}$).

2.6. Positions

The set of *positions* in a (raw) term t, $\mathrm{Pos}(t)$, is the subset of $\{0,1\}^*$ such that:

- $\mathrm{Pos}(x) = \mathrm{Pos}(\mathsf{f}) = \{\varepsilon\}$ if $x \in \mathcal{X}$ and $\mathsf{f} \in \mathcal{F}$
- $\mathrm{Pos}(tu) = \{\varepsilon\} \cup \{0w \mid w \in \mathrm{Pos}(t)\} \cup \{1w \mid w \in \mathrm{Pos}(u)\}$
- $\mathrm{Pos}(\lambda xt) = \{\varepsilon\} \cup \{0w \mid w \in \mathrm{Pos}(t)\}$

Given a (raw) term t, we denote by $t|_p$ its sub-raw-term at position $p \in \mathrm{Pos}(t)$, and by $t[u]_p$ the (raw) term obtained by replacing it by u.

A term t is *η-long* if every variable or function symbol occurring in it is maximally applied, that is, for all $p \in \mathrm{Pos}(t)$, if $t|_p \in \mathcal{X} \cup \mathcal{F}$ and $t|_p : \vec{T} \Rightarrow \mathsf{A}$, then there are $q \in \mathrm{Pos}(t)$ and $\vec{t}$ such that $p = q0^{|\vec{T}|}$ and $t|_q = t|_p\vec{t}$ [Hue76].

Given a term t and $p \in \mathrm{Pos}(t)$, the set $\mathrm{BV}(t,p)$ of binding variables above $t|_p$ is defined as follows:

- $\mathrm{BV}(t,\varepsilon) = \emptyset$
- $\mathrm{BV}(tu, 0p) = \mathrm{BV}(t,p)$
- $\mathrm{BV}(tu, 1p) = \mathrm{BV}(u,p)$
- $\mathrm{BV}(\lambda xt, 0p) = \{x\} \cup \mathrm{BV}(t,p)$

For instance, $\mathrm{Pos}(\lambda x\mathsf{f}x) = \{\varepsilon, 0, 00, 01\}$ and $\mathrm{BV}(\lambda x\mathsf{f}x, 0) = \{x\}$.

2.7. Rewriting

The relation of *β-reduction* (resp. *η-reduction*), $\to_\beta$ (resp. $\to_\eta$), is the monotone closure of $\{((\lambda xt)u, t^u_x) \mid t,u \in \mathcal{L}, x \in \mathcal{X}\}$ (resp. $\{(\lambda x(tx), t) \mid t \in \mathcal{L}, x \in \mathcal{X}, x \notin \mathrm{FV}(t)\}$). We write $t \xrightarrow{p}_\beta u$ to indicate that $t|_p = (\lambda xa)b$ and $u = t[a^b_x]_p$, and similarly for $t \xrightarrow{p}_\eta u$. Note that the relation $\to_{\beta\eta} = \to_\beta \cup \to_\eta$ preserves typing: if $t : T$ and $t \to_{\beta\eta} t'$, then $t' : T$.

An *equation* is a pair of terms (l,r), written $l = r$, such that $\tau(l) = \tau(r)$. A *(rewrite) rule* is a pair of terms (l,r), written $l \to r$, such that $\tau(l) = \tau(r)$, l is of the form $\mathsf{f}\vec{l}$ and $\mathrm{FV}(r) \subseteq \mathrm{FV}(l)$.

We assume neither that, if $\mathsf{f}\vec{l} \to r$ is a rule, then every occurrence of f in r comes applied to $|\vec{l}|$ arguments, nor that, if $\mathsf{f}\vec{l} \to r$ and $\mathsf{f}\vec{m} \to s$ are two distinct rules, then $|\vec{l}| = |\vec{m}|$. And, indeed, we will give examples of systems that do not satisfy these constraints in Section 4.6 (function ex) and Section 6 (after Lemma 20). Such systems are necessary for dealing with matching modulo $\beta\eta$ because we use curried symbols. In contrast, in HRSs [Nip91], function symbols are always maximally applied (wrt their types) since terms are in η-long form and rules are of the form $\mathsf{f}\vec{l} \to r$ with $\mathsf{f}\vec{l}$ of base type. Note however that, in [vdP96], van de Pol considers rules not necessarily in η-long form nor of base type.

The *rewriting relation* generated by a set of rules $\mathcal{R}$, written $\to_{\mathcal{R}}$, is the closure by monotony and substitution of $\mathcal{R}$. Hence, $t \to_{\mathcal{R}} u$ if there are $p \in \mathrm{Pos}(t)$, $l \to r \in \mathcal{R}$ and σ such that $t|_p = l\sigma$ and $u = t[r\sigma]_p$. For instance, with $\mathcal{R} = \{\mathsf{f}x \to x\}$, we have $\lambda x\mathsf{f}xy \to_{\mathcal{R}} \lambda xxy$. Note that rewriting preserves typing: if $t : T$ and $t \to_{\mathcal{R}} t'$, then $t' : T$.

Given a set of rules $\mathcal{R}$, let $\mathcal{D}(\mathcal{R}) = \{\mathsf{f} \in \mathcal{F} \mid \exists\vec{l}, \exists r, \mathsf{f}\vec{l} \to r \in \mathcal{R}\}$ be the subset of symbols *defined* by $\mathcal{R}$, and $\alpha_{\mathsf{f}} = \mathsf{sup}\{|\vec{l}| \mid \exists r, \mathsf{f}\vec{l} \to r \in \mathcal{R}\}$. Note that α_{f} is finite even if $\mathcal{R}$ is infinite for $\mathsf{f}\vec{l}$ is *simply* typed by assumption.[3]

[3]However, with polymorphic types, or dependent types together with type-level rewriting (*e.g.* strong elimination), α_{f} may be infinite if $\mathcal{R}$ is infinite.

2.8. Notations for relations

Given a relation R on a set A, let $R(t) = \{u \in A \mid tRu\}$ be the set of reducts or successors of t. An element t such that $R(t) = \emptyset$ is said to be in *normal form* or irreducible.

Given a relation R, let $R^=$ be the reflexive closure of R, R^+ its transitive closure, R^* its reflexive and transitive closure, and R^{-1} its inverse ($xR^{-1}y$ iff yRx).

However, we will denote by $\leftarrow_\beta$, $\leftarrow_\eta$ and $\leftarrow_\mathcal{R}$ the inverse relations of $\rightarrow_\beta$, $\rightarrow_\eta$ and $\rightarrow_\mathcal{R}$ respectively; by $\leftrightarrow_\beta$, $\leftrightarrow_\eta$ and $\leftrightarrow_{\beta\eta}$ the *symmetric closures* of $\rightarrow_\beta$, $\rightarrow_\eta$ and $\rightarrow_{\beta\eta}$ respectively (*i.e.* $\rightarrow_\beta \cup \leftarrow_\beta$, etc.); and by $=_\eta$ and $=_{\beta\eta}$ the reflexive and transitive closures of $\leftrightarrow_\eta$ and $\leftrightarrow_{\beta\eta}$ respectively.

Given two relations R and S, we denote their composition by juxtaposition and say that R *commutes* with S if $RS \subseteq SR$. For instance, if R is monotone, then $\rhd_\mathsf{s}$ commutes with R.

A relation R is *strongly confluent* if $R^{-1}R \subseteq (R^=)(R^=)^{-1}$, *locally confluent* if $R^{-1}R \subseteq R^*(R^{-1})^*$, and *confluent* if $(R^{-1})^*R^* \subseteq R^*(R^{-1})^*$. For instance, the relations $\rightarrow_\eta$, $\rightarrow_\beta$ and their union $\rightarrow_{\beta\eta}$ are all confluent [Pot78].

2.9. Notations for quasi-orderings

Given an equivalence relation R on a set A, we denote by $[t]_R$ the equivalence class of an element t, and by A/R the set of equivalence classes modulo R.

Given a quasi-ordering $\geq$ on a set A (transitive and reflexive relation), let $\simeq = \geq \cap \geq^{-1}$ be its *associated equivalence relation* and $> = \geq - \geq^{-1}$ be its *strict part* (transitive and irreflexive relation).

2.10. Well-founded relations

Given a set A, an element $a \in A$ is *strongly normalizing* wrt a relation R on A if there is no infinite sequence $a = a_0Ra_1R\ldots$ The relation R *terminates* (or is *noetherian* or *well-founded*[4]) on A if every element of A is strongly normalizing wrt R. Let $\mathrm{SN}(R)$ be the set of elements of A that are strongly normalizing wrt R. By abuse of language, we sometimes say that a quasi-ordering $\geq$ is well-founded when its strict part so is.

If R terminates (resp. is confluent) then every element has at least (resp. at most) one normal form. In particular, we will denote by $t{\downarrow_\eta}$ the unique normal form of t wrt $\rightarrow_\eta$.

Note that, if R is monotone, then $R \cup \rhd_\mathsf{s}$ terminates iff R terminates.

In this paper, we are interested in the termination of the relation $\rightarrow_\beta \cup \rightarrow_\mathcal{R}$, or variants thereof. Note that $\rightarrow_\beta$ terminates on well-typed terms [San67]. However, since termination is not a modular property (already in the first-order case) [Toy87], the termination of $\rightarrow_\mathcal{R}$ is generally not sufficient to guarantee the termination of $\rightarrow_\beta \cup \rightarrow_\mathcal{R}$. Moreover, considering $\rightarrow_\mathcal{R}$ alone does not make sense when, in a right-hand side of a rule, a free variable is applied to a term. This is not the case in CRSs and HRSs since, in these systems, the definition of rewriting includes some β-reductions after a rule application [vOvR93].

2.11. Product quasi-ordering

The *product* of n relations $R_1, \ldots, R_n$ on the sets $A_1, \ldots, A_n$ respectively is the relation $(R_1, \ldots, R_n)_{\mathrm{prod}}$ on $A_1 \times \ldots \times A_n$ such that $\vec{x}\,(R_1, \ldots, R_n)_{\mathrm{prod}}\,\vec{y}$ if, for all $i \in [1, n]$, $x_i R_i y_i$.

If each R_i is a quasi-ordering, then $(R_1, \ldots, R_n)_{\mathrm{prod}}$ is a quasi-ordering too. If, moreover, the strict parts of $R_1, \ldots, R_n$ are well-founded, then the strict part of $(R_1, \ldots, R_n)_{\mathrm{prod}}$ is well-founded too.

Given a quasi-ordering $\geq$ on a set A, let also $\geq_{\mathrm{prod}}$ denote the product quasi-ordering on A^n with each component ordered by $\geq$.

2.12. Multiset quasi-ordering

Given a set A, let $\mathcal{M} = \mathbb{M}(A)$ be the set of *finite multisets* on A (functions from A to $\mathbb{N}$ with finite support) [DM79]. Given a quasi-ordering $\geq_A$ on A, the *extension* of $\geq_A$ on finite multisets is the smallest quasi-ordering $\geq_\mathcal{M}$ containing $>^1_\mathcal{M} \cup \simeq_\mathcal{M}$ where $\simeq_\mathcal{M}$ and $>^1_\mathcal{M}$ are defined as follows [CJ03]:

- $\emptyset \simeq_\mathcal{M} \emptyset$, and $M + \{\!|x|\!\} \simeq_\mathcal{M} N + \{\!|y|\!\}$ if $M \simeq_\mathcal{M} N$ and $x \simeq_A y$;[5]

[4] In contrast with the mathematical tradition where a relation R is said *well-founded* if there is no infinite descending chain $a_0R^{-1}a_1R^{-1}\ldots$

[5] Here, $A + B$ is the multiset union of the multisets A and B, and $\{\!|y_1, \ldots, y_n|\!\}$ the multiset made of $y_1, \ldots, y_n$.

- $M + \{|x|\} >^1_{\mathcal{M}} M + \{|y_1, \ldots, y_n|\}$ $(n \geq 0)$ if, for every $i \in [0, n]$, $x >_A y_i$;

where $\simeq_A$ (resp. $>_A$) is the equivalence relation associated to (resp. strict part of) $\geq_A$.

Its associated equivalence relation is $\simeq_{\mathcal{M}}$. Its strict part $>_{\mathcal{M}}$ is $(>^1_{\mathcal{M}})^+ \simeq_{\mathcal{M}}$. It is well-founded if $>_A$ is well-founded.

Finally, let $\geq_{\text{mul}}$ be the quasi-ordering on A^* such that $\vec{x} \geq_{\text{mul}} \vec{y}$ if $\{|\vec{x}|\} \geq_{\mathcal{M}} \{|\vec{y}|\}$.

2.13. Lexicographic quasi-ordering

Given quasi-orderings $\geq_1, \ldots, \geq_n$ on sets $A_1, \ldots, A_n$, the *lexicographic quasi-ordering* on $A_1 \times \ldots \times A_n$, written $(\geq_1, \ldots, \geq_n)_{\text{lex}}$, is the union of the following two relations:

- $(\simeq_1, \ldots, \simeq_n)_{\text{prod}}$;
- $\vec{x} > \vec{y}$ if there is $i \in [1, n]$ such that $x_i >_i y_i$ and, for all $j < i$, $x_j \simeq_j y_j$;

where $\simeq_i$ (resp. $>_i$) is the equivalence relation associated to (resp. strict part of) $\geq_i$. If $>_1, \ldots, >_n$ are well-founded, then $>$ is well-founded too.

Given a quasi-ordering $\geq$ on a set A, let $\geq_{\text{lex}}$ also denote the lexicographic quasi-ordering on A^n with each component ordered by $\geq$.

2.14. Dependent lexicographic quasi-ordering

Given two sets A and B and, for each $x \in A$, a set $B_x \subseteq B$, the *dependent product* of A and $(B_x)_{x \in A}$ is the set $\Sigma_{x \in A} B_x$ of pairs $(x, y) \in A \times B$ such that $y \in B_x$. In the following, we use in many places a generalization to dependent products of the lexicographic quasi-ordering (generalizing to quasi-orderings Paulson's lexicographic ordering on dependent pairs [Pau86]):

Definition 1 (Dependent lexicographic quasi-ordering). The *dependent lexicographic quasi-ordering* (DLQO) on a dependent product $\Sigma_{x \in A} B_x$ associated to:

- a quasi-ordering $\geq_A$ on A;
- for each equivalence class E modulo $\simeq_A$, a set C_E equipped with a quasi-ordering $\geq_E$;
- for each $x \in A$, a *partial* function $\psi_x : B_x \to C_{[x]_{\simeq_A}}$;

is the union of the following two relations:

- $(x, y) \simeq (x', y')$ if $x \simeq_A x' \wedge \psi_x(y) \simeq_{[x]_{\simeq_A}} \psi_{x'}(y')$;
- $(x, y) > (x', y')$ if $x >_A x' \vee (x \simeq_A x' \wedge \psi_x(y) >_{[x]_{\simeq_A}} \psi_{x'}(y'))$;

where $\simeq_A$ (resp. $\simeq_E$) is the equivalence relation associated to $\geq_A$ (resp. $\geq_E$), and $>_A$ (resp. $>_E$) the strict part of $\geq_A$ (resp. $\geq_E$).

If $>_A$ and each $>_E$ are well-founded, then $>$ is well-founded too. Various examples of DLQOs will be given and used in the paper (in particular, in Sections 4.5.1 and 5.1).

3. Computability

The computability method was introduced by Tait to prove the weak normalization of (*i.e.* the existence of a normal form wrt) β-reduction in some extensions of the simply typed λ-calculus [Tai67], and was later extended by Girard for dealing with polymorphic types [Gir71] and strong normalization [Gir72, GLT88]. This method consists of:

1. defining a domain $\mathbf{Cand} \subseteq \mathcal{P}(\mathrm{SN}(\to_\beta))$ of *computability candidates* for interpreting types;[6]
2. interpreting each type T by a candidate $[\![T]\!] \in \mathbf{Cand}$;
3. proving that each term of type T is *computable*, *i.e.* belongs to $[\![T]\!]$, from which it follows that every typed term is strongly normalizing wrt $\to_\beta$.

In this section, we will see the various definitions that have been proposed for **Cand** so far, and discuss which ones are best suited for extension to arbitrary, and in particular non-orthogonal[7], rewrite systems. However, all those definitions satisfy the following properties:

- variables are computable: for every $P \in \mathbf{Cand}$, $\mathcal{X} \subseteq P$;
- **Cand** is stable by the operation $\propto\colon \mathcal{P}(\mathcal{L}) \times \mathcal{P}(\mathcal{L}) \to \mathcal{P}(\mathcal{L})$ defined by:

$$\propto(P, Q) = \{v \in \mathcal{L} \mid \forall t \in P, vt \in Q\}$$

  i.e. if $P, Q \in \mathbf{Cand}$, then $\propto(P, Q) \in \mathbf{Cand}$;
- **Cand** is stable by arbitrary[8] non-empty intersection:
  if $(A_i)_{i\in I}$ is a non-empty family of candidates, then $\bigcap_{i\in I} A_i \in \mathbf{Cand}$;
- **Cand** contains $\mathrm{SN}(\to_\beta)$.

The last two conditions imply that **Cand** has a structure of complete lattice for inclusion,[9] the greatest lower bound of a set $P \subseteq \mathbf{Cand}$ being given by the intersection $\bigcap P$ if $P \neq \emptyset$, and $\mathrm{SN}(\to_\beta)$ if $P = \emptyset$. However, its lowest upper bound (the smallest candidate containing the union) is *not* necessarily the union [Rib07a].

The intersection allows one to interpret quantification on types (polymorphism) or inductive types (see Section 4.6), while $\propto$ allows one to interpret $\Rightarrow$ so that, by definition, $vt \in [\![V]\!]$ if $v \in [\![T \Rightarrow U]\!]$ and $t \in [\![T]\!]$, which is the main problem when trying to prove the termination of β-reduction.

We now see every definition we are aware of:

- **Red**: Girard' set of *reducibility candidates* [Gir72, GLT88]. A set P belongs to **Red** if the following conditions are satisfied:

(R1) $P \subseteq \mathrm{SN}(\to_\beta)$;
(R2) P is stable by reduction: if $t \in P$ and $t \to_\beta u$, then $u \in P$;
(R3) if t is a *neutral*[10] term and $\to_\beta(t) \subseteq P$, then $t \in P$.

[6] In the following, like Girard in [Gir72, GLT88], we will in fact consider a domain $\mathbf{Cand}^T \subseteq \mathcal{P}(\mathcal{L}^T)$ for each type T, but this is not relevant in this section.

[7] A rewrite system is orthogonal if it is left-linear and non-ambiguous (*i.e.* has no critical pair). This is in particular the case of ML-like programs. An important property of orthogonal systems is their confluence [Hue80, Klo80, vO94].

[8] Finite or infinite.

[9] An inf-complete lattice L that has a biggest element is complete. The supremum of a set $P \subseteq L$ is indeed $\mathrm{glb}(\mathrm{ub}(P))$ where glb is the greatest lower bound and $\mathrm{ub}(P)$ is the *non-empty* set of all the upper bounds of P.

[10] Called "simple" in [Gir72] and "neutral" in [GLT88].

In λ-calculus with no function symbols, a term is neutral if it is not an abstraction. Neutral terms satisfy the following key property: if t is neutral then, for all terms u, $\rightarrow_\beta (tu) = \{t'u \mid t \rightarrow_\beta t'\} \cup \{tu' \mid u \rightarrow_\beta u'\}$, that is, the application of t cannot create new redexes.

- **Sat**: Tait' set of *saturated*[11] *sets* [Tai75]. A set P belongs to **Sat** if the following conditions are satisfied:

(S1) $P \subseteq \mathrm{SN}(\rightarrow_\beta)$;
(S2) P contains all the strongly normalizable terms of the form $x\vec{t}$;
(S3) if $t^u_x\vec{v} \in P$ and $u \in \mathrm{SN}(\rightarrow_\beta)$, then $(\lambda xt)u\vec{v} \in P$.

- **SatInd**: Parigot' smallest subset of **Sat** containing $\mathrm{SN}(\rightarrow_\beta)$ and stable by $\propto$ and $\bigcap$ [Par97]. As Parigot remarked, for β-reduction, it is not necessary to consider all saturated sets but only those that can be obtained from $\mathrm{SN}(\rightarrow_\beta)$ by $\propto$ and intersection.

- **Bi**: Parigot' set of *bi-orthogonals*[12] [Par93, Par97] is the set $\{\propto^* (E, \mathrm{SN}(\rightarrow_\beta)) \mid \emptyset \neq E \subseteq \mathrm{SN}(\rightarrow_\beta)^*\}$ where $\mathrm{SN}(\rightarrow_\beta)^*$ is the set of finite sequences of elements of $\mathrm{SN}(\rightarrow_\beta)$ and $\propto^*: \mathcal{P}(\mathcal{L}^*) \times \mathcal{P}(\mathcal{L}) \rightarrow \mathcal{P}(\mathcal{L})$ extends $\propto$ as follows:

$$\propto^* (E, Q) = \{v \in \mathcal{L} \mid \forall \vec{t} \in E, v\vec{t} \in Q\}$$

Note that a sequence $\vec{t} \in \mathcal{L}^*$ can be seen as the context $[]\vec{t}$. Hence,

$$\propto^* (E, Q) = \{v \in \mathcal{L} \mid \forall e \in E, e[v] \in Q\}.$$

Reducibility candidates and saturated sets are studied in [Gal90]. In particular, every reducibility candidate is a saturated set: $\mathbf{Red} \subseteq \mathbf{Sat}$. The converse does not hold in general since a saturated set does not need to be stable by reduction: for instance, the smallest saturated set containing $\lambda x(\lambda yy)x$ does not contain λxx. However, Riba showed that every saturated set stable by reduction is a reducibility candidate [Rib07a]. Hence, $\mathbf{Red} = \mathbf{Sat}_\rightarrow = \{P \in \mathbf{Sat} \mid \rightarrow_\beta (P) \subseteq P\}$. In [Par97], Parigot showed that every element of **SatInd** is a bi-orthogonal: $\mathbf{SatInd} \subseteq \mathbf{Bi}$. Finally, Riba showed that every bi-orthogonal is a reducibility candidate [Rib07b]: $\mathbf{Bi} \subseteq \mathbf{Red}$. In particular, bi-orthogonals are stable by reduction. On the other hand, I don't know whether **SatInd**, **Bi** and **Red** are distinct. In conclusion, we currently have the following relations:

$$\mathbf{SatInd} \subseteq \mathbf{Bi} \subseteq \mathbf{Red} = \mathbf{Sat}_\rightarrow \subsetneq \mathbf{Sat}$$

A natural question is then to know to which extent each one of these sets can be used to handle rewriting, and if a set allows to show the termination of more systems than the others. All these definitions rely on the form of *redexes* (reducible expressions): **Red** uses the notion of neutral term, a set $P \in \mathbf{Sat}$ has to be stable by head-expansion (inverse relation of head-reduction), and **Bi** is defined as the set of bi-orthogonals wrt a relation between terms and contexts that allows one to build redexes.

- **Bi** being exclusively based on the notion of context, it does not seem possible to extend it to non-orthogonal rewrite relations.

- The saturated sets could perhaps be extended by adding:

(S4) if $l \rightarrow r \in \mathcal{R}$, $r\sigma\vec{t} \in P$ and $\sigma \in \mathrm{SN}(\rightarrow_\beta \cup \rightarrow_\mathcal{R})$, then $l\sigma\vec{t} \in P$.

In order to have $\mathrm{SN}(\rightarrow_\beta \cup \rightarrow_\mathcal{R}) \in \mathbf{Sat}_\mathcal{R}$, one has then to prove that $l\sigma\vec{t} \in \mathrm{SN}(\rightarrow_\beta \cup \rightarrow_\mathcal{R})$ if $r\sigma\vec{t} \in \mathrm{SN}(\rightarrow_\beta \cup \rightarrow_\mathcal{R})$ and $\sigma \in \mathrm{SN}(\rightarrow_\beta \cup \rightarrow_\mathcal{R})$, which is generally not the case if $\mathcal{R}$ is not orthogonal. This problem could perhaps be solved by considering *all* the head-reducts of $l\sigma$, but then we would arrive at a condition similar to (R3).

[11] This expression seems due to Gallier [Gal90].

[12] Parigot did not use the expression "bi-orthogonal". To my knowledge, this expression first appears in [VM04]. See [Abe06], p. 67, for a discussion about the origin of this expression. Anyway, Parigot computability predicates are indeed bi-orthogonals wrt the orthogonality relation $\perp$ between $\mathcal{P}(\mathrm{SN}(\rightarrow_\beta))$ and $\mathcal{P}(\mathrm{SN}(\rightarrow_\beta)^*)$ such that $P \perp E$ if $\forall v \in P, \forall \vec{t} \in E, v\vec{t} \in \mathrm{SN}(\rightarrow_\beta)$. The (right) orthogonal of $P \subseteq \mathrm{SN}(\rightarrow_\beta)$ is $P^\perp = \{\vec{t} \in \mathrm{SN}(\rightarrow_\beta)^* \mid \forall v \in P, v\vec{t} \in \mathrm{SN}(\rightarrow_\beta)\}$, while the (left) orthogonal of $E \subseteq \mathrm{SN}(\rightarrow_\beta)^*$ is ${}^\perp E = \propto^* (E, \mathrm{SN}(\rightarrow_\beta))$. One can then see that $\mathbf{Bi} = \{P \subseteq \mathrm{SN}(\rightarrow_\beta) \mid P \neq \emptyset \wedge {}^\perp(P^\perp) = P\}$.

- In contrast to the previous domains, Girard's reducibility candidates seem easy to extend to arbitrary rewrite relations. This is therefore the notion of computability that we will use in the following.

4. Rewriting with matching modulo α-equivalence

In this section, we provide a survey on the notion of computability closure for standard rewriting (that is in fact rewriting modulo α-equivalence, because terms are defined modulo α-equivalence) first introduced in [BJO99, BJO02]. We present the computability closure progressively by showing at each step how it has to be extended to handle new term constructions. Omitted proofs can be found in [BJO02, Bla05]. For dealing with recursive function definitions (Section 4.5 below), we introduce a new more general rule based on the notion of $\mathcal{F}$-quasi-ordering compatible with application (Definition 6) and provide various examples of such $\mathcal{F}$-quasi-orderings in Section 4.5.1 (and later in Section 5.1).

4.1. Definition of computability

To extend to rewriting Girard's definition of computability predicates [GLT88], we first have to define the set of neutral terms. By analogy with abstractions, a term of the form $\mathsf{f}\vec{t}$ with $\mathsf{f} \in \mathcal{D}(\mathcal{R})$ should be neutral only if f is applied to enough arguments wrt $\mathcal{R}$, *i.e.* $|\vec{t}| \geq \alpha_{\mathsf{f}}$. Otherwise, $\mathsf{f}\vec{t}u$ could be head-reducible and the key property of neutral terms, that $\rightarrow(tu) = \{t'u \mid t \rightarrow t'\} \cup \{tu' \mid u \rightarrow u'\}$ whenever t is neutral, would not hold. Now, what about terms of the form $\mathsf{f}\vec{t}$ with $\mathsf{f} \in \mathcal{F} - \mathcal{D}(\mathcal{R})$ (undefined symbols)? We could *a priori* consider them as neutral. However, for dealing with higher-order subterms in Sections 4.6 and 4.7, we will consider type interpretations for which it seems difficult to prove (R3) if such terms are neutral. We therefore exclude them from neutral terms:

Definition 2 (Computability candidates). Given a set $\mathcal{R}$ of rewrite rules of the form $\mathsf{f}\vec{l} \rightarrow r$, a term is *neutral*[13] if it is of the form $x\vec{v}$, $(\lambda x t)u\vec{v}$ or $\mathsf{f}\vec{v}$ with $\mathsf{f} \in \mathcal{D}(\mathcal{R})$ and $|\vec{v}| \geq \alpha_{\mathsf{f}} = sup\{|\vec{l}| \mid \exists r, \mathsf{f}\vec{l} \rightarrow r \in \mathcal{R}\}$.

Given a type T, let $\mathbf{Red}^T_{\mathcal{R}}$ be the set of all the sets $P \subseteq \mathcal{L}^T$ such that:

(R1) $P \subseteq \mathrm{SN}(\rightarrow)$ where $\rightarrow = \rightarrow_\beta \cup \rightarrow_{\mathcal{R}}$;

(R2) P is stable by reduction: if $t \in P$ and $t \rightarrow u$, then $u \in P$;

(R3) if $t : T$ is neutral and $\rightarrow(t) \subseteq P$, then $t \in P$.

Given $P \in \mathbf{Red}^T_{\mathcal{R}}$ and $Q \in \mathbf{Red}^U_{\mathcal{R}}$, let $\propto(P, Q) = \{v : T \Rightarrow U \mid \forall t \in P, vt \in Q\}$.

Note that computability predicates are sets of well-typed terms and that all the elements of a computability predicate have the same type.

For the sake of simplicity, in all the remaining of the paper, we write SN instead of $\mathrm{SN}(\rightarrow)$, but $\rightarrow$ will have different meanings in sections 5 and 6.

We now check that the family $(\mathbf{Red}^T_{\mathcal{R}})_{T \in \mathcal{T}}$ has the properties described in Section 3:

Lemma 1 *For every type T, $\mathbf{Red}^T_{\mathcal{R}}$ is stable by non-empty intersection and admits $\mathrm{SN}^T = \{t : T \mid t \in \mathrm{SN}\}$ as greatest element. Moreover, for all $T, U \in \mathcal{T}$, $P \in \mathbf{Red}^T_{\mathcal{R}}$ and $Q \in \mathbf{Red}^U_{\mathcal{R}}$, $\propto(P, Q) \in \mathbf{Red}^{T \Rightarrow U}_{\mathcal{R}}$.*

Proof. The fact that $\mathrm{SN}^T \in \mathbf{Red}_{\mathcal{R}}$ and the stability by non-empty intersection are easily proved. We only detail the stability by $\propto$. Let $T, U \in \mathcal{T}$, $P \in \mathbf{Red}^T_{\mathcal{R}}$ and $Q \in \mathbf{Red}^U_{\mathcal{R}}$. Every element of $\propto(P, Q)$ is of type $T \Rightarrow U$.

(R1) Let $v \in \propto(P, Q)$. Let x be a variable of type T. By (R3), $x \in P$. By definition of $\propto$, $vx \in Q$. By (R1), $vx \in \mathrm{SN}$. Thus, $v \in \mathrm{SN}$.

(R2) Let $v \in \propto(P, Q)$, $v' \in \rightarrow(v)$ and $t \in P$. By definition of $\propto$, $vt \in Q$. By (R2), $v't \in Q$.

[13] We will give a more general definition in Definition 16.

(R3) Let $v : T \Rightarrow U$ neutral such that $\rightarrow(v) \subseteq \propto(P, Q)$, and $t \in P$. We show that $vt \in Q$ by well-founded induction on t with $\rightarrow$ as well-founded relation ($t \in \mathrm{SN}$ by (R1)). Since v is neutral, vt is neutral too. Hence, by (R3), it is sufficient to show that $\rightarrow(vt) \subseteq Q$. Let $w \in \rightarrow(vt)$. We first prove (a): either $w = v't$ with $v \rightarrow v'$, or $w = vt'$ with $t \rightarrow t'$. We proceed by case on $vt \rightarrow w$:

- $vt \rightarrow_\beta w$. Since v is neutral, v is not an abstraction and (a) is satisfied.
- $vt \rightarrow_{\mathcal{R}} w$. If there are $\mathsf{f}\vec{ll} \rightarrow r \in \mathcal{R}$ and σ such that $vt = \mathsf{f}\vec{l\sigma}l\sigma$ and $w = r\sigma$, then $v = \mathsf{f}\vec{l\sigma}$ and $|\vec{l}| < |\vec{ll}| \leq \alpha_{\mathsf{f}}$. Since v is neutral, this is not possible. Thus (a) is verified.

We now show that $w \in Q$.

- Case $w = v't$ with $v \rightarrow v'$. By assumption, $v' \in \propto(P, Q)$. As $t \in P$, we have $w \in Q$.
- Case $w = vt'$ with $t \rightarrow t'$. By (R2), $t' \in P$. Thus, by the induction hypothesis, $w \in Q$. ■

Therefore, as already mentioned in Section 3, every $\mathbf{Red}^T_{\mathcal{R}}$ is a complete lattice for inclusion.

Now, one can easily check Tait's property (S3) described in the previous section (implying that elements of $\mathbf{Red}^T_{\mathcal{R}}$ are Tait saturated sets with $\mathrm{SN}(\rightarrow_\beta)$ replaced by $\mathrm{SN}(\rightarrow_\beta \cup \rightarrow_{\mathcal{R}})$):

Lemma 2 *Given $T \in \mathcal{T}$ and $P \in \mathbf{Red}^T_{\mathcal{R}}$, $(\lambda xt)u\vec{v} \in P$ iff $(\lambda xt)u\vec{v} : T$, $t^u_x\vec{v} \in P$ and $u \in \mathrm{SN}$.*

Proof. Assume that $(\lambda xt)u\vec{v} \in P$. Then, $(\lambda xt)u\vec{v} : T$. By (R2), $t^u_x\vec{v} \in P$. By (R1), $(\lambda xt)u\vec{v} \in \mathrm{SN}$. Therefore, $u \in \mathrm{SN}$.

Assume now that $(\lambda xt)u\vec{v} : T$, $t^u_x\vec{v} \in P$ and $u \in \mathrm{SN}$. By (R1), $t^u_x\vec{v} \in \mathrm{SN}$. Therefore, $\vec{v} \in \mathrm{SN}$, $t^u_x \in \mathrm{SN}$ and $t \in \mathrm{SN}$. We now prove that, for all $t, u, \vec{v} \in \mathrm{SN}$, $(\lambda xt)u\vec{v} \in P$, by induction on $\rightarrow_{\mathrm{prod}}$. Since $(\lambda xt)u\vec{v} : T$ and $(\lambda xt)u\vec{v}$ is neutral, by (R3), it suffices to prove that every reduct w of $(\lambda xt)u\vec{v}$ belongs to P. Since rules are of the form $\mathsf{f}\vec{l} \rightarrow r$, there are two possible cases:

- $w = t^u_x\vec{v}$. Then, $w \in P$ by assumption.
- $w = (\lambda xt')u'\vec{v'}$ and $tu\vec{v} \rightarrow_{\mathrm{prod}} t'u'\vec{v'}$. Then, $w \in P$ by the induction hypothesis. ■

Corollary 1 *Given $T, U \in \mathcal{T}$, $P \in \mathbf{Red}^T_{\mathcal{R}}$ and $Q \in \mathbf{Red}^U_{\mathcal{R}}$, $\lambda xt \in \propto(Q, P)$ iff $\lambda xt : U \Rightarrow T$ and, for all $u \in Q$, $t^u_x \in P$.*

Proof. Assume that $\lambda xt \in \propto(Q, P)$ and $u \in Q$. Then, by definition of $\propto$, $\lambda xt : U \Rightarrow T$ and $(\lambda xt)u \in P$. Therefore, by (R2), $t^u_x \in P$. Assume now that $\lambda xt : U \Rightarrow T$ and, for all $u \in P$, $t^u_x \in P$. By definition, $\lambda xt \in \propto(Q, P)$ if, for all $u \in Q$, $(\lambda xt)u \in P$. So, let $u \in Q$. By (R1), $u \in \mathrm{SN}$. Therefore, by Lemma 2, $(\lambda xt)u \in P$. ■

Given two sets A and B and, for each $x \in A$, a set $B_x \subseteq B$, let $\Pi_{x\in A}B_x = \mathcal{P}(\Sigma_{x\in A}B_x)$ be the set of *partial* functions $f : A \rightarrow B$ such that, for all $x \in \mathrm{dom}(f)$, $f(x) \in B_x$.

Given an interpretation of type constants $I \in \Pi_{\mathsf{B}\in\mathcal{B}}\mathbf{Red}^{\mathsf{B}}_{\mathcal{R}}$, the interpretation of types $[\![\_]\!]^I \in \Pi_{T\in\mathcal{T}}\mathbf{Red}^{\mathcal{T}}_{\mathcal{R}}$ is defined as follows:

- $[\![\mathsf{B}]\!]^I = I(\mathsf{B})$ if $\mathsf{B} \in \mathcal{B}$,
- $[\![T \Rightarrow U]\!]^I = \propto([\![T]\!]^I, [\![U]\!]^I)$.

We say that a type constant B is *basic* if its interpretation is SN^{B}, and that a symbol $\mathsf{f} : \vec{T} \Rightarrow \mathsf{B}$ is *basic* if B is basic. Let the *basic interpretation* be the interpretation I such that $I(\mathsf{B}) = \mathrm{SN}^{\mathsf{B}}$ for all $\mathsf{B} \in \mathcal{B}$.

We say that a term $t : T$ is *computable* wrt a base type interpretation I if $t \in [\![T]\!]^I$. A substitution σ is computable wrt a base type interpretation I if, for all $x \in \mathcal{X}$, $x\sigma \in [\![\tau(x)]\!]^I$. Note that, by (R3), variables are computable. Therefore, the identity substitution is always computable.

By definition of the interpretation of arrow types, a symbol $\mathsf{f} : \vec{T} \Rightarrow U$ is computable wrt a base type interpretation I if, for all $\vec{t} \in [\![\vec{T}]\!]^I$, $\mathsf{f}\vec{t} \in [\![U]\!]^I$. So, let Σ^I be the set of pairs $(\mathsf{f}, \vec{t})$ such that $\mathsf{f} : \vec{T} \Rightarrow U$ and $\vec{t} \in [\![\vec{T}]\!]^I$ (f may be partially applied in $\mathsf{f}\vec{t}$), and let Σ^I_{max} be the subset of Σ^I made of the pairs $(\mathsf{f}, \vec{t})$ such that $U \in \mathcal{B}$, that is, when f is maximally applied.

In the following, we may drop the exponent I when it is clear from the context.

Theorem 1 *The relation $\to_\beta \cup \to_{\mathcal{R}}$ terminates on well-typed terms if there is $I \in \Pi_{\mathsf{B}\in\mathcal{B}}\mathbf{Red}^{\mathsf{B}}_{\mathcal{R}}$ such that every non-basic undefined symbol and every defined symbol is computable.*

Proof. It suffices to prove that every well-typed term is computable. For dealing with abstraction, we prove the more general statement that, for all $t : T$ and computable σ, $t\sigma \in [\![T]\!]$, by induction on t. This indeed implies that every well-typed term is computable since the identity substitution is computable by (R3). We proceed by case on t:

- $t = x \in \mathcal{X}$. Then, $t\sigma = x\sigma \in [\![T]\!]$ since σ is computable.
- $t = uv$. By the induction hypothesis, $u\sigma \in [\![\tau(v) \Rightarrow T]\!]$ and $v\sigma \in [\![\tau(v)]\!]$. Therefore, by definition of $[\![\_]\!]$, $t\sigma = (u\sigma)(v\sigma) \in [\![T]\!]$.
- $t = \mathsf{f} : \vec{T} \Rightarrow \mathsf{A}$. If f is a defined symbol or a non-basic undefined symbol, then $t\sigma = \mathsf{f}$ is computable by assumption. Otherwise, f is a basic undefined symbol. By definition, it is computable if, for all $\vec{t} \in [\![\vec{T}]\!]$, $\mathsf{f}\vec{t} \in [\![\mathsf{A}]\!]$. Since f is basic, $[\![\mathsf{A}]\!] = \mathrm{SN}$. Now, one can easily prove that $\mathsf{f}\vec{t} \in \mathrm{SN}$, by induction on $\vec{t}$ with $\to_{\mathrm{prod}}$ as well-founded relation ($\vec{t} \in \mathrm{SN}$ by (R1)).
- $t = \lambda x u$. Wlog we can assume that σ is away from $\{x\}$. Hence, $t\sigma = \lambda x(u\sigma)$. By Corollary 1, $\lambda x(u\sigma) \in [\![T]\!] = [\![\tau(x) \Rightarrow \tau(u)]\!]$ if $\lambda x(u\sigma) : \tau(x) \Rightarrow \tau(U)$ and $(u\sigma)^v_x \in [\![\tau(u)]\!]$ for all $v \in [\![\tau(x)]\!]$. Since σ is away from $\{x\}$, we have $(u\sigma)^v_x = u\theta$ where $x\theta = v$ and $y\theta = y\sigma$ if $y \neq x$. Since θ is computable, by induction hypothesis, $u\theta \in [\![\tau(u)]\!]$. ■

Note that, with the basic interpretation, there is no non-basic undefined symbol. In Section 4.4, we will see another interpretation with which non-basic undefined symbols are computable.

4.2. Core computability closure

The next step consists then in proving that every defined symbol is computable, *i.e.* $\mathsf{f} \in [\![\tau(\mathsf{f})]\!]^I$ for all $\mathsf{f} \in \mathcal{D}(\mathcal{R})$. Assume that $\tau(\mathsf{f}) = \vec{T} \Rightarrow \mathsf{A}$. As just seen above, f is computable if, for all $\vec{t} \in [\![\vec{T}]\!]^I$, $\mathsf{f}\vec{t} \in I(\mathsf{A})$. Since $\mathsf{f} \in \mathcal{D}(\mathcal{R})$ and $|\vec{t}| \geq \alpha_{\mathsf{f}}$, $\mathsf{f}\vec{t}$ is neutral and, by (R3), belongs to $I(\mathsf{A})$ if all its reducts so do. The notion of *computability closure* enforces this property.

Definition 3 (Computability closure). A *computability closure* is a function CC mapping every $\mathsf{f} \in \mathcal{D}(\mathcal{R})$ and $\vec{l} \in \mathcal{L}^*$ such that $\mathsf{f}\vec{l}$ is well-typed to a set of well-typed terms.

Definition 4 (Valid computability closure – first definition). [14] A computability closure CC is *valid* wrt a base type interpretation I if it satisfies the following properties:

- it is *stable by substitution*: $t\sigma \in \mathrm{CC}_{\mathsf{f}}(\vec{l}\sigma)$ whenever $t \in \mathrm{CC}_{\mathsf{f}}(\vec{l})$;
- it *preserves computability* wrt I: every element of $\mathrm{CC}_{\mathsf{f}}(\vec{l})$ is computable whenever $\vec{l}$ so are.

Theorem 2 *Given $I \in \Pi_{\mathsf{B}\in\mathcal{B}}\mathbf{Red}^{\mathsf{B}}_{\mathcal{R}}$, every defined symbol is computable if there is a valid computability closure* CC *such that, for every rule $\mathsf{f}\vec{l} \to r \in \mathcal{R}$, we have $r \in \mathrm{CC}_{\mathsf{f}}(\vec{l})$.*

Proof. As just explained, it is sufficient to prove that, for all $(\mathsf{f}, \vec{t}) \in \Sigma_{\mathsf{max}}$ with $\mathsf{f} \in \mathcal{D}(\mathcal{R})$ and $\mathsf{f} : \vec{T} \Rightarrow \mathsf{A}$, every reduct t of $\mathsf{f}\vec{t}$ belongs to $[\![\mathsf{A}]\!]$. We proceed by well-founded induction on $\vec{t}$ with $\to_{\mathrm{prod}}$ as well-founded relation ($\vec{t} \in \mathrm{SN}$ by (R1)). There are two possible cases:

- There is $\vec{u}$ such that $t = \mathsf{f}\vec{u}$ and $\vec{t} \to_{\mathrm{prod}} \vec{u}$. By (R2), $\vec{u} \in [\![\vec{T}]\!]$. Therefore, by the induction hypothesis, $\mathsf{f}\vec{u} \in [\![\mathsf{A}]\!]$.

[14] We give a more general definition in Definition 5.

- There are $\vec{w}$, $\mathsf{f}\vec{l} \to r \in \mathcal{R}$ and σ such that $\vec{t} = \vec{l}\sigma\vec{w}$ and $t = r\sigma\vec{w}$. Since $r \in \mathrm{CC}_\mathsf{f}(\vec{l})$ and CC is stable by substitution, we have $r\sigma \in \mathrm{CC}_\mathsf{f}(\vec{l}\sigma)$. Since $\vec{l}\sigma$ are computable and CC preserves computability, we have $r\sigma$ computable. Finally, since $\vec{w}$ is computable, we have t computable. ■

Hence, the termination of $\to_\beta \cup \to_\mathcal{R}$ can be reduced to finding computability-preserving operations to define a computability closure. Among such operations, one can consider the ones of Figure 1 that directly follow from the definition or properties of computability.

Figure 1: Computability closure operations I

(arg)	$\{\vec{l}\} \subseteq \mathrm{CC}_\mathsf{f}(\vec{l})$
(app)	if $t \in \mathrm{CC}_\mathsf{f}(\vec{l})$, $t : U \Rightarrow V$, $u \in \mathrm{CC}_\mathsf{f}(\vec{l})$ and $u : U$, then $tu \in \mathrm{CC}_\mathsf{f}(\vec{l})$
(red)	if $t \in \mathrm{CC}_\mathsf{f}(\vec{l})$ and $t \to u$, then $u \in \mathrm{CC}_\mathsf{f}(\vec{l})$
(undef-basic)	if $\mathsf{g} \in \mathcal{F} - \mathcal{D}(\mathcal{R})$, $\mathsf{g} : \vec{T} \Rightarrow \mathsf{B}$, $\mathsf{B} \in \mathcal{B}$ and $[\![\mathsf{B}]\!] = \mathrm{SN}^\mathsf{B}$, then $\mathsf{g} \in \mathrm{CC}_\mathsf{f}(\vec{l})$

Theorem 3 *For all $I \in \Pi_{\mathsf{B}\in\mathcal{B}}\mathbf{Red}^\mathsf{B}_\mathcal{R}$, the smallest computability closure closed by the operations I is valid.*

Proof. • Stability by substitution. We prove that, for all $t \in \mathrm{CC}_\mathsf{f}(\vec{l})$, we have $t\sigma \in \mathrm{CC}_\mathsf{f}(\vec{l}\sigma)$, by induction on the definition of $\mathrm{CC}_\mathsf{f}(\vec{l})$.

(arg) By (arg), $\{\vec{l}\sigma\} \subseteq \mathrm{CC}_\mathsf{f}(\vec{l}\sigma)$.

(app) By the induction hypothesis, $\{t\sigma, u\sigma\} \subseteq \mathrm{CC}_\mathsf{f}(\vec{l}\sigma)$. Therefore, by (app), $(tu)\sigma = (t\sigma)(u\sigma) \in \mathrm{CC}_\mathsf{f}(\vec{l}\sigma)$.

(red) By the induction hypothesis, $t\sigma \in \mathrm{CC}_\mathsf{f}(\vec{l}\sigma)$. Since $\to$ is stable by substitution, $t\sigma \to u\sigma$. Therefore, by (red), $u\sigma \in \mathrm{CC}_\mathsf{f}(\vec{l}\sigma)$.

(undef-basic) By (undef-basic), $\mathsf{g}\sigma = \mathsf{g} \in \mathrm{CC}_\mathsf{f}(\vec{l}\sigma)$.

- Preservation of computability. Assume that $\vec{l}$ are computable. We prove that, for all $t \in \mathrm{CC}_\mathsf{f}(\vec{l})$, t is computable, by induction on the definition of $\mathrm{CC}_\mathsf{f}(\vec{l})$.

(arg) $\vec{l}$ are computable by assumption.

(app) By the induction hypothesis, t and u are computable. Therefore, by definition of $[\![U \Rightarrow V]\!]$, tu is computable.

(red) By the induction hypothesis, t is computable. Therefore, by (R2), u is computable.

(undef-basic) After the proof of Theorem 1, g is computable. ■

Therefore, using for I the basic interpretation, we get:

Corollary 2 *The relation $\to_\beta \cup \to_\mathcal{R}$ terminates on well-typed terms if, for every rule $\mathsf{f}\vec{l} \to r \in \mathcal{R}$, we have $r \in \mathrm{CC}_\mathsf{f}(\vec{l})$, where* CC *is the smallest computability closure closed by the operations I.*

4.3. Handling abstractions and bound variables

Consider now the following symbol definition, where T, U and V are any type:

$$\mathsf{o} : (U \Rightarrow V) \Rightarrow (T \Rightarrow U) \Rightarrow (T \Rightarrow V)$$
$$f \circ g \quad \to \quad \lambda x\ f\ (g\ x)$$

It cannot be handled by the operations I. By Corollary 1, a term λxt is computable if, for all $u \in [\![\tau(x)]\!]$, t^u_x is computable. We can therefore extend the previous core definition of CC by the rules of Figure 2 if we generalize validity as follows:

Definition 5 (Valid computability closure – extended definition). [15] A computability closure CC is *valid* wrt a base type interpretation I if:

- it is *stable by substitution*: $t\sigma \in \mathrm{CC}_\mathsf{f}(\vec{l\sigma})$ whenever $t \in \mathrm{CC}_\mathsf{f}(\vec{l})$ and σ is away from $\mathrm{FV}(t) - \mathrm{FV}(\vec{l})$;
- it *preserves computability* wrt I: $t\theta$ is computable whenever $t \in \mathrm{CC}_\mathsf{f}(\vec{l})$, $\vec{l}$ are computable, θ is computable and $\mathrm{dom}(\theta) \subseteq \mathrm{FV}(t) - \mathrm{FV}(\vec{l})$.

Note that, if $\mathrm{FV}(t) \subseteq \mathrm{FV}(\vec{l})$ as it is required for the right-hand side of a rule, then the above conditions reduce to the ones of Definition 4.

Figure 2: Computability closure operations II

(var)	$\mathcal{X} - \mathrm{FV}(\vec{l}) \subseteq \mathrm{CC}_\mathsf{f}(\vec{l})$
(abs)	if $t \in \mathrm{CC}_\mathsf{f}(\vec{l})$ and $x \in \mathcal{X} - \mathrm{FV}(\vec{l})$, then $\lambda xt \in \mathrm{CC}_\mathsf{f}(\vec{l})$

Theorem 4 *For all $I \in \Pi_{\mathsf{B}\in\mathcal{B}}\mathbf{Red}^{\mathsf{B}}_{\mathcal{R}}$, the smallest computability closure* CC *closed by the operations I and II is valid.*

Proof. We proceed as in Theorem 3 but only detail the new cases.

- Stability by substitution. We prove that, for all $t \in \mathrm{CC}_\mathsf{f}(\vec{l})$ and σ away from $\mathrm{FV}(t) - \mathrm{FV}(\vec{l})$, we have $t\sigma \in \mathrm{CC}_\mathsf{f}(\vec{l\sigma})$, by induction on the definition of $\mathrm{CC}_\mathsf{f}(\vec{l})$.

  (var) Let $x \in \mathcal{X} - \mathrm{FV}(\vec{l})$. Since σ is away from $\mathrm{FV}(x) - \mathrm{FV}(\vec{l})$, we have $x\sigma = x$ and $x \in \mathcal{X} - \mathrm{FV}(\vec{l\sigma})$. Therefore, by (var), $x \in \mathrm{CC}_\mathsf{f}(\vec{l\sigma})$.

  (abs) Wlog we can assume that σ is away from $\{x\}$. Hence, $(\lambda xt)\sigma = \lambda x(t\sigma)$ and, since $x \in \mathcal{X} - \mathrm{FV}(\vec{l})$, we have $x \in \mathcal{X} - \mathrm{FV}(\vec{l\sigma})$. Therefore, by (abs), $\lambda x(t\sigma) \in \mathrm{CC}_\mathsf{f}(\vec{l\sigma})$ for, by the induction hypothesis, $t\sigma \in \mathrm{CC}_\mathsf{f}(\vec{l\sigma})$.

- Preservation of computability. Assume that $\vec{l}$ are computable. We prove that, for all $t \in \mathrm{CC}_\mathsf{f}(\vec{l})$ and computable θ such that $\mathrm{dom}(\theta) \subseteq \mathrm{FV}(t) - \mathrm{FV}(\vec{l})$, we have $t\theta$ computable, by induction on $\mathrm{CC}_\mathsf{f}(\vec{l})$.

  (var) Let $x \in \mathcal{X} - \mathrm{FV}(\vec{l})$. Then, $x\theta$ is computable by assumption.

  (abs) Wlog we can assume that θ is away from $\{x\}$. Hence, $(\lambda xt)\theta = \lambda x(t\theta)$. Now, by Corollary 1, $\lambda x(t\theta)$ is computable if, for all computable $u : \tau(x)$, $(t\theta)^u_x$ is computable. Since θ is away from $\{x\}$, $(t\theta)^u_x = t\sigma$ where $x\sigma = u$ and $y\theta = y\sigma$ if $y \neq x$. Now, σ is computable and $\mathrm{dom}(\sigma) \subseteq \mathrm{FV}(t) - \mathrm{FV}(\vec{l})$ for $\mathrm{dom}(\theta) \subseteq \mathrm{FV}(\lambda xt) - \mathrm{FV}(\vec{l})$. Therefore, by the induction hypothesis, $t\sigma$ is computable. ■

For instance, we have $\{f, g\} \subseteq \mathrm{CC}_\mathsf{o}$ by (arg), $x \in \mathrm{CC}_\mathsf{o}$ by (var), $f(gx) \in \mathrm{CC}_\mathsf{o}(f, g)$ by (app) twice, and $\lambda xf(gx) \in \mathrm{CC}_\mathsf{o}(f, g)$ by (abs).

[15] This definition replaces the one given in Definition 4.

4.4. Handling basic subterms

Consider now the following definition on unary natural numbers (Peano integers):

$$\mathsf{z} : \mathsf{N}; \quad \mathsf{s} : \mathsf{N} \Rightarrow \mathsf{N}$$

$$\begin{array}{rcl} \mathsf{pred}\ \mathsf{z} & \to & \mathsf{z} \\ \mathsf{pred}\ (\mathsf{s}\ x) & \to & x \end{array}$$

In order to handle this definition, we need to extend the computability closure with some subterm operation. Unfortunately, $\rhd_{\mathsf{s}}$ does not always preserve computability as shown by the following example:[16]

$$\mathsf{f} : \mathsf{A} \Rightarrow (\mathsf{A} \Rightarrow \mathsf{B}); \quad \mathsf{c} : (\mathsf{A} \Rightarrow \mathsf{B}) \Rightarrow \mathsf{A}$$

$$\mathsf{f}\ (\mathsf{c}\ y) \quad \to \quad y$$

Indeed, with $w = \lambda x \mathsf{f} x x$, we have $w(\mathsf{c}w) \to_\beta \mathsf{f}(\mathsf{c}w)(\mathsf{c}w) \to_{\mathcal{R}} w(\mathsf{c}w) \to_\beta \ldots$ [Men87]. Therefore, $\mathsf{c}w \in \mathrm{SN}$ but $w \notin \propto(\mathrm{SN}, \mathrm{SN})$ since $w(\mathsf{c}w) \notin \mathrm{SN}$.

On the other hand, $\rhd_{\mathsf{s}}$ preserves termination. Hence, we can add the operation of Figure 3 for basic subterms (we omit the proof).

Figure 3: Computability closure operations III

(subterm-basic) if $t \in \mathrm{CC}_{\mathsf{f}}(\vec{l})$, $t \rhd_{\mathsf{s}} u : \mathsf{B} \in \mathcal{B}$ and $[\![\mathsf{B}]\!] = \mathrm{SN}^{\mathsf{B}}$, then $u \in \mathrm{CC}_{\mathsf{f}}(\vec{l})$

Hence, to handle the above predecessor function definition, it is enough to take $I(\mathsf{N}) = \mathrm{SN}^{\mathsf{N}}$.

In Sections 4.6 and 6, we will see other computability-preserving subterm operations.

4.5. Handling recursive functions

Consider now a simple recursive function definition:

$$+ : \mathsf{N} \Rightarrow \mathsf{N} \Rightarrow \mathsf{N}$$

$$\begin{array}{rcl} \mathsf{z} + y & \to & y \\ (\mathsf{s}\ x) + y & \to & \mathsf{s}\ (x + y) \end{array}$$

For handling such a recursive definition, and more generally mutually recursively defined functions, we need to extend $\mathrm{CC}_{\mathsf{f}}(\vec{l})$ with terms of the form $\mathsf{g}\vec{m}$. In order to ensure termination, we can try to use some well-founded DLQO $\geq$ on Σ (DLQOs are defined in Definition 1 and Σ just before Theorem 1) so that $(\mathsf{f}, \vec{l}) > (\mathsf{g}, \vec{m})$ and prove by induction on $>$ that CC preserves computability and defined function symbols are computable. However, we cannot consider arbitrary DLQOs. Indeed, since we consider curried symbols and, by (app), adding arguments preserves termination, the number of arguments is not a valid termination criterion as shown by the following example:

$$\mathsf{val} : (\mathsf{N} \Rightarrow \mathsf{N}) \Rightarrow \mathsf{N}; \quad \mathsf{f} : \mathsf{N} \Rightarrow (\mathsf{N} \Rightarrow \mathsf{N})$$

$$\begin{array}{rcl} \mathsf{val}\ x & \to & x\ \mathsf{z} \\ \mathsf{f}\ x\ \mathsf{z} & \to & \mathsf{val}\ (\mathsf{f}\ x) \end{array}$$

where $\mathsf{f}\ x\ \mathsf{z} \to \mathsf{val}\ (\mathsf{f}\ x) \to \mathsf{f}\ x\ \mathsf{z} \to \ldots$

We therefore need to consider DLQOs compatible with application:

[16] Note that the rule $\mathsf{f}\ (\mathsf{c}\ x) \to x$ means that c is injective and thus that, in a set-theoretical interpretation of types, the cardinality of the function space $\mathsf{A} \to \mathsf{B}$ is smaller than or equal to the cardinality of A, which is hardly possible if B is of cardinality greater than or equal to 2.

Definition 6 ($\mathcal{F}$-quasi-ordering). An *$\mathcal{F}$-quasi-ordering* is a DLQO on Σ. A relation R on Σ is:

- *compatible with application* if, for all $(\mathsf{f},\vec{l}\vec{v}),(\mathsf{g},\vec{m}\vec{w}) \in \Sigma_{\mathsf{max}}$, $(\mathsf{f},\vec{l}\vec{v})R(\mathsf{g},\vec{m}\vec{w})$ whenever $(\mathsf{f},\vec{l})R(\mathsf{g},\vec{m})$;
- *stable by substitution* if $(\mathsf{f},\vec{l}\sigma)R(\mathsf{g},\vec{m}\sigma)$ whenever $(\mathsf{f},\vec{l})R(\mathsf{g},\vec{m})$, σ is away from $\mathrm{FV}(\vec{m})-\mathrm{FV}(\vec{l})$ and $\vec{l}\sigma,\vec{m}\sigma$ are computable.

A simple way to get an $\mathcal{F}$-quasi-ordering compatible with application is to restrict comparisons to pairs $(\mathsf{f},\vec{t})$ such that f is maximally applied in $\mathsf{f}\vec{t}$. For instance, given a quasi-ordering $\geq$ on terms, the DLQO associated to:

- the identity relation on $\mathcal{F}$;
- for each equivalence class E modulo identity, the quasi-ordering $\geq_{\mathrm{prod}}$ (resp. $\geq_{\mathrm{mul}}$);
- for each symbol f, the identity function $\psi_{\mathsf{f}}(\vec{t}) = \vec{t}$ if f is maximally applied in $\mathsf{f}\vec{t}$;

is an $\mathcal{F}$-quasi-ordering compatible with application, that is stable by substitution if $\geq$ so is. For the sake of simplicity, we also denote such a DLQO by $\geq_{\mathrm{prod}}$ (resp. $\geq_{\mathrm{mul}}$) and its strict part by $>_{\mathrm{prod}}$ (resp. $>_{\mathrm{mul}}$). Hence, $\to^*_{\mathrm{prod}}$, $(\trianglerighteq_{\mathsf{s}})_{\mathrm{mul}}$ and, more generally, $(\to^*\trianglerighteq_{\mathsf{s}})_{\mathrm{mul}}$[17] are $\mathcal{F}$-quasi-orderings compatible with application and stable by substitution. For the sake of simplicity, we will denote $(\to^+)_{\mathrm{prod}}$ by $\to_{\mathrm{prod}}$.

Definition 7 (Valid $\mathcal{F}$-quasi-ordering). A quasi-ordering $\geq$ on terms is *compatible with reduction* if $>\cup\to$ is well-founded, where $>$ is the strict part of $\geq$. It is *valid* if, moreover, $>$ and the equivalence relation associated to $\geq$ are both stable by substitution.

An $\mathcal{F}$-quasi-ordering $\geq$ is *compatible with reduction* if $>\cup\to_{\mathrm{prod}}$ is well-founded on Σ_{max}, where $>$ is the strict part of $\geq$. It is *valid* if, moreover, $>$ and the equivalence relation associated to $\geq$ are both stable by substitution and compatible with application.

Note that $>\cup\to_{\mathrm{prod}}$ is required to be well-founded on Σ_{max}, that is, on pairs $(\mathsf{f},\vec{t})$ such that $\mathsf{f}:\vec{T}\Rightarrow\mathsf{A}$, $\mathsf{A}\in\mathcal{B}$ and $\vec{t}\in[\![\vec{T}]\!]$. Note also that, by (R1), $\to_{\mathrm{prod}}$ is well-founded on computable terms.

For instance, $\geq_{\mathrm{prod}}$ and $\geq_{\mathrm{mul}}$ are both valid if $\geq$ so is. In particular, $\to^*_{\mathrm{prod}}$ and $(\to^*\trianglerighteq_{\mathsf{s}})_{\mathrm{mul}}$ are both valid ($\triangleright_{\mathsf{s}}$ commutes with $\to$ since $\to$ is monotone).

In the next subsection, we will give another example of valid $\mathcal{F}$-quasi-ordering.

With a valid $\mathcal{F}$-quasi-ordering, we can add the operation of Figure 4.

Figure 4: Computability closure operations IV

(rec)	if $\mathsf{g}:\vec{M}\Rightarrow U$, $\vec{m}:\vec{M}$, $\vec{m}\in\mathrm{CC}_{\mathsf{f}}(\vec{l})$ and $(\mathsf{f},\vec{l})>(\mathsf{g},\vec{m})$, then $\mathsf{g}\vec{m}\in\mathrm{CC}_{\mathsf{f}}(\vec{l})$

Lemma 3 *Let $\geq$ be a valid $\mathcal{F}$-quasi-ordering, $(\mathsf{f},\vec{t})\in\Sigma_{\mathsf{max}}$ and assume that, for all $(\mathsf{g},\vec{u})\in\Sigma_{\mathsf{max}}$ such that $(\mathsf{f},\vec{t})>(\mathsf{g},\vec{u})$, $\mathsf{g}\vec{u}$ is computable. Then, for all $\vec{l}$, $\vec{w}$, t and θ such that $\vec{t}=\vec{l}\vec{w}$, θ is computable,* $\mathrm{dom}(\theta)\subseteq\mathrm{FV}(t)-\mathrm{FV}(\vec{l})$ *and $t\in\mathrm{CC}_{\mathsf{f}}(\vec{l})$, where* CC *is the smallest computability closure closed by the operations I to IV, we have $t\theta$ computable.*

Proof. We proceed as for Theorem 4 by proving that, for all $t\in\mathrm{CC}_{\mathsf{f}}(\vec{l})$ and computable θ such that $\mathrm{dom}(\theta)\subseteq\mathrm{FV}(t)-\mathrm{FV}(\vec{l})$, $t\theta$ is computable, by induction on CC, but only detail the new case:

[17] $\to^*\trianglerighteq_{\mathsf{s}}$ is the smallest quasi-ordering containing both $\to$ and $\triangleright_{\mathsf{s}}$. Its strict part on SN is $\to^+\cup\to^*\triangleright_{\mathsf{s}}$.

(rec) We have $\vec{m}\theta$ computable by the induction hypothesis. Since $\mathrm{dom}(\theta) \subseteq \mathrm{FV}(\mathsf{g}\vec{m}) - \mathrm{FV}(\vec{l})$, $\vec{l}\theta = \vec{l}$. Since $>$ is stable by substitution, we have $(\mathsf{f}, \vec{l}) > (\mathsf{g}, \vec{m}\theta)$. Assume now that $U = \vec{V} \Rightarrow \mathsf{B}$ and let $\vec{v} \in [\![\vec{V}]\!]$. Since $>$ is compatible with application, we have $(\mathsf{f}, \vec{l}\vec{w}) > (\mathsf{g}, \vec{m}\theta\vec{v})$. Hence, by assumption, $\mathsf{g}\vec{m}\theta\vec{v}$ is computable. Therefore, $\mathsf{g}\vec{m}\theta$ is computable. ■

Theorem 5 *The relation $\to_\beta \cup \to_\mathcal{R}$ terminates on well-typed terms if there are $I \in \Pi_{\mathsf{B}\in\mathcal{B}}\mathbf{Red}^{\mathsf{B}}_\mathcal{R}$ and a valid $\mathcal{F}$-quasi-ordering $\geq$ such that:*

- *every non-basic undefined symbol is computable;*
- *for every rule $\mathsf{f}\vec{l} \to r \in \mathcal{R}$, we have $r \in \mathrm{CC}_\mathsf{f}(\vec{l})$, where* CC *is the smallest computability closure closed by the operations I to IV.*

Proof. We follow the proof of Theorem 2 that, for all $(\mathsf{f}, \vec{t}) \in \Sigma_{\mathsf{max}}$ with $\mathsf{f} \in \mathcal{D}(\mathcal{R})$, every reduct t of $\mathsf{f}\vec{t}$ is computable, but proceed by induction on $> \cup \to_{\mathrm{prod}}$. There are two cases:

- There is $\vec{u}$ such that $t = \mathsf{f}\vec{u}$ and $\vec{t} \to_{\mathrm{prod}} \vec{u}$. By (R2), $\vec{u}$ is computable. Therefore, by the induction hypothesis, $\mathsf{f}\vec{u}$ is computable.
- There are $\vec{w}$, $\mathsf{f}\vec{l} \to r \in \mathcal{R}$ and σ such that $\vec{t} = \vec{l}\sigma\vec{w}$ and $t = r\sigma\vec{w}$. Since $r \in \mathrm{CC}_\mathsf{f}(\vec{l})$ and CC is stable by substitution (for $>$ is stable by substitution), we have $r\sigma \in \mathrm{CC}_\mathsf{f}(\vec{l}\sigma)$. Thus, by Lemma 3, $r\sigma$ is computable since, for all $(\mathsf{g}, \vec{u}) \in \Sigma_{\mathsf{max}}$, if $(\mathsf{f}, \vec{t}) > (\mathsf{g}, \vec{u})$, then $\mathsf{g}\vec{u}$ is computable by the induction hypothesis. ■

As a consequence, by taking the basic interpretation for I, we get:

Corollary 3 *The relation $\to_\beta \cup \to_\mathcal{R}$ terminates on well-typed terms if, for every rule $\mathsf{f}\vec{l} \to r \in \mathcal{R}$, we have $r \in \mathrm{CC}_\mathsf{f}(\vec{l})$, where* CC *is the smallest computability closure closed by the operations I to IV, and $\geq$ is any $\mathcal{F}$-quasi-ordering valid wrt the basic interpretation.*

In the first paper implicitly using the notion of computability closure for higher-order rewriting [JO91, JO97a], Jouannaud and Okada take the basic interpretation for I and define for CC a schema generalizing Gödel' system T recursion schema on Peano integers [Göd58] to arbitrary first-order data types. This schema is included in any computability closure closed by the operations I to IV with the $\mathcal{F}$-quasi-ordering $(\trianglerighteq_\mathsf{s})_{\mathrm{stat}}$ defined in the next subsection. The present inductive formulation first appeared in [BJO99, BJO02]. In Section 4.5.2, we provide various examples of systems that can be proved terminating by using this corollary.

4.5.1. Examples of valid $\mathcal{F}$-quasi-orderings

We have seen that a simple way to get an $\mathcal{F}$-quasi-ordering compatible with application is to only compare terms of base type. Another way is to always compare the same fixed subset of arguments by using a particular case of *arguments filtering system* (AFS) [AG00]:

Definition 8 (Arguments filtering system). A *filter* is a word on $\mathbb{N} - \{0\}$. The arity of a filter $\varphi = k_1 \dots k_n$ is $\|\varphi\|_\infty = max\{0, k_1, \dots, k_n\}$. A word w is compatible with a filter φ if $|w| \geq \|\varphi\|_\infty$. We denote by φ^A the function mapping every word $\vec{a} \in A^*$ compatible with $\varphi = k_1 \dots k_n$ to $a_{k_1} \dots a_{k_n}$. An *arguments filtering system* (AFS) is a function φ providing, for each $\mathsf{f} \in \mathcal{D}(\mathcal{R})$, a filter φ_f of arity $\|\varphi_\mathsf{f}\|_\infty \leq \alpha_\mathsf{f}$.

An AFS describes, for each symbol f, which arguments, and in which order, these arguments must be compared. For instance, if $\varphi_\mathsf{f} = 322$, then $\varphi^\mathcal{L}_\mathsf{f}(t_1t_2t_3\dots) = t_3t_2t_2$. Hence, when comparing $(\mathsf{f}, t_1t_2t_3)$ and $(\mathsf{f}, u_1u_2u_3u_4)$, one in fact compares $t_3t_2t_2$ and $u_3u_2u_2$ only.

Following [Der79, KL80], here is an $\mathcal{F}$-quasi-ordering allowing both multiset and lexicographic comparisons depending on a function $\mathrm{stat} : \mathcal{F} \to \{\mathrm{lex}, \mathrm{mul}\}$:

Definition 9 (Status $\mathcal{F}$-quasi-ordering). Given a quasi-ordering $\geq$ on terms, a quasi-ordering $\geq_\mathcal{F}$ on $\mathcal{F}$, an AFS φ and a function $\mathrm{stat} : \mathcal{F} \to \{\mathrm{lex}, \mathrm{mul}\}$ compatible with $\simeq_\mathcal{F}$, *i.e.* such that:

- $\mathrm{stat}_{\mathsf{f}} = \mathrm{stat}_{\mathsf{g}}$ whenever $\mathsf{f} \simeq_{\mathcal{F}} \mathsf{g}$;
- $|\varphi_{\mathsf{f}}| = |\varphi_{\mathsf{g}}|$ whenever $\mathsf{f} \simeq_{\mathcal{F}} \mathsf{g}$ and $\mathrm{stat}_{\mathsf{f}} = \mathrm{lex}$;

let $\geq_{\mathrm{stat}}$ be the DLQO associated to:

- the quasi-ordering $\geq_{\mathcal{F}}$ on $\mathcal{F}$;
- for each equivalence class E modulo $\simeq_{\mathcal{F}}$ of status mul (resp. lex), the quasi-ordering $\geq_{\mathrm{mul}}$ (resp. $\geq_{\mathrm{lex}}$);
- for each symbol f, the function $\psi_{\mathsf{f}}(\vec{t}) = \varphi^{\mathcal{L}}_{\mathsf{f}}(\vec{t})$.

The $\mathcal{F}$-quasi-ordering $\geq_{\mathrm{stat}}$ is valid whenever $\geq$ so is. In particular, $(\to^* \trianglerighteq_{\mathsf{s}})_{\mathrm{stat}}$ is valid.

4.5.2. Examples of termination proofs based on computability closure

With the above closure operations, one can already prove the termination of a large class of rewrite systems including:

- Gödel system T [Göd58]:

$$\mathsf{rec}^T_{\mathsf{N}} : \mathsf{N} \Rightarrow T \Rightarrow (\mathsf{N} \Rightarrow T \Rightarrow T) \Rightarrow T, \text{ for every type } T$$

$$\begin{array}{rcl} \mathsf{rec}^T_{\mathsf{N}}\ \mathsf{z}\ u\ v & \to & u \\ \mathsf{rec}^T_{\mathsf{N}}\ (\mathsf{s}\ x)\ u\ v & \to & v\ x\ (\mathsf{rec}^T_{\mathsf{N}}\ x\ u\ v) \end{array}$$

To give an example, let us detail why the right-hand side of the second rule is in the computability closure of the left-hand. We take the identity relation on $\mathcal{F}$ for $\geq_{\mathcal{F}}$, $\varphi_{\mathsf{rec}^T_{\mathsf{N}}} = 1$ as AFS (only the first argument of $\mathsf{rec}^T_{\mathsf{N}}$ will be used in comparisons), and $\mathrm{stat}_{\mathsf{rec}^T_{\mathsf{N}}} = \mathrm{lex}$. Then, we have $\{\mathsf{s}\ x, u, v\} \subseteq \mathrm{CC} = \mathrm{CC}_{\mathsf{rec}^T_{\mathsf{N}}}(\mathsf{s}\ x, u, v)$ by (arg), $x \in \mathrm{CC}$ by (subterm-basic), $\mathsf{rec}^T_{\mathsf{N}}\ x\ u\ v \in \mathrm{CC}$ by (rec) for $\mathsf{s}\ x \rhd_{\mathsf{s}} x$, and $v\ x\ (\mathsf{rec}^T_{\mathsf{N}}\ x\ u\ v) \in \mathrm{CC}$ by (app) twice.

- Ackermann's function:

$$\mathsf{ack} : \mathsf{N} \Rightarrow \mathsf{N} \Rightarrow \mathsf{N}$$

$$\begin{array}{rcl} \mathsf{ack}\ \mathsf{z}\ n & \to & \mathsf{s}\ n \\ \mathsf{ack}\ (\mathsf{s}\ m)\ \mathsf{z} & \to & \mathsf{ack}\ m\ (\mathsf{s}\ \mathsf{z}) \\ \mathsf{ack}\ (\mathsf{s}\ m)\ (\mathsf{s}\ n) & \to & \mathsf{ack}\ m\ (\mathsf{ack}\ (\mathsf{s}\ m)\ n) \end{array}$$

One can easily check that, for each rule, its right-hand side is in the computability closure of its left-hand side by taking $\varphi_{\mathsf{ack}} = 12$ and $\mathrm{stat}_{\mathsf{ack}} = \mathrm{lex}$.

- The following non-orthogonal set of rules for subtraction on unary natural numbers:

$$- : \mathsf{N} \Rightarrow \mathsf{N} \Rightarrow \mathsf{N}$$

$$\begin{array}{rcl} \mathsf{z} - x & \to & \mathsf{z} \\ x - \mathsf{z} & \to & x \\ (\mathsf{s}\ x) - (\mathsf{s}\ y) & \to & x - y \\ x - x & \to & \mathsf{z} \end{array}$$

can also be proved terminating by taking $\varphi_{\mathsf{sub}} = 1$ and $\mathrm{stat}_{\mathsf{sub}} = \mathrm{lex}$.

- Here is an example of a rule for computing subtyping constraints on simple types that requires multiset comparisons (take $\varphi_{\leq} = 12$ and $\mathrm{stat}_{\leq} = \mathrm{mul}$):

$$\mathsf{arrow} : \mathsf{T} \Rightarrow \mathsf{T} \Rightarrow \mathsf{T}; \quad \leq : \mathsf{T} \Rightarrow \mathsf{T} \Rightarrow \mathsf{C}; \quad \wedge : \mathsf{C} \Rightarrow \mathsf{C} \Rightarrow \mathsf{C}$$
$$\mathsf{arrow}\ x\ y \leq \mathsf{arrow}\ x'\ y' \quad \rightarrow \quad x' \leq x \wedge y \leq y'$$

- Here is an example of mutually defined functions requiring a true quasi-ordering on function symbols ($\mathsf{height_T} \simeq_{\mathcal{F}} \mathsf{height_F}$):

$$\mathsf{nil} : \mathsf{F}; \quad \mathsf{cons} : \mathsf{T} \Rightarrow \mathsf{F} \Rightarrow \mathsf{F}; \quad \mathsf{leaf} : \mathsf{T}; \quad \mathsf{node} : \mathsf{F} \Rightarrow \mathsf{T}; \quad \mathsf{height_T} : \mathsf{T} \Rightarrow \mathsf{N}; \quad \mathsf{height_F} : \mathsf{F} \Rightarrow \mathsf{N}$$
$$\begin{array}{rcl} \mathsf{height_F}\ \mathsf{nil} & \rightarrow & \mathsf{z} \\ \mathsf{height_F}\ (\mathsf{cons}\ t\ f) & \rightarrow & \mathsf{max}\ (\mathsf{height_T}\ t)\ (\mathsf{height_F}\ f) \\ \mathsf{height_T}\ \mathsf{leaf} & \rightarrow & \mathsf{z} \\ \mathsf{height_T}\ (\mathsf{node}\ f) & \rightarrow & \mathsf{s}\ (\mathsf{height_F}\ f) \end{array}$$

- Finally, here is an example showing that the operations I to IV can already handle rules with matching on basic *defined* symbols (we will see the case of *non-basic* defined symbols in Section 4.7):

$$\times : \mathsf{N} \Rightarrow \mathsf{N} \Rightarrow \mathsf{N}$$
$$\begin{array}{rcl} \mathsf{z} + y & \rightarrow & y \\ (\mathsf{s}\ x) + y & \rightarrow & x + (\mathsf{s}\ y) \\ (x + y) + z & \rightarrow & x + (y + z) \\ \mathsf{z} \times y & \rightarrow & \mathsf{z} \\ (\mathsf{s}\ x) \times y & \rightarrow & (x \times y) + y \\ (x + y) \times z & \rightarrow & (x \times z) + (y \times z) \end{array}$$

4.6. Handling higher-order subterms

The closure operations presented so far do not enable us to deal with functions defined by induction on higher-order inductive types, that is, on inductive types with constructors taking functions as arguments. Here are some examples:

- The "addition" on the following (type theoretic) ordinal notation [CPM88]:

$$\mathsf{zero} : \mathsf{O} \quad \mathsf{suc} : \mathsf{O} \Rightarrow \mathsf{O} \quad \mathsf{lim} : (\mathsf{N} \Rightarrow \mathsf{O}) \Rightarrow \mathsf{O} \quad + : \mathsf{O} \Rightarrow \mathsf{O} \Rightarrow \mathsf{O}$$
$$\begin{array}{rcl} \mathsf{zero} + y & \rightarrow & y \\ (\mathsf{suc}\ x) + y & \rightarrow & \mathsf{suc}\ (x + y) \\ (\mathsf{lim}\ x) + y & \rightarrow & \mathsf{lim}\ (\lambda n\ (x\ n) + y) \end{array}$$

- The computation of the prenex normal form in the predicate calculus [MN98]:

$$\bot, \top : \mathsf{F}; \quad \neg : \mathsf{F} \Rightarrow \mathsf{F}; \quad \wedge, \vee : \mathsf{F} \Rightarrow \mathsf{F} \Rightarrow \mathsf{F}; \quad \forall, \exists : (\mathsf{T} \Rightarrow \mathsf{F}) \Rightarrow \mathsf{F}$$
$$\begin{array}{rcll} (\forall\ P) \wedge Q & \rightarrow & \forall\ (\lambda x\ (P\ x) \wedge Q) & \\ \neg\ (\forall\ P) & \rightarrow & \exists\ (\lambda x \neg\ (P\ x)) & \ldots \end{array}$$

- The list of labels of a tree in breadth-first order using continuations (we only give the definition of one of the functions) [Hof95]:

$$\mathsf{nil} : \mathsf{L}; \quad \mathsf{cons} : \mathsf{N} \Rightarrow \mathsf{L} \rightarrow \mathsf{L}; \quad \mathsf{d} : \mathsf{C}; \quad \mathsf{c} : ((\mathsf{C} \Rightarrow \mathsf{L}) \Rightarrow \mathsf{L}) \rightarrow \mathsf{C}; \quad \mathsf{ex} : \mathsf{C} \Rightarrow \mathsf{L}$$
$$\begin{array}{rcl} \mathsf{ex}\ \mathsf{d} & \rightarrow & \mathsf{nil} \\ \mathsf{ex}\ (\mathsf{c}\ x) & \rightarrow & x\ \mathsf{ex} \end{array}$$

Indeed, in all these examples, there are two problems. First, we need the higher-order arguments of a computable function-headed term to be computable, *e.g.* x in $(\mathsf{lim}\ x)$. Second, we need to have a DLQO in which $(\mathsf{lim}\ x)$ is bigger than $(x\ n)$, where n is a bound variable.

But we have already seen in Section 4.4 that the first property is not always satisfied. Fortunately, under some conditions, it is possible to define an interpretation I satisfying this property by using the fact that $\mathbf{Red}_{\mathcal{R}}$ is a complete lattice (as seen in Section 3) on which, therefore, any monotone function has a fixpoint [Tar55]. Following [Mat98], two different definitions are possible that we illustrate with the type O of ordinals:[18]

- An *elimination-based* definition using recursor symbols. For instance, for O, one can define the family of recursor symbols $\mathsf{rec}_{\mathsf{O}}^T$ indexed by $T \in \mathcal{T}$ as follows:

$$\mathsf{rec}_{\mathsf{O}}^T : \mathsf{O} \Rightarrow T \Rightarrow (\mathsf{O} \Rightarrow T \Rightarrow T) \Rightarrow ((\mathsf{N} \Rightarrow \mathsf{O}) \Rightarrow (\mathsf{N} \Rightarrow T) \Rightarrow T) \Rightarrow T$$

$$\begin{array}{rcl} \mathsf{rec}_{\mathsf{O}}^T\ \mathsf{zero}\ u\ v\ w & \to & u \\ \mathsf{rec}_{\mathsf{O}}^T\ (\mathsf{suc}\ x)\ u\ v\ w & \to & v\ x\ (\mathsf{rec}_{\mathsf{O}}^T\ x\ u\ v\ w) \\ \mathsf{rec}_{\mathsf{O}}^T\ (\mathsf{lim}\ x)\ u\ v\ w & \to & w\ x\ (\lambda n\ \mathsf{rec}_{\mathsf{O}}^T\ (x\ n)\ u\ v\ w) \end{array}$$

  and define $I(\mathsf{O})$ as some fixpoint of the following monotone function:

$$\begin{array}{c} F_{\mathsf{O}}(X) = \{t \in \mathcal{L} \mid \forall T \in \mathcal{T}, \forall P \in \mathbf{Red}_{\mathcal{R}}^T, \forall u \in [\![\mathsf{A}]\!]^J, \forall v \in [\![\mathsf{O} \Rightarrow \mathsf{A} \Rightarrow \mathsf{A}]\!]^J, \\ \forall w \in [\![(\mathsf{N} \Rightarrow \mathsf{O}) \Rightarrow (\mathsf{N} \Rightarrow \mathsf{A}) \Rightarrow \mathsf{A}]\!]^J, \mathsf{rec}_{\mathsf{O}}^T\ t\ u\ v\ w \in [\![\mathsf{A}]\!]^J\} \\ \text{where } \mathsf{A} \text{ is a type constant distinct from } \mathsf{O} \text{ and } \mathsf{N}^{19}, \\ J(\mathsf{A}) = P,\ J(\mathsf{O}) = X \text{ and } J(\mathsf{N}) = I(\mathsf{N}) \end{array}$$

  The computability of $\mathsf{rec}_{\mathsf{O}}^T$ directly follows from the definition of $I(\mathsf{O})$. And for proving that x is computable if $(\mathsf{lim}\, x)$ so is, it suffices to take $T = \mathsf{O}$ and $w = \lambda x \lambda y x$ which is clearly computable. Indeed, in this case, $\mathsf{rec}_{\mathsf{O}}^{\mathsf{O}}(\mathsf{lim}\, x)uvw \to wx(\lambda n \mathsf{rec}_{\mathsf{O}}^{\mathsf{O}}(xn)uvw) \to x$ and we can conclude by (R2). Finally, proving that constructors are computable is no more complicated.

- An *introduction-based* definition using constructors only. In this approach, $I(\mathsf{O})$ is defined as some fixpoint of the following monotone function:

$$\begin{array}{c} F_{\mathsf{O}}(X) = \{t \in \mathrm{SN} \mid \forall u, (t \to^* \mathsf{suc}\ u \Rightarrow u \in X) \wedge (t \to^* \mathsf{lim}\ u \Rightarrow u \in [\![\mathsf{N} \Rightarrow \mathsf{O}]\!]^J)\} \\ \text{where } J(\mathsf{O}) = X \text{ and } J(\mathsf{N}) = I(\mathsf{N}) \end{array}$$

  In this case, the computability of constructor arguments directly follows from the definition of $I(\mathsf{O})$.

In [Mat98], p. 116-117, Matthes proves that, when using saturated sets, the introduction-based interpretation is included into the elimination-based interpretation and provides an example of type for which the two interpretations are distinct, by using the fact that some saturated sets are not stable by reduction. It is not too difficult to check that this cannot happen with reducibility candidates.

Anyway, in both cases, the monotony of F_{O} is due to the fact that O occurs only *positively* in the types of the arguments of the constructors of O, knowing that A occurs *positively* in $\mathsf{B} \Rightarrow \mathsf{A}$ and *negatively* in $\mathsf{A} \Rightarrow \mathsf{B}$. More formally:

Definition 10 (Positive and negative positions). Given a type T, the *positive* (resp. *negative*) positions of T, $\mathrm{Pos}^+(T)$ (resp. $\mathrm{Pos}^-(T)$), are the subsets of $\{0,1\}^*$ defined as follows:

[18] These definitions can be generalized to any *positive* inductive type (see Definition 10 just after) [Men87, BJO02].

[19] We assume that $\mathcal{B}$ is infinite. Alternatively, we could consider type variables.

- $\mathrm{Pos}^+(\mathsf{B}) = \{\varepsilon\}$
- $\mathrm{Pos}^-(\mathsf{B}) = \emptyset$
- $\mathrm{Pos}^+(T \Rightarrow U) = \{0w \mid w \in \mathrm{Pos}^-(T)\} \cup \{1w \mid w \in \mathrm{Pos}^+(U)\}$
- $\mathrm{Pos}^-(T \Rightarrow U) = \{0w \mid w \in \mathrm{Pos}^+(T)\} \cup \{1w \mid w \in \mathrm{Pos}^-(U)\}$

And the positions in a type T of the occurrences of a type constant B, $\mathrm{Pos}(\mathsf{B}, T)$, are:

- $\mathrm{Pos}(\mathsf{B}, \mathsf{B}) = \{\varepsilon\}$
- $\mathrm{Pos}(\mathsf{B}, \mathsf{C}) = \emptyset$ if $\mathsf{B} \neq \mathsf{C}$
- $\mathrm{Pos}(\mathsf{B}, T \Rightarrow U) = \{0w \mid w \in \mathrm{Pos}(\mathsf{B}, T)\} \cup \{1w \mid w \in \mathrm{Pos}(\mathsf{B}, U)\}$

This leads to the following common restrictions one can for instance find in the Calculus of Inductive Constructions (CIC)[20] [CPM88, Wer94] and proof assistants based on CIC like Agda [BDN09], Coq [Coq14] or Matita [ARCT11]:

Definition 11 (Standard inductive system). Given a set $\mathcal{R}$ of rewrite rules, the set of type constants $\mathcal{B}$ and the set of undefined function symbols $\mathcal{F} - \mathcal{D}(\mathcal{R})$ (constructors) form a *standard inductive system* if there is a well-founded quasi-ordering $\geq_{\mathcal{B}}$ on $\mathcal{B}$ such that, for all $\mathsf{B} \in \mathcal{B}$, $\mathsf{c} \in \mathcal{F} - \mathcal{D}(\mathcal{R})$, $\mathsf{c} : \vec{T} \Rightarrow \mathsf{B}$, $i \in [1, |\vec{T}|]$ and C occurring in T_i, either $\mathsf{C} <_{\mathcal{B}} \mathsf{B}$ or else $\mathsf{C} \simeq_{\mathcal{B}} \mathsf{B}$ and $\mathrm{Pos}(\mathsf{C}, T_i) \subseteq \mathrm{Pos}^+(T_i)$.

Taking a quasi-ordering instead of an ordering allows us to deal with mutually defined inductive types. However, in this case, one has to reason on equivalence classes modulo $\simeq_{\mathcal{B}}$ because, if $\mathsf{B} \simeq_{\mathcal{B}} \mathsf{C}$, then the interpretation of B and the interpretation of C have to be defined at the same time.

In such a system, one can define $I \in \Pi_{\mathsf{B} \in \mathcal{B}} \mathbf{Red}_{\mathcal{R}}^{\mathsf{B}}$ by induction on $>_{\mathcal{B}}$ and, for each equivalence class E modulo $\simeq_{\mathcal{B}}$, as some fixpoint S_E of a monotone function F_E (similar to the function F_{O} above) on the complete lattice $E \to \mathbf{Red}_{\mathcal{R}}$ ordered point-wise by inclusion ($I \leq J$ if, for all $\mathsf{B} \in E$, $I(\mathsf{B}) \subseteq J(\mathsf{B})$). See Lemma 14 in [BJO02] or Section 6.3 in [Bla05] for more details about that. For each type constant B, $I(\mathsf{B})$ is then defined as $S_{[\mathsf{B}]_{\simeq_{\mathcal{B}}}}(\mathsf{B})$.

With this interpretation, all the symbols $\mathsf{f} \in \mathcal{F} - \mathcal{D}(\mathcal{R})$ (constructors) are computable and one can add to the computability closure the operations of Figure 5.

Figure 5: Computability closure operations V for standard inductive systems (Definition 11)

(undef)	$\mathcal{F} - \mathcal{D}(\mathcal{R}) \subseteq \mathrm{CC}_{\mathsf{f}}(\vec{l})$
(subterm-undef)	if $\mathsf{g}\vec{t} \in \mathrm{CC}_{\mathsf{f}}(\vec{l})$, $\mathsf{g}\vec{t} : \mathsf{B}$ and $\mathsf{g} \in \mathcal{F} - \mathcal{D}(\mathcal{R})$, then $\{\vec{t}\} \subseteq \mathrm{CC}_{\mathsf{f}}(\vec{l})$

However, the stable subterm ordering is not sufficient to prove the termination of the systems given above. For instance, for the addition on O, starting from an argument of the form $(\mathsf{lim}\ x)$, we have a recursive call with an argument of the form $(x\ n)$ where n is a bound variable. Although x is a subterm of $(\mathsf{lim}\ x)$, $(x\ n)$ is not. In the case of continuations, this is even worse: starting from an argument of the form $(\mathsf{c}\ x)$, the function ex is applied to no argument but is itself argument of x...

If, for S_E, we take the *smallest* fixpoint of F_E (the set of fixpoints is itself a complete lattice [Tar55]), then it can be obtained by transfinite iteration [CC79]: there is an ordinal $\mathfrak{a}$ such that, for all $\mathsf{B} \in E$, $S_E = F_E^{\mathfrak{a}}(\bot_E)$ where $\bot_E$ is the smallest element of $E \to \mathbf{Red}_{\mathcal{R}}$ and $F_E^{\mathfrak{a}}$ is defined by transfinite induction:

[20] In fact, in CIC, inductive types are even restricted to *strictly-positive* inductive types (see Definition 13) for termination may be lost when considering some *polymorphic* non-strictly positive types [CPM88].

- $F_E^0(X) = X$
- $F_E^{\mathfrak{a}+1}(X) = F_E(F_E^{\mathfrak{a}}(X))$
- $F_E^{\mathfrak{l}}(X) = \bigcup\{F_E^{\mathfrak{a}}(X) \mid \mathfrak{a} < \mathfrak{l}\}$ if $\mathfrak{l}$ is a limit ordinal

This provides us with a notion of rank to compare computable terms:

Definition 12 (Rank of a computable term). The *rank* of a term $t \in I(\mathsf{B})$, $\mathrm{rk}_{\mathsf{B}}(t)$, is the smallest ordinal $\mathfrak{a}$ such that $t \in F^{\mathfrak{a}}_{[\mathsf{B}]_{\simeq_{\mathcal{B}}}}(\bot_{[\mathsf{B}]_{\simeq_{\mathcal{B}}}})(\mathsf{B})$. Let $\succeq_{\mathsf{B}}$ be the quasi-ordering on $I(\mathsf{B})$ such that $t \succeq_{\mathsf{B}} u$ if $\mathrm{rk}_{\mathsf{B}}(t) \geq \mathrm{rk}_{\mathsf{B}}(u)$.

Note that some terms may have a rank bigger than ω. For instance, with $\mathsf{i} : \mathsf{N} \Rightarrow \mathsf{O}$ defined by the rules $\mathsf{i}\ \mathsf{z} \to \mathsf{zero}$ and $\mathsf{i}(\mathsf{s}\ n) \to \mathsf{suc}(\mathsf{i}\ n)$, we have $\mathrm{rk}_{\mathsf{O}}(\mathsf{lim}\ \mathsf{i}) = \omega + 1$.

The relation $\succ_{\mathsf{B}}$ is compatible with reduction since $t \succeq_{\mathsf{B}} u$ whenever $t \in [\![\mathsf{B}]\!]$ and $t \to u$ (reduction cannot increase the rank of a term by (R2)). However, it is not stable by substitution. For instance, $\mathsf{s}\ \mathsf{z} >_{\mathsf{O}} y$ for $\mathrm{rk}_{\mathsf{O}}(\mathsf{s}\ \mathsf{z}) = 1$ and $\mathrm{rk}_{\mathsf{O}}(y) = 0$, but $\mathsf{s}\ \mathsf{z} <_{\mathsf{O}} \mathsf{s}\ (\mathsf{s}\ \mathsf{z})$ for $\mathrm{rk}_{\mathsf{O}}(\mathsf{s}\ (\mathsf{s}\ \mathsf{z})) = 2$. Restricting $t \succ_{\mathsf{B}} u$ to the cases where $\mathrm{FV}(u) \subseteq \mathrm{FV}(t)$ is not a solution since, with the addition on O, we have to compare $(\mathsf{lim}\ x)$ and $(x\ n)$. Instead, we will consider a sub-quasi-ordering of $\succeq_{\mathsf{B}}$ due to Coquand [Coq92] that is valid (in a sense that will be precised after the definition) and, in which, $(\mathsf{lim}\ x)$ is bigger than $(x\ n)$:

Definition 13 (Structural subterm ordering). The i-th argument of $\mathsf{c} : \vec{T} \Rightarrow \mathsf{B}$ is *strictly positive* if T_i is of the form $\vec{U} \Rightarrow \mathsf{C}$ with $\mathsf{C} \simeq \mathsf{B}$ and, for all D occurring in $\vec{U}$, $\mathsf{D} <_{\mathcal{B}} \mathsf{B}$. Let $\rhd_{\mathsf{s}}^{\mathsf{acc}}$ be the smallest sub-ordering of $\rhd_{\mathsf{s}}$ such that, for all $\mathsf{c} : \vec{T} \Rightarrow \mathsf{B}$, $\vec{t} : \vec{T}$ and $i \in [1, |\vec{t}|]$, we have $\mathsf{c}\vec{t} \rhd_{\mathsf{s}}^{\mathsf{acc}} t_i$ if the i-th argument of c is strictly positive. Given a term of the form $\mathsf{f}\vec{l}$, a term $t : T$ is *structurally bigger* than a term $u : U$, written $t >_{\mathsf{s}}^{\mathsf{f}\vec{l}} u$, if T and U are equivalent type *constants* and there are v and $\vec{x} \in \mathcal{X} - \mathrm{FV}(\vec{l})$ such that $t \rhd_{\mathsf{s}}^{\mathsf{acc}} v$ and $u = v\vec{x}$.[21] Finally, let $\geq_{\mathsf{s}}^{\mathsf{f}\vec{l}}$ be the reflexive closure of $>_{\mathsf{s}}^{\mathsf{f}\vec{l}}$.

For instance, $\mathsf{lim}\ x >_{\mathsf{s}}^{(\mathsf{lim}\ x)+y} x\ n$ for $\mathsf{lim}\ x : \mathsf{O}$, $x\ n : \mathsf{O}$, $\mathsf{lim}\ x \rhd_{\mathsf{s}}^{\mathsf{acc}} x$ and $n \in \mathcal{X} - \mathrm{FV}((\mathsf{lim}\ x) + y)$.

The relation $>_{\mathsf{s}}^{\mathsf{f}\vec{l}}$ is valid in the following generalized sense. First, $l_i\sigma >_{\mathsf{s}}^{\mathsf{f}\vec{l}\sigma} u\sigma$ whenever $l_i >_{\mathsf{s}}^{\mathsf{f}\vec{l}} u$, $\mathrm{dom}(\sigma) \subseteq \mathrm{FV}(\vec{l})$ and σ is away from $\mathrm{FV}(u) - \mathrm{FV}(\vec{l})$. Second, if $l_i >_{\mathsf{s}}^{\mathsf{f}\vec{l}} u$, $l_i : \mathsf{B}$ is computable, θ is computable and $\mathrm{dom}(\theta) \subseteq \mathrm{FV}(u) - \mathrm{FV}(\vec{l})$, then $u\theta : \mathsf{B}$ is computable and $l_i \succ_{\mathsf{B}} u\theta$ (see Lemma 18 in [BJO02] or Lemma 54 in [Bla05]). Hence, by adapting Lemma 3, we can provide an instance of Theorem 5 able to handle functions defined by induction on the structural subterm ordering, by using a status $\mathcal{F}$-quasi-ordering compatible with the rank ordering (that is defined on terms of the same computability predicate only):

Definition 14. An AFS φ and a map $\mathrm{stat} : \mathcal{F} \to \{\mathrm{lex}, \mathrm{mul}\}$ compatible with an equivalence relation $\simeq_{\mathcal{F}}$ on $\mathcal{F}$ are *compatible with the rank ordering* when the following conditions are satisfied:

- if E is an equivalence class modulo $\simeq_{\mathcal{F}}$ of status mul, then there is a constant type B^E such that, for all $\mathsf{f} \in E$ with $\mathsf{f} : \vec{T} \Rightarrow \mathsf{A}$ and $\varphi_{\mathsf{f}} = k_1 \dots k_n$, we have $T_{k_i} = \mathsf{B}^E$ for every $i \in [1, n]$;
- if E is an equivalence class modulo $\simeq_{\mathcal{F}}$ of status lex, then there is a sequence of constant types $\vec{\mathsf{B}}^E$ such that, for all $\mathsf{f} \in E$ with $\mathsf{f} : \vec{T} \Rightarrow \mathsf{A}$, we have $\varphi^{\mathcal{T}}_{\mathsf{f}}(\vec{T}) = \vec{\mathsf{B}}^E$.

Theorem 6 *In a standard inductive system, the relation $\to_\beta \cup \to_{\mathcal{R}}$ terminates on well-typed terms if there are a well-founded quasi-ordering $\geq_{\mathcal{F}}$ on $\mathcal{F}$, an AFS φ and a status map* stat *compatible with $\simeq_{\mathcal{F}}$ and the rank ordering such that, for every rule $\mathsf{f}\vec{l} \to r \in \mathcal{R}$, we have $r \in \mathrm{CC}_{\mathsf{f}}(\vec{l})$, where* CC *is the smallest computability closure closed by the operations I to V with, in (rec), $> = (\to^* \geq_{\mathsf{s}}^{\mathsf{f}\vec{l}})_{\mathrm{stat}}$.*[22]

[21] We could improve this definition by taking $\vec{x} \in \mathrm{CC}_{\mathsf{f}}(\vec{l})$ instead of $\vec{x} \in \mathcal{X} - \mathrm{FV}(\vec{l})$ only [BJO02, Bla06b].

[22] Here, we in fact consider a family of $\mathcal{F}$-quasi-orderings indexed by $\mathsf{f}\vec{l}$.

PROOF. By adapting Lemma 3, we can follow the proof of Theorem 5 but proceed by induction on the DLQO $\succ_{\mathrm{stat}}$ associated to:

- the quasi-ordering $\geq_{\mathcal{F}}$ on $\mathcal{F}$;
- for each equivalence class E modulo $\simeq_{\mathcal{F}}$ of status mul (resp. lex with $|\vec{\mathsf{B}}^E| = n$), the quasi-ordering $(\succ_{\mathsf{B}^E})_{\mathrm{mul}}$ (resp. $(\succ_{\mathsf{B}_1^E}, \ldots, \succ_{\mathsf{B}_n^E})_{\mathrm{lex}}$);
- for each symbol f, the function $\psi_{\mathsf{f}}(\vec{t}) = \varphi^{\mathcal{L}}_{\mathsf{f}}(\vec{t})$,

which is compatible with application and reduction. ■

Using this theorem, we can prove the termination of the first two examples given at the beginning of this section, or the rules defining the recursor on O. For instance, if we take $\varphi_+ = 1$ and $\mathrm{stat}_+ = \mathrm{lex}$, then $\{\mathsf{lim}\ x, y\} \subseteq \mathrm{CC} = \mathrm{CC}_+(\mathsf{lim}\ x, y)$ by (arg), $n \in \mathrm{CC}$ by (var) for $n \in \mathcal{X} - \mathrm{FV}(\mathsf{lim}\ x, y)$, $x \in \mathrm{CC}$ by (subterm-undef), $(x\ n) + y \in \mathrm{CC}$ by (rec) for $\mathsf{lim}\ x >_{\mathsf{s}}^{(\mathsf{lim}\ x)+y} x\ n$, $\lambda n(x\ n) + y \in \mathrm{CC}$ by (abs) for $n \in \mathcal{X} - \mathrm{FV}(\mathsf{lim}\ x, y)$, and $\mathsf{lim}\ (\lambda n(x\ n) + y) \in \mathrm{CC}$ by (undef).

This is however not sufficient to orient the rules defining the function ex above since the type for continuations is not strictly positive. To deal with non-strictly positive types, one needs to consider type constants with size annotations [Abe04, BFG+04, BR06].

4.7. Handling matching on non-basic defined symbols

We have already seen at the end of Section 4.5 that the rule (subterm-basic) allows to handle matching on *basic* defined symbols and not only *undefined* symbols (constructors) as in the previous section. Consider now the following set of rules on the strictly-positive type O of ordinals:

$$+ : \mathsf{O} \Rightarrow \mathsf{O} \Rightarrow \mathsf{O}$$

$$\begin{array}{rcl} \mathsf{zero} + y & \to & y \\ (\mathsf{suc}\ x) + y & \to & \mathsf{suc}\ (x + y) \\ (\mathsf{lim}\ x) + y & \to & \mathsf{lim}\ (\lambda n\ (x\ n) + y) \\ (x + y) + z & \to & x + (y + z) \end{array}$$

For handling the last rule (associativity), we need x and y to be computable whenever $x + y$ so is. But this does not follow from the interpretation of types in standard inductive systems which ensures that all the arguments of a computable term of the form $\mathsf{f}\vec{t}$ are computable if f is an *undefined* symbol (constructor) and some positivity conditions are satisfied. However, the introduction-based interpretation of types can be easily extended to include other symbols as long as the positivity conditions are satisfied. Moreover, these conditions can be checked for each argument independently. Hence the following definitions:

Definition 15 (Accessible argument). Given a well-founded quasi-ordering $\geq_{\mathcal{B}}$ on $\mathcal{B}$, the set of *accessible positions* of a symbol $\mathsf{f} : \vec{T} \Rightarrow \mathsf{B}$, $\mathrm{Acc}(\mathsf{f})$, is the set of integers $i \in [1, |\vec{T}|]$ such that, for all C occurring in T_i, either $\mathsf{C} <_{\mathcal{B}} \mathsf{B}$ or else $\mathsf{C} \simeq_{\mathcal{B}} \mathsf{B}$ and $\mathrm{Pos}(\mathsf{C}, T_i) \subseteq \mathrm{Pos}^+(T_i)$.

Let $\mathcal{M}(\mathcal{R})$ be the set of symbols f that are *strict* subterms of a left-hand side of a rule and for which $\mathrm{Acc}(\mathsf{f})$ is not empty (*matched* symbols with accessible arguments).

Then, for $I(\mathsf{O})$, we can take:

$$\begin{array}{l} F_{\mathsf{O}}(X) = \{t \in \mathrm{SN} \mid \forall \mathsf{f} \in \mathcal{M}(\mathcal{R}), \forall \vec{T}, \forall \vec{u}, \tau(\mathsf{f}) = \vec{T} \Rightarrow \mathsf{O} \wedge |\vec{T}| = |\vec{u}| \wedge t \to^* \mathsf{f}\vec{u} \Rightarrow \\ \qquad\qquad \forall i \in \mathrm{Acc}(\mathsf{f}), u_i \in [\![T_i]\!]^J\}^{23} \end{array}$$

where $J(\mathsf{O}) = X$ and $J(\mathsf{N}) = I(\mathsf{N})$. But, for $F_{\mathsf{O}}(X)$ to satisfy the property (R3), we need to exclude from the set of neutral terms the terms of the form $\mathsf{f}\vec{t}$ with $\mathsf{f} \in \mathcal{M}(\mathcal{R})$:

Definition 16 (Neutral term - New definition). Given a set $\mathcal{R}$ of rewrite rules, a term is *neutral*[24] if

[23] In a standard inductive system, all the arguments of a constructor are accessible ($\mathrm{Acc}(\mathsf{f}) = [1, |\vec{T}|]$ for every $\mathsf{f} \in \mathcal{F} - \mathcal{D}(\mathcal{R})$). In this case, this new definition of F_{O} is equivalent to the introduction-based definition given in the previous section if one takes $\mathrm{Acc}(\mathsf{f}) = \emptyset$ for every $\mathsf{f} \in \mathcal{D}(\mathcal{R})$, and assumes that $\mathcal{M}(\mathcal{R}) = \mathcal{F} - \mathcal{D}(\mathcal{R})$.

[24] This definition generalizes and replaces the one given in Definition 2.

it is of the form $x\vec{v}$, $(\lambda xt)u\vec{v}$ or $\mathsf{f}\vec{v}$ with $\mathsf{f} \in \mathcal{D}(\mathcal{R}) - \mathcal{M}(\mathcal{R})$ and $|\vec{v}| \geq \alpha_{\mathsf{f}} = sup\{|\vec{l}| \mid \exists r, \mathsf{f}\vec{l} \to r \in \mathcal{R}\}$.

Then, we have the following property:

Lemma 4 *A term $a : \mathsf{A}$ is computable* iff *all its reducts are computable* and, *for all $\mathsf{f} \in \mathcal{M}(\mathcal{R})$, $i \in \mathrm{Acc}(\mathsf{f})$ and $\vec{a}$ such that $a = \mathsf{f}\vec{a}$, a_i is computable.*

Proof. The only-if part directly follows from (R2) and the definition of the interpretation. For the if-part, first note that $a \in \mathrm{SN}$ for all its reducts are computable and $[\![\mathsf{A}]\!] \subseteq \mathrm{SN}$ by (R1). Now, let $\mathsf{f} \in \mathcal{M}(\mathcal{R})$, $i \in \mathrm{Acc}(\mathsf{f})$ and $\vec{a}$ such that $a \to^* \mathsf{f}\vec{a}$. If $a = \mathsf{f}\vec{a}$, then a_i is computable by assumption. Otherwise, there is a' such that $a \to a' \to^* \mathsf{f}\vec{a}$ and, since a' is computable by assumption, a_i is computable. ■

Figure 6: Computability closure operations V

(undef)	$\mathcal{F} - \mathcal{D}(\mathcal{R}) \subseteq \mathrm{CC}_{\mathsf{f}}(\vec{l})$
(subterm-acc)	if $\mathsf{g}\vec{t} \in \mathrm{CC}_{\mathsf{f}}(\vec{l})$, $\mathsf{g} \in \mathcal{M}(\mathcal{R})$, $\mathsf{g}\vec{t} : \mathsf{B} \in \mathcal{B}$ and $i \in \mathrm{Acc}(\mathsf{g})$, then $t_i \in \mathrm{CC}_{\mathsf{f}}(\vec{l})$

We can then generalize the closure operations of Figure 5 for standard inductive systems to the closure operations of Figure 6 (we omit the proof).

In addition, we can also give a syntactic criterion for the condition $[\![\mathsf{B}]\!] = \mathrm{SN}$ used in (undef-basic) and (subterm-basic) (see Lemma 16 in [BJO02] and Lemma 49 in [Bla05]):

Definition 17 (Basic type). A type constant is *basic* if its equivalence class modulo $\simeq_{\mathcal{B}}$ is basic. An equivalence class E is *basic* if for all $\mathsf{B} \in E$, $\mathsf{f} \in \mathcal{M}(\mathcal{R})$, $\mathsf{f} : \vec{T} \Rightarrow \mathsf{B}$, $i \in \mathrm{Acc}(\mathsf{f})$, T_i is a type constant C such that $\mathsf{C} \in E$ or else $\mathsf{C} <_{\mathcal{B}} \mathsf{B}$ and $[\mathsf{C}]_{\simeq_{\mathcal{B}}}$ is basic.

In particular, all first-order data types (natural numbers, lists of natural numbers, trees, etc.) are basic.

5. Rewriting modulo an equational theory

Rewriting theory has been initially introduced as a decision tool for equational theories [KB70]. Indeed, an equational theory $=_{\mathcal{E}}$, *i.e.* the smallest congruence containing $\mathcal{E}$, is decidable if there is a set $\mathcal{R}$ of rewrite rules such that $\to_{\mathcal{R}}$ terminates, is confluent, correct ($\mathcal{R} \subseteq =_{\mathcal{E}}$) and complete ($\mathcal{E} \subseteq =_{\mathcal{R}}$). Knuth and Bendix invented a completion procedure that, in case of success, builds such a set from $\mathcal{E}$. This procedure consists in orienting the equations of $\mathcal{E}$ (and those generated in the course of the procedure) in order to use them as rewrite rules.

Yet, some equations or sets of equations, like commutativity, or associativity and commutativity together (associativity alone is orientable), are not orientable (no orientation leads to a terminating relation). A solution consists then in reasoning modulo these unorientable equations $\mathcal{E}$ and consider class rewriting modulo $\mathcal{E}$, *i.e.* the relation[25] $t =_{\mathcal{E}} \to_{\mathcal{R}} u$ if there is t' such that $t =_{\mathcal{E}} t'$ and $t' \to_{\mathcal{R}} u$ [LB77, Hue80].

Another solution, preferred in practice since it makes rewriting more tractable, consists in considering rewriting with *matching modulo* $\mathcal{E}$, *i.e.* the relation $t \to_{\mathcal{R},\mathcal{E}} u$ if there are a position $p \in \mathrm{Pos}(t)$, a rule $l \to r \in \mathcal{R}$ and a substitution σ such that $t|_p =_{\mathcal{E}} l\sigma$ and $u = t[r\sigma]_p$ [PS81, JK86]. Efficient implementations of rewriting with matching modulo some equational theories like associativity and commutativity have been developed [Eke96, KM01] that are for instance used to simulate and verify systems modeling chemical reactions or cryptographic protocols.[26]

[25] We use the relation and notation of [Hue80] and not the relation $\to_{\mathcal{R}/\mathcal{E}} = =_{\mathcal{E}} \to_{\mathcal{R}} =_{\mathcal{E}}$ used in [JK86] for it makes proofs simpler, but the two relations are equivalent from the point of view of termination.

[26] Indeed, the order of molecules in a chemical formula is irrelevant, and the order in which messages are received may be different from the order messages are sent.

However, we will only consider class rewriting in this paper. But, since rewriting with matching modulo is included in class rewriting, the termination of class rewriting implies the termination of rewriting with matching modulo. Moreover, many confluence results for rewriting with matching modulo relies on termination of class rewriting [JK86].

We now show how the notions of computability and computability closure can be adapted to prove the termination of the relation $\to = \to_\beta \cup =_\mathcal{E} \to_\mathcal{R}$ for an important class of equational theories $=_\mathcal{E}$.

First note that, if there is a *non-regular*[27] equation (*e.g.* $x \times 0 = 0$), then the relation $=_\mathcal{E} \to_\mathcal{R}$ does not terminate. Indeed, if there are $g = d \in \mathcal{E}$, $x \in \mathrm{FV}(g) - \mathrm{FV}(d)$ and $l \to r \in \mathcal{R}$, then $d = d^l_x =_\mathcal{E} g^l_x \to^+_\mathcal{R} g^r_x =_\mathcal{E} d^r_x = d$ [JK86].

Similarly, if there is a regular *non-linear*[28] *collapsing*[29] equation (*e.g.* $x \wedge x = x$), then $=_\mathcal{E} \to_\mathcal{R}$ does not terminate either. Indeed, assume that $t = x \in \mathcal{E}$ and x freely occurs at two positions p and q in t, and let $t' = t[y]_p$ where $y \notin \mathrm{FV}(t)$. If $l \to r \in \mathcal{R}$, then $l =_\mathcal{E} t(^l_x) = t'(^l_x)(^l_y) \to_\mathcal{R} t'(^l_x)(^r_y) \dots$ [JK86].

We will therefore restrict our attention to regular and non-collapsing equations, thus excluding regular, linear and collapsing equations like $x + 0 = x$, which are easily oriented though.

We now extend the notion of neutral term by taking equations into account:

Definition 18 (Neutral term modulo equations). Given a set $\mathcal{R}$ of rewrite rules of the form $\mathsf{f}\vec{l} \to r$ and a set $\mathcal{E}$ of equations of the form $\mathsf{f}\vec{l} = \mathsf{g}\vec{m}$, a term is *neutral* if it is of the form $x\vec{v}$, $(\lambda x t)u\vec{v}$ or $\mathsf{f}\vec{v}$ with $\mathsf{f} \in \mathcal{D}(\mathcal{R} \cup \mathcal{E} \cup \mathcal{E}^{-1}) - \mathcal{M}(\mathcal{R} \cup \mathcal{E} \cup \mathcal{E}^{-1})$[30] and $|\vec{v}| \geq \alpha_\mathsf{f} = sup\{|\vec{l}| \mid \exists r, \mathsf{f}\vec{l} \to r \in \mathcal{R} \cup \mathcal{E} \cup \mathcal{E}^{-1}\}$. An equation $l = r$ is neutral if l is of the form $\mathsf{f}\vec{l}$, r is of the form $\mathsf{g}\vec{m}$, and both l and r are neutral. A set of equations $\mathcal{E}$ is neutral if every equation of $\mathcal{E}$ is neutral.

Note that this definition generalizes Definition 16 for they are identical if $\mathcal{E} = \emptyset$. Note also that, if $\mathsf{f}\vec{l}\vec{l} = r \in \mathcal{E}$, then $\mathsf{f}\vec{l}$ is *not* neutral.

Next, we need the set of neutral terms to be stable by $=_\mathcal{E}$. It is not sufficient to require that, for each equation $l = r \in \mathcal{E}$, l is neutral iff r is neutral, as shown by the following counter-example: for each equation $l = r \in \mathcal{E} = \{\mathsf{f} = \mathsf{g}, \mathsf{f}\ x = \mathsf{h}, \mathsf{g}\ x\ y = \mathsf{k}\}$, l is neutral iff r is neutral (f and g are not neutral, $\mathsf{f}\ x$ and h are neutral, and $\mathsf{g}\ x\ y$ and k are neutral), but $\mathsf{f}\ x =_\mathcal{E} \mathsf{g}\ x$, $\mathsf{f}\ x$ is neutral and $\mathsf{g}\ x$ is not neutral because $\alpha_\mathsf{f} = 1$, $\alpha_\mathsf{g} = 2$ and $\alpha_\mathsf{h} = \alpha_\mathsf{k} = 0$. However, it is sufficient to require $\mathcal{E}$ to be neutral:

Lemma 5 *If $\mathcal{E}$ is neutral, then the set of neutral terms is stable by $=_\mathcal{E}$.*

Proof. Note that $=_\mathcal{E}$ is the reflexive and transitive closure of $\leftrightarrow_\mathcal{E} = \to_\mathcal{E} \cup \leftarrow_\mathcal{E}$ (the symmetric closure of $\to_\mathcal{E}$). We can therefore proceed by induction on the number of $\leftrightarrow_\mathcal{E}$ steps, and prove that the set of neutral terms is stable by $\leftrightarrow_\mathcal{E}$. So, let t be a neutral term and assume that $t \leftrightarrow_\mathcal{E} t'$. We check that t' is neutral:

- $x\vec{v} \leftrightarrow_\mathcal{E} t'$. Since equations are of the form $\mathsf{f}\vec{l} = \mathsf{g}\vec{m}$, t' is of the form $x\vec{v}'$ with $\vec{v}(\leftrightarrow_\mathcal{E})_{\mathrm{prod}}\vec{v}'$.
- $(\lambda x t)u\vec{v} \leftrightarrow_\mathcal{E} t'$. Since equations are of the form $\mathsf{f}\vec{l} = \mathsf{g}\vec{m}$, t' is of the form $(\lambda x t')u'\vec{v}'$ with $tu\vec{v}(\leftrightarrow_\mathcal{E})_{\mathrm{prod}}t'u'\vec{v}'$.
- $\mathsf{f}\vec{v} \leftrightarrow_\mathcal{E} t'$ with $\mathsf{f} \in \mathcal{D}(\mathcal{R} \cup \mathcal{E} \cup \mathcal{E}^{-1})$ and $|\vec{v}| \geq \alpha_\mathsf{f}$. Either $t' = \mathsf{f}\vec{v}'$ and $\vec{v}(\leftrightarrow_\mathcal{E})_{\mathrm{prod}}\vec{v}'$, or there are $\vec{w}$, $\mathsf{f}\vec{l} = \mathsf{g}\vec{m} \in \mathcal{E}$ and σ such that $\vec{v} = \vec{l}\sigma\vec{w}$ and $t' = \mathsf{g}\vec{m}\sigma\vec{w}$. Since $\mathcal{E}$ is neutral, $|\vec{l}| \geq \alpha_\mathsf{f}$ and $|\vec{m}| \geq \alpha_\mathsf{g}$. Thus, t' is neutral. ■

Finally, we need $\mathrm{SN}(\to)$ and thus $\mathrm{SN}(\to_\beta)$ to be stable by $=_\mathcal{E}$. This can be achieved by requiring $=_\mathcal{E}$ to commute with $\to_\beta$. Putting every thing together, we get:

[27] $l = r$ is *regular* if $\mathrm{FV}(l) = \mathrm{FV}(r)$.
[28] $l = r$ is *linear* is both l and r are linear.
[29] $l = r$ is *collapsing* if $l \in \mathcal{X}$ or $r \in \mathcal{X}$
[30] See Definition 15.

Definition 19 (Admissible theory). A set of equations $\mathcal{E}$ is *admissible* if $\mathcal{E}$ is made of regular, non-collapsing and neutral equations only, and $=_\mathcal{E}$ commutes with $\to_\beta$.

In particular, $=_\mathcal{E}$ commutes with $\to_\beta$ if:

Lemma 6 *Given a set of equations $\mathcal{E}$ of the form $\mathsf{f}\vec{l} = \mathsf{g}\vec{m}$, $=_\mathcal{E}$ commutes with $\to_\beta$ if $\mathcal{E}$ satisfies all the following conditions:*

- *$\mathcal{E}$ is linear: $\forall l = r \in \mathcal{E}$, l and r are linear;*
- *$\mathcal{E}$ is regular: $\forall l = r \in \mathcal{E}$, $\mathrm{FV}(l) = \mathrm{FV}(r)$;*
- *$\mathcal{E}$ is algebraic: $\forall l = r \in \mathcal{E}$, l and r are algebraic*[31]*.*

Proof. We proceed by induction on the number of $\leftrightarrow_\mathcal{E}$-steps and show that, if $t \overset{p}{\leftarrow}_\mathcal{E} u \overset{q}{\to}_\beta v$, then $t \to_\beta =_\mathcal{E} v$. The case $\to_\mathcal{E} \to_\beta \subseteq \to_\beta =_\mathcal{E}$ is similar for conditions on equations are symmetric.

- $p \# q$. Then, $t \to_\beta \leftarrow_\mathcal{E} v$.
- $p < q$. Then, there are $l \to r$ and σ such that $u|_p = l\sigma$ and $t = u[r\sigma]_p$. Since r is algebraic and linear, there is $x \in \mathrm{FV}(l)$ such that $v = u[l\sigma']_p$, $x\sigma \to_\beta x\sigma'$ and, for all $y \neq x$, $y\sigma' = y\sigma$. Since r is regular and linear, $t \to_\beta u[r\sigma'] \leftarrow_\mathcal{E} v$.
- $p = q$. Not possible for equations are of the form $\mathsf{f}\vec{l} = \mathsf{g}\vec{m}$.
- $p > q$. There are x, a, b such that $u|_q = (\lambda x a)b$ and $v = u[a^b_x]_q$. Since equations are of the form $\mathsf{f}\vec{l} = \mathsf{g}\vec{m}$:
  - Either there is a' such that $a \to_\mathcal{E} a'$ and $t = u[(\lambda x a')b]_q$. Then, $t \to_\beta u[a'^b_x]_q \leftarrow_\mathcal{E} v$.
  - Or there is b' such that $b \to_\mathcal{E} b'$ and $t = u[(\lambda x a)b']_q$. Then, $t \to_\beta u[a^{b'}_x]_q \leftarrow^*_\mathcal{E} v$. ■

The condition of algebraicity could be slightly relaxed. For instance, the commutation of quantifiers necessary for ensuring the confluence of the rewrite rules computing the prenex normal form of a formula [MN98] commutes with $\to_\beta$:

$$\forall(\lambda x \forall(\lambda y P x y)) \quad = \quad \forall(\lambda y \forall(\lambda x P x y))$$

Now, we generalize the notion of computability to rewriting modulo some admissible theory:

Definition 20 (Computability predicates for rewriting modulo equations). Given an admissible set of equations $\mathcal{E}$ and a type T, let $\mathbf{Red}^T_{\mathcal{R}/\mathcal{E}}$ be the set of all the sets $P \subseteq \mathcal{L}^T$ such that:

(R1) $P \subseteq \mathrm{SN}(\to)$ where $\to = \to_\beta \cup =_\mathcal{E} \to_\mathcal{R}$;
(R2) P is stable by $\to \cup =_\mathcal{E}$;
(R3) if $t : T$ is neutral and $\to(t) \subseteq P$, then $t \in P$.

Note that $\mathbf{Red}^T_{\mathcal{R}/\emptyset} = \mathbf{Red}^T_\mathcal{R}$. We now check that the family $(\mathbf{Red}^T_{\mathcal{R}/\mathcal{E}})_{T \in \mathcal{T}}$ has all the required properties:

Lemma 7 *If $\mathcal{E}$ is an admissible set of equations and $T \in \mathcal{T}$, then $\mathbf{Red}^T_{\mathcal{R}/\mathcal{E}}$ is stable by non-empty intersection and admits SN^T as greatest element. Moreover, for all $T, U \in \mathcal{T}$, $P \in \mathbf{Red}^T_{\mathcal{R}/\mathcal{E}}$ and $Q \in \mathbf{Red}^U_{\mathcal{R}/\mathcal{E}}$, $\propto(P, Q) \in \mathbf{Red}^{T \Rightarrow U}_{\mathcal{R}/\mathcal{E}}$.*

[31] They contain no subterm of the form $\lambda x t$ or $x t$.

PROOF. The proof is similar to the one of Lemma 1. We only detail the cases that are different. We have $\mathrm{SN}^T \in \mathbf{Red}^T_{\mathcal{R}/\mathcal{E}}$ for $=_\mathcal{E}$ commutes with $\to_\beta$ and thus with $\to$. For the stability by $\propto$, we only detail (R3). Let $T, U \in \mathcal{T}$, $P \in \mathbf{Red}^T_{\mathcal{R}/\mathcal{E}}$ and $Q \in \mathbf{Red}^U_{\mathcal{R}/\mathcal{E}}$, $v : T \Rightarrow U$ neutral with $\to(v) \subseteq \propto(P, Q)$, and $t \in P$. We prove that $vt \in Q$ by well-founded induction on t ordered by $\to$ ($t \in \mathrm{SN}$ by (R1)). Since vt is neutral, by (R3), it suffices to prove that, for all w' such that $vt \to w'$, we have $w' \in Q$.

We first prove (a): there are v' and t' such that $w' = v't'$ with either $v \to v'$ and $t =_\mathcal{E} t'$, or $v =_\mathcal{E} v'$ and $t \to t'$. We proceed by case on $vt \to w'$:

- $vt \to_\beta w'$. Since v is neutral, it is not an abstraction and either $w' = v't$ with $v \to_\beta v'$, or $w' = vt'$ with $t \to_\beta t'$. Hence, (a) is satisfied.
- $vt =_\mathcal{E} w \to_\mathcal{R} w'$. We prove (a) by induction on the number of $\leftrightarrow_\mathcal{E}$-steps. If $vt = w$, then we are done. Assume now that, $vt \leftrightarrow_\mathcal{E} w =_\mathcal{E} \to_\mathcal{R} w'$.

  The term w can neither be a variable nor an abstraction for equations are of the form $\mathsf{f}\vec{l} = \mathsf{g}\vec{m}$.

  Assume that there are $\mathsf{f}\vec{l} = \mathsf{g}\vec{m} \in \mathcal{E}$ and σ such that $w = \mathsf{f}\vec{l}\sigma$ and $vt = \mathsf{g}\vec{m}\sigma$. Since $vt = \mathsf{g}\vec{m}\sigma$, there are $\vec{k}$ and k such that $\vec{m} = \vec{k}k$ and $v = \mathsf{g}\vec{k}\sigma$. But, then, v cannot be neutral for $|\vec{k}| < |\vec{m}| \leq \alpha_\mathsf{g}$.

  Therefore, there are a and b such that $w = ab$, $t =_\mathcal{E} a$ and $u =_\mathcal{E} b$. Now, by the induction hypothesis, there are v' and t' such that $w' = v't'$ with either $a \to_\mathcal{R} v'$ and $b =_\mathcal{E} t'$, or $a =_\mathcal{E} v'$ and $b \to_\mathcal{R} t'$. Hence, (a) holds.

If $v \to v'$ and $t =_\mathcal{E} t'$, then $w' = v't' \in Q$ for $v' \in \propto(P, Q)$ by assumption and $t' \in P$ by (R2). Otherwise, $v =_\mathcal{E} v'$ and $t \to t'$. Then, $t' \in P$ by (R2), and v' is neutral since neutral terms are stable by $=_\mathcal{E}$. Assume now that $v' \to v''$. Since $\mathcal{E}$ is admissible, $=_\mathcal{E}$ commutes with $\to_\beta$ and thus with $\to$. Hence, there is e such that $v \to e =_\mathcal{E} v''$, and $v'' \in \propto(P, Q)$ for $e \in \propto(P, Q)$ by assumption and $\propto(P, Q)$ satisfies (R2). Therefore, $\to(v') \subseteq \propto(P, Q)$ and, by the induction hypothesis on t', $w' = v't' \in Q$. ■

Figure 7: Computability closure operations I'

(mod) if $t \in \mathrm{CC}_\mathsf{f}(\vec{l})$ and $t =_\mathcal{E} u$, then $u \in \mathrm{CC}_\mathsf{f}(\vec{l})$

We now show how to extend Theorem 5.

Theorem 7 *Given a set of rules $\mathcal{R}$ and an admissible set of equations $\mathcal{E}$, the relation $\to = \to_\beta \cup =_\mathcal{E} \to_\mathcal{R}$ terminates on well-typed terms if there are $I \in \Pi_{\mathsf{B} \in \mathcal{B}} \mathbf{Red}^\mathsf{B}_{\mathcal{R}/\mathcal{E}}$ and a valid $\mathcal{F}$-quasi-ordering $\geq$ containing $(=_\mathcal{E})_{\mathrm{prod}}$ such that:*

- *every non-basic undefined symbol is computable;*
- *for every equation $\mathsf{f}\vec{l} = \mathsf{g}\vec{m} \in \mathcal{E}$, $\vec{m} \in \mathrm{CC}_\mathsf{f}(\vec{l})$, $\vec{l} \in \mathrm{CC}_\mathsf{g}(\vec{m})$ and $(\mathsf{f}, \vec{l}) \simeq (\mathsf{g}, \vec{m})$;*
- *for every rule $\mathsf{f}\vec{l} \to r \in \mathcal{R}$, $r \in \mathrm{CC}_\mathsf{f}(\vec{l})$;*

where CC *is the smallest computability closure closed by the operations I to IV, and I'.*

PROOF. We proceed as for Theorem 5 and show that, for all $(\mathsf{f}, \vec{t}) \in \Sigma_{\mathsf{max}}$, every reduct t of $\mathsf{f}\vec{t}$ is computable, by induction on $> \cup \to_{\mathrm{prod}}$. There are two cases:

1. $t = \mathsf{f}\vec{u}$ with $\vec{t}\ (\to_\beta)_{\mathrm{prod}}\ \vec{u}$. By (R2), $\vec{u}$ is computable. Therefore, by the induction hypothesis, t is computable.

2. Otherwise, $\mathsf{f}\vec{t} =_{\mathcal{E}} u \to_{\mathcal{R}} t$. We first prove by induction on the number of equational steps between $\mathsf{f}\vec{t}$ and u, that u is of the form $\mathsf{g}\vec{u}$ with $\vec{u}$ computable and $(\mathsf{f}, \vec{t}) \simeq (\mathsf{g}, \vec{u})$. If there is no equational step, this is immediate. So, assume that $\mathsf{f}\vec{t} =_{\mathcal{E}} u' \leftrightarrow_{\mathcal{E}} u$. By the induction hypothesis, u' is of the form $\mathsf{g}\vec{u}$ with $\vec{u}$ computable and $(\mathsf{f}, \vec{t}) \simeq (\mathsf{g}, \vec{u})$. The conditions on rules being symmetric, the case of $\leftarrow_{\mathcal{E}}$ is similar to the one of $\to_{\mathcal{E}}$ for which there are two cases:

(a) $u = \mathsf{g}\vec{v}$ with $\vec{u}\ (\to_{\mathcal{E}})_{\text{prod}}\ \vec{v}$. By (R2), $\vec{v}$ is computable and $(\mathsf{g}, \vec{u}) \simeq (\mathsf{g}, \vec{v})$ since $\simeq$ contains $(=_{\mathcal{E}})_{\text{prod}}$. Therefore, by transitivity, $(\mathsf{f}, \vec{t}) \simeq (\mathsf{g}, \vec{v})$.

(b) There are $\mathsf{g}\vec{l} = \mathsf{h}\vec{m} \in \mathcal{E}$, σ and $\vec{w}$ such that $\vec{u} = \vec{l}\sigma\vec{w}$ and $u = \mathsf{h}\vec{m}\sigma\vec{w}$. By assumption, $\vec{m} \in \text{CC}_{\mathsf{g}}(\vec{l})$ and $(\mathsf{g}, \vec{l}) \simeq (\mathsf{h}, \vec{m})$. Since $\simeq$ is stable by substitution, $(\mathsf{g}, \vec{l}\sigma) \simeq (\mathsf{h}, \vec{m}\sigma)$. Since $\simeq$ is compatible with application, $(\mathsf{g}, \vec{l}\sigma\vec{w}) \simeq (\mathsf{h}, \vec{m}\sigma\vec{w})$ and, by transitivity, $(\mathsf{f}, \vec{t}) \simeq (\mathsf{h}, \vec{m}\sigma\vec{w})$. Now, since $>$ is stable by substitution, CC is stable by substitution and $\vec{m}\sigma \in \text{CC}_{\mathsf{g}}(\vec{l}\sigma)$. Hence, by Lemma 3 and induction hypothesis, $\vec{m}\sigma$ is computable.

Now, for t, there are two possibilities:

(a) $t = \mathsf{g}\vec{v}$ with $\vec{u}\ (\to_{\mathcal{R}})_{\text{prod}}\ \vec{v}$. By (R2), $\vec{v}$ is computable and, by the induction hypothesis, t is computable.

(b) There are $\mathsf{g}\vec{l} \to r \in \mathcal{R}$, σ and $\vec{w}$ such that $\vec{u} = \vec{l}\sigma\vec{w}$ and $t = r\sigma\vec{w}$. By assumption, $r \in \text{CC}_{\mathsf{g}}(\vec{l})$. Since CC is stable by substitution, we have $r\sigma \in \text{CC}_{\mathsf{g}}(\vec{l}\sigma)$. Hence, by Lemma 3 and induction hypothesis, $r\sigma$ is computable. ■

5.1. $\mathcal{F}$-quasi-ordering compatible with permutative theories

We now define an $\mathcal{F}$-quasi-ordering satisfying the previous conditions for a general class of equational theories including *permutative*[32] axioms like associativity and commutativity together [LB77]. It is based on the notion of *alien subterm* used when studying the preservation (modularity) of properties like confluence and termination of the disjoint union of two rewrite systems [Gra91, Gra94, FJ94].

Definition 21 (Alien subterms). Let $\mathcal{M} = \mathbb{M}(\text{SN})$ be the set of finite multisets on SN. Given a set $E \subseteq \mathcal{F}$, the *E-alien subterms* (E-aliens for short) of a multiset $M \in \mathcal{M}$, $\text{Aliens}_E(M)$, is the multiset of terms defined by induction on $\rhd_{\mathcal{M}}$ as follows:

- $\text{Aliens}_E(\emptyset) = \emptyset$;
- $\text{Aliens}_E(M + N) = \text{Aliens}_E(M) + \text{Aliens}_E(N)$;
- $\text{Aliens}_E(\{\!|t|\!\}) = \text{Aliens}_E(\{\!|\vec{t}|\!\})$ if $t = \mathsf{f}\vec{t}$ and $\mathsf{f} \in E$,
- $\text{Aliens}_E(\{\!|t|\!\}) = \{\!|t|\!\}$ otherwise.

Given an equivalence $\simeq_{\mathcal{F}}$ on $\mathcal{F}$, a set of equations $\mathcal{E}$ is *compatible with $\simeq_{\mathcal{F}}$-aliens* if every equation of $\mathcal{E}$ is of the form $\mathsf{f}\vec{l} = \mathsf{g}\vec{m}$ with $\mathsf{f} \simeq_{\mathcal{F}} \mathsf{g}$ and $\text{Aliens}_{[\mathsf{f}]_{\simeq_{\mathcal{F}}}}(\vec{l}) = \text{Aliens}_{[\mathsf{g}]_{\simeq_{\mathcal{F}}}}(\vec{m})$.

Note that $\{\!|t|\!\} \trianglerighteq_{\mathcal{M}} \text{Aliens}_E(t)$. For instance, $\text{Aliens}_{\{+\}}((x + y) + (z \times (t + u))) = \{\!|x, y, z \times (t + u)|\!\}$.

Note also that, for all $\mathcal{F}$-quasi-orderings $\geq$, if $\mathcal{E}$ is compatible with $\simeq_{\mathcal{F}}$-aliens then, for all equations $\mathsf{f}\vec{l} = \mathsf{g}\vec{m} \in \mathcal{E}$, $(\mathsf{f}, \vec{l}) \simeq (\mathsf{g}, \vec{m})$, as required in Theorem 7.

We now prove some properties of aliens:

Lemma 8 *If θ is a substitution, then $\text{Aliens}_E(M\theta) = \varphi^{\theta}_E(\text{Aliens}_E(M))$, where $\varphi^{\theta}_E(M)$ is defined by induction on $\rhd_{\mathcal{M}}$ as follows:*

- $\varphi^{\theta}_E(\emptyset) = \emptyset$;

[32] An equation $l = r$ is *permutative* if every variable or function symbol has the same number of occurrences in l than it has in r. Such equations appear in algebra (permutative semi-groups), category theory (middle four exchange rule of Mac Lane), linear logic, the calculus of structures (medial rule) [Str07], automated deduction, ...

- $\varphi^\theta_E(M+N) = \varphi^\theta_E(M) + \varphi^\theta_E(N)$;
- $\varphi^\theta_E(\{\!|x\vec{u}|\!\}) = \mathrm{Aliens}_E(\{\!|\vec{t}|\!\}) + \varphi^\theta_E(\mathrm{Aliens}_E(\{\!|\vec{u}|\!\}))$ *if* $x \in \mathcal{X}$, $x\theta = \mathsf{f}\vec{t}$ *and* $\mathsf{f} \in E$;
- $\varphi^\theta_E(\{\!|a|\!\}) = \{\!|a\theta|\!\}$ *otherwise.*

Proof. By induction on M with $\rhd_{\mathcal{M}}$ as well-founded relation. □

In the following, we assume given a quasi-ordering $\geq_{\mathcal{F}}$ on $\mathcal{F}$, a set of equations $\mathcal{E}$ compatible with $\simeq_{\mathcal{F}}$-aliens, and an equivalence class E modulo $\simeq_{\mathcal{F}}$.

Lemma 9 *If* $M\,(=_{\mathcal{E}})_{\mathcal{M}}\,N$, *then* $\mathrm{Aliens}_E(M)\,(=_{\mathcal{E}})_{\mathcal{M}}\,\mathrm{Aliens}_E(N)$.

Proof. We proceed by induction on M with $\rhd_{\mathcal{M}}$ as well-founded relation:

- $M = N = \emptyset$. Then, $\mathrm{Aliens}_E(M) = \emptyset = \mathrm{Aliens}_E(N)$.
- $M = P + \{\!|a|\!\}$, $N = Q + \{\!|b|\!\}$, $P\,(=_{\mathcal{E}})_{\mathcal{M}}\,Q$ and $a =_{\mathcal{E}} b$. By the induction hypothesis, $\mathrm{Aliens}_E(P)\,(=_{\mathcal{E}})_{\mathcal{M}}$ $\mathrm{Aliens}_E(Q)$. We now prove that $\mathrm{Aliens}_E(\{\!|a|\!\})\,(=_{\mathcal{E}})_{\mathcal{M}}\,\mathrm{Aliens}_E(\{\!|b|\!\})$, by induction on the number of $\leftrightarrow_{\mathcal{E}}$ steps. And since conditions on equations are symmetric, it sufficient to prove that, if $a \rightarrow_{\mathcal{E}} b$, then $\mathrm{Aliens}_E(\{\!|a|\!\})\,(=_{\mathcal{E}})_{\mathcal{M}}\,\mathrm{Aliens}_E(\{\!|b|\!\})$:
  - $a = x\vec{u}$. Since equations are of the form $\mathsf{f}\vec{l} = \mathsf{g}\vec{m}$, there is $\vec{v}$ such that $b = x\vec{v}$. Therefore, $\mathrm{Aliens}_E(\{\!|a|\!\}) = \{\!|a|\!\}\,(=_{\mathcal{E}})_{\mathcal{M}}\,\{\!|b|\!\} = \mathrm{Aliens}_E(\{\!|b|\!\})$.
  - $a = (\lambda x s)\vec{u}$. Since equations are of the form $\mathsf{f}\vec{l} = \mathsf{g}\vec{m}$, there are t and $\vec{v}$ such that $b = (\lambda x t)\vec{v}$. Therefore, $\mathrm{Aliens}_E(\{\!|a|\!\}) = \{\!|a|\!\}\,(=_{\mathcal{E}})_{\mathcal{M}}\,\{\!|b|\!\} = \mathrm{Aliens}_E(\{\!|b|\!\})$.
  - $a = \mathsf{f}\vec{u}$, $b = \mathsf{f}\vec{v}$ and $\vec{u}\,(\rightarrow_{\mathcal{E}})_{\mathrm{prod}}\,\vec{v}$.
    - $\mathsf{f} \in E$. By the induction hypothesis, $\mathrm{Aliens}_E(\{\!|\vec{u}|\!\})\,(=_{\mathcal{E}})_{\mathcal{M}}\,\mathrm{Aliens}_E(\{\!|\vec{v}|\!\})$. Therefore, $\mathrm{Aliens}_E(\{\!|a|\!\})$ $(=_{\mathcal{E}})_{\mathcal{M}}\,\mathrm{Aliens}_E(\{\!|b|\!\})$.
    - $\mathsf{f} \notin E$. Then, $\mathrm{Aliens}_E(\{\!|a|\!\}) = \{\!|a|\!\}\,(=_{\mathcal{E}})_{\mathcal{M}}\,\{\!|b|\!\} = \mathrm{Aliens}_E(\{\!|b|\!\})$.
  - There are $\vec{w}$, $\mathsf{f}\vec{l} = \mathsf{g}\vec{m} \in \mathcal{E}$ and σ such that $a = \mathsf{f}\vec{l}\sigma\vec{w}$ and $b = \mathsf{g}\vec{m}\sigma\vec{w}$. Since $\mathcal{E}$ is compatible with $\simeq_{\mathcal{F}}$-aliens, $\mathsf{f} \simeq_{\mathcal{F}} \mathsf{g}$ and $\mathrm{Aliens}_E(\vec{l}) = \mathrm{Aliens}_E(\vec{m})$.
    - $\mathsf{f} \in E$. Then, $[\mathsf{f}]_{\simeq_{\mathcal{F}}} = E$, $\mathrm{Aliens}_E(\{\!|a|\!\}) = \mathrm{Aliens}_E(\{\!|\vec{l}\sigma|\!\}) + \mathrm{Aliens}_E(\vec{w})$ and $\mathrm{Aliens}_E(\{\!|b|\!\}) = \mathrm{Aliens}_E(\{\!|\vec{m}\sigma|\!\}) + \mathrm{Aliens}_E(\vec{w})$. By Lemma 8, $\mathrm{Aliens}_E(\{\!|\vec{l}\sigma|\!\}) = \varphi^\sigma_E(\mathrm{Aliens}_E(\{\!|\vec{l}|\!\}))$ and $\mathrm{Aliens}_E(\{\!|\vec{m}\sigma|\!\}) = \varphi^\sigma_E(\mathrm{Aliens}_E(\{\!|\vec{m}|\!\}))$. Therefore, $\mathrm{Aliens}_E(\{\!|a|\!\}) = \mathrm{Aliens}_E(\{\!|b|\!\})$.
    - $\mathsf{f} \notin E$. Then, $\mathsf{g} \notin E$ and $\mathrm{Aliens}_E(\{\!|a|\!\}) = \{\!|a|\!\}\,(=_{\mathcal{E}})_{\mathcal{M}}\,\{\!|b|\!\} = \mathrm{Aliens}_E(\{\!|b|\!\})$. ■

Lemma 10 *If* $M\,(=_{\mathcal{E}})_{\mathcal{M}}\,N$, *then* $\varphi^\theta_E(M)\,(=_{\mathcal{E}})_{\mathcal{M}}\,\varphi^\theta_E(N)$.

Proof. We proceed by induction on M with $\rhd_{\mathcal{M}}$ as well-founded relation:

- $M = N = \emptyset$. Then, $\varphi^\theta_E(M) = \emptyset = \varphi^\theta_E(N)$.
- $M = P + \{\!|a|\!\}$, $N = Q + \{\!|b|\!\}$, $P\,(=_{\mathcal{E}})_{\mathcal{M}}\,Q$ and $a =_{\mathcal{E}} b$. By the induction hypothesis, $\varphi^\theta_E(P)\,(=_{\mathcal{E}})_{\mathcal{M}}\,\varphi^\theta_E(Q)$. We now prove that $\varphi^\theta_E(a)\,(=_{\mathcal{E}})_{\mathcal{M}}\,\varphi^\theta_E(b)$:
  - Assume that $a = x\vec{u}$, $x\theta = \mathsf{f}\vec{w}$ and $\mathsf{f} \in E$. Since equations are of the form $\mathsf{f}\vec{l} = \mathsf{g}\vec{m}$, there is $\vec{v}$ such that $b = x\vec{v}$ and $\vec{u}(=_{\mathcal{E}})_{\mathrm{prod}}\vec{v}$. Hence, $\{\!|\vec{u}|\!\}\,(=_{\mathcal{E}})_{\mathcal{M}}\,\{\!|\vec{v}|\!\}$ and, by Lemma 9, $\mathrm{Aliens}_E(\{\!|\vec{u}|\!\})\,(=_{\mathcal{E}})_{\mathcal{M}}\,\mathrm{Aliens}_E(\{\!|\vec{v}|\!\})$. By the induction hypothesis, $\varphi^\theta_E(\mathrm{Aliens}_E(\{\!|\vec{u}|\!\}))\,(=_{\mathcal{E}})_{\mathcal{M}}\,\varphi^\theta_E(\mathrm{Aliens}_E(\{\!|\vec{v}|\!\}))$. Therefore, we have $\varphi^\theta_E(a) = \mathrm{Aliens}_E(\{\!|\vec{w}|\!\}) + \varphi^\theta_E(\mathrm{Aliens}_E(\{\!|\vec{u}|\!\}))\,(=_{\mathcal{E}})_{\mathcal{M}}\,\mathrm{Aliens}_E(\{\!|\vec{w}|\!\}) + \varphi^\theta_E(\mathrm{Aliens}_E(\{\!|\vec{v}|\!\})) = \varphi^\theta_E(b)$.
  - Otherwise, $\varphi^\theta_E(a) = \{\!|a|\!\}\,(=_{\mathcal{E}})_{\mathcal{M}}\,\{\!|b|\!\} = \varphi^\theta_E(b)$. ■

The ordering on terms that compares the alien subterms with $(\rhd_\mathsf{s})_\mathcal{M}$ is not stable by substitution as shown by the following example: $\text{Aliens}_{\{\mathsf{f}\}}(\{\!|xy|\!\}) = \{\!|xy|\!\}\,(\rhd_\mathsf{s})_\mathcal{M}\,\{\!|y|\!\} = \text{Aliens}_{\{\mathsf{f}\}}(\{\!|y|\!\})$ and $\text{Aliens}_{\{\mathsf{f}\}}(\{\!|\mathsf{f}y|\!\}) = \{\!|y|\!\}$. Therefore, we consider the following restriction of $\rhd_\mathsf{s}$:

Definition 22. Let $\rhd_\mathsf{s}^\mathsf{alg}$ be the smallest sub-ordering of $\rhd_\mathsf{s}$ such that, for all $\mathsf{f} : \vec{T} \Rightarrow U$, $\vec{t} : \vec{T}$ and $i \in [1, |\vec{t}|]$, $\mathsf{f}\vec{t} \rhd_\mathsf{s}^\mathsf{alg} t_i$. Let $\trianglerighteq_\mathsf{s}^\mathsf{alg}$ be its reflexive closure.

Lemma 11 *Let $\geq_\mathcal{F}$ be a quasi-ordering on $\mathcal{F}$ and $\mathcal{E}$ a set of equations such that:*

- *$\mathcal{E}$ is admissible and compatible with $\simeq_\mathcal{F}$-aliens;*
- *in each equivalence class modulo $=_\mathcal{E}$, the size of terms is bounded.*

Then, the DLQO $\dot{\geq}$ associated to:

- *the quasi-ordering $\simeq_\mathcal{F}$ on $\mathcal{F}$;*
- *for each equivalence class E modulo $\simeq_\mathcal{F}$, the quasi-ordering $(=_\mathcal{E} \trianglerighteq_\mathsf{s}^\mathsf{alg})_\mathcal{M}$;*
- *for each symbol f, the function $\psi_\mathsf{f}(\vec{t}) = \text{Aliens}_{[\mathsf{f}]_{\simeq_\mathcal{F}}}(\{\!|\vec{t}|\!\})$ if f is maximally applied in $\mathsf{f}\vec{t}$;*

is a valid $\mathcal{F}$-quasi-ordering containing $(=_\mathcal{E})_\text{prod}$.

Proof. The relation $=_\mathcal{E} \rhd_\mathsf{s}^\mathsf{alg}$ is well-founded since $\rhd_\mathsf{s}^\mathsf{alg}$ commutes with $=_\mathcal{E}$ (for $=_\mathcal{E}$ is monotone) and, in each equivalence class modulo $=_\mathcal{E}$, the size of terms is bounded (see the proof of Proposition 15 in [JK86]).

Therefore, the strict part of $\geq = (=_\mathcal{E} \trianglerighteq_\mathsf{s}^\mathsf{alg})$ is $> = (=_\mathcal{E} \rhd_\mathsf{s}^\mathsf{alg})$, which is well-founded, and its associated equivalence relation is $=_\mathcal{E}$.

Let $\dot{>}$ be the strict part of $\dot{\geq}$ and $\dot{=}$ be its associated equivalence relation.

- Compatibility of $\dot{>}$ with application. The relation $\dot{>}$ is compatible with application for it only compares pairs $(\mathsf{f}, \vec{t})$ such that f is maximally applied in $\mathsf{f}\vec{t}$.
- Compatibility of $\dot{>}$ with reduction. The relation $\rhd_\mathsf{s}^\mathsf{alg}$ commutes with $\to$ for $\to$ is monotone. The relation $=_\mathcal{E}$ commutes with $\to_\beta$ for $\mathcal{E}$ is admissible. The relation $=_\mathcal{E}$ trivially commutes with $=_\mathcal{E} \to_\mathcal{R}$. Therefore, $>$ commutes with $\to$. Since both $>$ and $\to$ are well-founded on SN, $> \cup \to$ is well-founded on SN. Hence, $\dot{>} \cup \to_\text{prod}$ is well-founded on Σ_max.
- Stability of $\dot{=}$ by substitution. It follows from the lemmas 8, 9 and 10.
- Stability of $\dot{>}$ by substitution. Let E be an $\simeq_\mathcal{F}$-equivalence class, and assume that $\text{Aliens}_E(\{\!|\vec{t}|\!\}) >_\mathcal{M} \text{Aliens}_E(\{\!|\vec{u}|\!\})$. Then, there are M, $P \neq \emptyset$, N and Q such that $\text{Aliens}_E(\{\!|\vec{t}|\!\}) = M + P$, $\text{Aliens}_E(\{\!|\vec{u}|\!\}) = N + Q$, $M\,(=_\mathcal{E})_\mathcal{M}\,N$ and, (*) for all $q \in Q$, there is $p \in P$ such that $p > q$. Now, let θ be a substitution. By Lemma 8, $\text{Aliens}_E(\{\!|\vec{t}\theta|\!\}) = \varphi_E^\theta(M) + \varphi_E^\theta(P)$ and $\text{Aliens}_E(\{\!|\vec{u}\theta|\!\}) = \varphi_E^\theta(N) + \varphi_E^\theta(Q)$. By Lemma 10, $\varphi_E^\theta(M)\,(=_\mathcal{E})_\mathcal{M}\,\varphi_E^\theta(N)$. We now prove that $\varphi_E^\theta(P) >_\mathcal{M} \varphi_E^\theta(Q)$. To this end, it suffices to prove that, for all $q \in Q$, there is $p \in P$ such that $\varphi_E^\theta(\{\!|p|\!\}) >_\mathcal{M} \varphi_E^\theta(\{\!|q|\!\})$, that is, $\text{Aliens}_E(\{\!|p\theta|\!\}) >_\mathcal{M} \text{Aliens}_E(\{\!|q\theta|\!\})$. So, let $q \in Q$. After (*), there is $p \in P$ such that $p > q$. By definition of $>$, there are $\vec{w}$ and $i \in [1, |\vec{w}|]$ such that $p =_\mathcal{E} \mathsf{k}\vec{w}$ and $w_i \trianglerighteq_\mathsf{s}^\mathsf{alg} q$. Since equations are of the form $\mathsf{f}\vec{l} = \mathsf{g}\vec{m}$, there are h and $\vec{v}$ such that $p = \mathsf{h}\vec{v}$. Since p is an E-alien, $\mathsf{h} \notin E$ and $\text{Aliens}_E(\{\!|p\theta|\!\}) = \{\!|p\theta|\!\}$. Since $>$ is stable by substitution, $p\theta > q\theta$ and thus $\{\!|p\theta|\!\} >_\mathcal{M} \{\!|q\theta|\!\}$. By definition of aliens, $\{\!|q\theta|\!\}\,(\rhd_\mathsf{s}^\mathsf{alg})_\mathcal{M}\,\text{Aliens}_E(\{\!|q\theta|\!\})$. Therefore, by transitivity, $\text{Aliens}_E(\{\!|p\theta|\!\}) >_\mathcal{M} \text{Aliens}_E(\{\!|q\theta|\!\})$. ■

Note that the terms of an equivalence class modulo $\mathcal{E}$ are of bounded size if, for instance, the equivalence classes modulo $\mathcal{E}$ are of finite cardinality. This is in particular the case of associativity and commutativity together.

5.2. Example of termination proof

As an example, we check that the conditions of Theorem 7 are satisfied by the set $\mathcal{R}$ of rules defining the addition on Peano integers given at the beginning of Section 4.5, and the following set $\mathcal{E}$ of equations (associativity and commutativity):

$$\begin{array}{rcl}(x+y)+z & = & x+(y+z)\\ x+y & = & y+x\end{array}$$

by taking the identity relation for $\simeq_{\mathcal{F}}$ and the $\mathcal{F}$-quasi-ordering $\dot{\geq}$ of Lemma 11.

The set of equations $\mathcal{E}$ is neutral. By Lemma 6, $=_{\mathcal{E}}$ commutes with $\to_\beta$ since $\mathcal{E}$ is linear, regular and algebraic. Therefore, $\mathcal{E}$ is admissible.

The set of equations $\mathcal{E}$ is compatible with $\simeq_{\mathcal{F}}$-aliens since, for the associativity equation, we have $+ \simeq_{\mathcal{F}} +$ and $\mathrm{Aliens}_{\{+\}}(\{\!|x+y,z|\!\}) = \{\!|x,y,z|\!\} = \mathrm{Aliens}_{\{+\}}(\{\!|x,y+z|\!\})$, and for the commutativity equation, we have $+ \simeq_{\mathcal{F}} +$ and $\mathrm{Aliens}_{\{+\}}(\{\!|x,y|\!\}) = \{\!|x,y|\!\} = \mathrm{Aliens}_{\{+\}}(\{\!|y,x|\!\})$.

Hence, by Lemma 11, $\dot{\geq}$ is a valid $\mathcal{F}$-quasi-ordering containing $(=_{\mathcal{E}})_{\mathrm{prod}}$ for, in each equivalence class modulo $=_{\mathcal{E}}$, the size of terms is bounded.

We now check the conditions on rules and equations:[33]

- For the first rule defining addition, we have $x \in \mathrm{CC}_+(0,x)$ by (arg).
- For the second rule defining addition, we have $x+y \in \mathrm{CC}_+(x,\mathsf{suc}\ y)$ by (rec) since $\mathrm{Aliens}_{\{+\}}(x,\mathsf{suc}\ y) = \{\!|x,\mathsf{suc}\ y|\!\} >_{\mathcal{M}} \mathrm{Aliens}_{\{+\}}(x,y) = \{\!|x,y|\!\}$, and thus $\mathsf{suc}\ (x+y) \in \mathrm{CC}_+(x,\mathsf{suc}\ y)$ by (undef) and (app).
- For the commutativity equation, we have $\{y,x\} \subseteq \mathrm{CC}_+(x,y)$ and $\{x,y\} \subseteq \mathrm{CC}_+(y,x)$ by (arg).
- Finally, for the associativity equation, we have $x \in \mathrm{CC}_+(x+y,z)$ by (arg) and (subterm-acc), and $y+z \in \mathrm{CC}_+(x+y,z)$ by (rec) since $\mathrm{Aliens}_{\{+\}}(x+y,z) = \{\!|x,y,z|\!\} >_{\mathcal{M}} \{\!|y,z|\!\} = \mathrm{Aliens}_{\{+\}}(y,z)$. Similarly, we have $x+y \in \mathrm{CC}_+(x,y+z)$ and $z \in \mathrm{CC}_+(x,y+z)$.

6. Rewriting with matching modulo $\beta\eta$

In this section, we extend the results of Section 4 to rewriting with matching modulo $\beta\eta$. Consider the following rewrite rule used for defining a formal derivation operator:

$$\begin{array}{c}\mathsf{sin},\mathsf{cos} : \mathsf{R} \Rightarrow \mathsf{R};\quad \times : \mathsf{R} \Rightarrow \mathsf{R} \Rightarrow \mathsf{R};\quad \mathsf{D} : (\mathsf{R} \Rightarrow \mathsf{R}) \Rightarrow (\mathsf{R} \Rightarrow \mathsf{R})\\ \mathsf{D}\ (\lambda x\ \mathsf{sin}\ (F\ x)) \quad \to \quad \lambda x\ (\mathsf{D}\ F\ x) \times (\mathsf{cos}\ (F\ x))\end{array}$$

Using matching modulo α-equivalence only, this rule can be applied neither to $\mathsf{D}(\mathsf{sin})$ nor to $\mathsf{D}(\lambda x\,\mathsf{sin}\,x)$. But it can be applied to $\mathsf{D}(\lambda x\,\mathsf{sin}\,x)$ if we use matching modulo β-equivalence, since $x \leftarrow_\beta (\lambda x x)x$,[34] and to $\mathsf{D}(\mathsf{sin})$ if we use matching modulo $\beta\eta$-equivalence, since $\mathsf{sin} \leftarrow_\eta \lambda x\,\mathsf{sin}\,x$.

Although matching modulo $\beta\eta$ is decidable [Sti09], it is of non-elementary complexity [Sta79] (while unification modulo $\beta\eta$ [Hue76] and matching modulo β are both undecidable [Loa03]). There is however an important fragment for which the complexity is linear: the class of β-normal η-long terms in which every free variable is applied to distinct bound variables, introduced by Miller for λProlog [Mil91, Qia93]. For instance, $\lambda x\,\mathsf{sin}(Fx)$, $\lambda x\lambda y Fyx$ and $\lambda x x(Fx)$ are patterns (if they are in η-long form), while Fx and $\lambda x Fxx$ are not patterns. However, in this paper, we will consider a slightly different class of terms:

Definition 23 (Patterns). A term t is a *pattern* if $t \in \mathcal{P}_{\mathrm{FV}(t)}$ where $\mathcal{P}_V$ is defined as follows:

[33] As already remarked, the condition $(\mathsf{f},\vec{l}) \doteq (\mathsf{g},\vec{m})$ for every equation $\mathsf{f}\vec{l} = \mathsf{g}\vec{m}$ follows from compatibility with $\simeq_{\mathcal{F}}$-aliens.

[34] In contrast with a common practice (Barendregt's variable convention [Bar92]), we often use the same variable name for both a bound and a free variable. Although it may be confusing at first sight, it has the advantage of avoiding some variable renamings.

- if $t \in \mathcal{P}_V$, then $\lambda x t \in \mathcal{P}_{V-\{x\}}$;
- if $\mathsf{f} \in \mathcal{F}$, $\mathsf{f} : \vec{T} \Rightarrow U$, $\vec{t} : \vec{T}$ and $\vec{t} \in \mathcal{P}_V$, then $\mathsf{f}\vec{t} \in \mathcal{P}_V$;
- if $x \in V$, $x : \vec{T} \Rightarrow U$, $\vec{t} : \vec{T}$ and $\vec{t}$ η-reduces to pairwise distinct variables not in V, then $x\vec{t} \in \mathcal{P}_V$.

Our definition excludes Miller patterns where a bound variable is applied to a free variable like $\lambda x x(Fx)$, which is not very common in practice. On the other hand, our patterns do not need to be in η-long form.

To apply the computability closure technique to rewriting with matching modulo $\beta\eta$, we need to prove that, if $\mathsf{f}\vec{t} =_{\beta\eta} \mathsf{f}\vec{l}\sigma \to_{\mathcal{R}} r\sigma$ and $\vec{t}$ are computable, then $\vec{l}\sigma$ is computable, so that $r\sigma$ is computable if $r \in \mathrm{CC}_{\mathsf{f}}(\vec{l})$. By confluence of $\to_{\beta\eta}$ [Pot78] and η-postponement ($\to_{\beta\eta}^* \subseteq \to_\beta^* \to_\eta^*$) [CF58, Tak95], for each $i \in [1, |\vec{l}|]$, there is u_i such that $t_i \to_\beta^* u_i =_\eta \leftarrow_\beta^* l_i\sigma$. By (R2), u_i is computable. Therefore, we are left to prove that, if u_i is computable and $u_i =_\eta \leftarrow_\beta^* l_i\sigma$, then $l_i\sigma$ is computable. While computability is preserved by η-equivalence (see Lemma 20 below), it cannot be the case for arbitrary β-expansions because β-expansion may introduce non-terminating subterms.

In [Mil91], Section 9.1, Miller remarks that, if $t =_{\beta\eta} l\sigma$, l is a pattern *à la* Miller, t and σ are in β-normal η-long form, then $t =_{\beta_0\eta} l\sigma$, where $\to_{\beta_0}$ is the restriction of $\to_\beta$ to redexes of the form $(\lambda x t)x$ (or, by α-equivalence, of the form $(\lambda x t)y$ with $y \in \mathcal{X}$ and $\tau(x) = \tau(y)$). So, when the left-hand sides of rules are patterns, matching modulo $\beta\eta$ reduces to matching modulo $\beta_0\eta$. We now check that $\to_{\beta_0\eta}$ terminates and is *strongly* confluent:

Lemma 12 *$\to_\eta$ terminates and is strongly confluent.*

Proof. The relation $\to_\eta$ terminates for it makes the size of terms decrease. Assume that $t \overset{p}{\leftarrow}_\eta u \overset{q}{\to}_\eta v$.

- $p \# q$. Then, $t \to_\eta \leftarrow_\eta v$.
- $p = q$. Then, $t = v$.
- $p > q$. Then, there are a and a' such that $u|_q = \lambda x a x$, $x \notin \mathrm{FV}(a)$, $v = u[a]_q$, $a \to_\eta a'$ and $t = u[\lambda x a' x]_q$. Since $\mathrm{FV}(a') \subseteq \mathrm{FV}(a)$, $x \notin \mathrm{FV}(a')$ and $t \to_\eta u[a']_q \leftarrow_\eta v$.
- $p < q$. By symmetry, $t \to_\eta \leftarrow_\eta v$. ■

Lemma 13 *$\to_{\beta_0\eta}$ terminates and is strongly confluent on well-typed terms.*

Proof. The relation $\to_{\beta_0\eta}$ terminates on well-typed terms for it is a sub-relation of $\to_{\beta\eta}$ which terminates on well-typed terms [Pot78]. Assume that $t \overset{p}{\leftarrow}_{\beta_0\eta} u \overset{q}{\to}_{\beta_0\eta} v$. If $p \# q$, then $t \to_{\beta_0\eta} u \leftarrow_{\beta_0\eta} v$.

- $t \overset{p}{\leftarrow}_\eta u \overset{q}{\to}_\eta v$. Then, $t = v$ or $t \to_\eta \leftarrow_\eta v$ by Lemma 12.
- $t \overset{p}{\leftarrow}_\eta u \overset{q}{\to}_{\beta_0} v$.
  - $p = q$. Not possible.
  - $p > q$. There is a such that $u|_q = (\lambda x a)x$ and $v = u[a]_q$.
    - $p = q0$. There is d such that $a = dx$, $x \notin \mathrm{FV}(d)$ and $t = u[dx]_q$. Thus, $t = v$.
    - $p > q0$. There is a' such that $a \to_\eta a'$ and $t = u[(\lambda x a')x]_q$. Thus, $t \to_{\beta_0} u[a']_q \leftarrow_\eta v$.
    - $p > q1$. Not possible.
  - $p < q$. There is a such that $u|_p = \lambda x a x$, $x \notin \mathrm{FV}(a)$ and $t = u[a]_p$.
    - $p0 = q$.[35] There is b such that $a = \lambda y b$ and $v = u[a^x_y]_p$. Since u is well-typed, $\tau(x) = \tau(y)$ and, by α-equivalence, we can assume wlog that $y = x$. Thus, $t = v$.

[35] This is exactly the situation of Nederpelt's counter-example to the confluence of $\to_{\beta\eta}$ on untyped Church-style λ-terms [Ned73], which is in fact a counter-example to the confluence of $\to_{\beta_0\eta}$ on untyped Church-style λ-terms.

* $p0 < q$. There is a' such that $a \to_{\beta_0} a'$ and $v = u[\lambda x a' x]_p$. Thus, $t \to_{\beta_0} u[a']_p \leftarrow_\eta v$.

- $t \overset{p}{\leftarrow}_{\beta_0} u \overset{q}{\to}_{\beta_0} v$.
  - $p = q$. Then, $v = t$.
  - $p < q$. There are a and a' such that $u|_p = (\lambda x a)x$, $t = u[a]_p$, $a \to_{\beta_0} a'$ and $v = u[(\lambda x a')x]_p$. Thus, $t \to_{\beta_0} u[a']_p \leftarrow_{\beta_0} v$.
  - $p > q$. By symmetry, $t \to_{\beta_0} \leftarrow_{\beta_0} v$. ■

Hence, if $t =_{\beta_0\eta} l\sigma$, then $t \to^*_{\beta_0\eta} \leftarrow^*_{\beta_0\eta} l\sigma$[36]. Therefore, we could try to prove that computability is preserved by β_0-expansion, all the more so since that, for matching modulo α-equivalence, computability is preserved by *head*-β_0-expansion as shown by Lemma 2 (a result that also holds with pattern matching modulo $\beta\eta$ under some conditions on $\mathcal{R}$ as we will see it in Lemma 17 below). But this does not seem easy to prove in general for two reasons.

First, a proof that u is computable whenever $t \leftarrow_{\beta_0} u$ and t is computable, by induction on the size of t does not seem to go through. Indeed, assume that $u = \lambda x s$. Then, $t = \lambda x r$ and $r \leftarrow_{\beta_0} s$. But $\lambda x s$ is computable if, for all $e \in [\![\tau(x)]\!]$, s^e_x is computable. Of course, r^e_x is computable but we generally do not have $r^e_x \leftarrow_{\beta_0} s^e_x$. We therefore need to consider not β_0-expansion but a restricted form of β-expansion that is stable by instantiation of the bound variables of a pattern:

Definition 24 (Leaf-β-expansion). The set $\mathrm{LPos}(t)$ of the (disjoint) *leaf positions* of a term t is defined as follows:

- $\mathrm{LPos}(t) = \{0^{n-1}1p | p \in \mathrm{LPos}(t_1)\} \cup \ldots \cup \{1p | p \in \mathrm{LPos}(t_n)\}$ if $t = \mathsf{f}\, t_1 \ldots t_n$ and $\mathsf{f} \in \mathcal{F}$;
- $\mathrm{LPos}(t) = \{0p | p \in \mathrm{LPos}(u)\}$ if $t = \lambda x u$;
- $\mathrm{LPos}(t) = \{\varepsilon\}$ otherwise.

Given a term v and a leaf position $p \in \mathrm{LPos}(v)$, let the relation of *β-leaf-expansion* wrt v at position p be the relation $t \leftarrow_{\beta,v,p} u$ if there are $\vec{t}, a, x, e, \vec{b}$ such that $t = v[\vec{t}]_{\vec{q}}[a^e_x \vec{b}]_p$ and $u = v[\vec{t}]_{\vec{q}}[(\lambda x a) e \vec{b}]_p$, where $\vec{q}$ are all the leaf positions of v distinct from p.

Second, since we do not consider rewriting on terms in β-normal form, $l\sigma$ can contain some arbitrary β-redex $(\lambda x a)b$ which, after some $\beta_0\eta$-reductions, becomes a β_0-redex because $b \to^*_{\beta_0\eta} x$. However, if l is a pattern then such a β-redex can only occur in σ. Therefore, it is not needed to reduce it for checking that t matches l modulo $\beta\eta$. For the sake of simplicity, we will enforce this property in the definition of rewriting itself by using the notion of *valuation* used for defining rewriting in CRSs[37] [KvOvR93]:

Definition 25 (Valuation). A substitution σ is *valid* wrt a term t if, for all $p \in \mathrm{LPos}(t)$, x and $t_1, \ldots, t_n$ such that $t|_p = x t_1 \ldots t_n$, there are pairwise distinct variables $y_1, \ldots, y_n$ and a term a such that $x\sigma = \lambda y_1 \ldots \lambda y_n a$. Let the *valuation* of a term t by a substitution σ, written $\hat{\sigma}(t)$, be the term:

- $\lambda x \hat{\sigma}(u)$ if $t = \lambda x u$ and σ is away from $\{x\}$;
- $\mathsf{f}\ \hat{\sigma}(t_1) \ldots \hat{\sigma}(t_n)$ if $t = \mathsf{f}\, t_1 \ldots t_n$;
- $a\{y_1 \mapsto t_1, \ldots, y_n \mapsto t_n\}$ if $t = x t_1 \ldots t_n$, $x\sigma = \lambda y_1 \ldots \lambda y_n a$ and $y_1, \ldots, y_n$ are pairwise distinct variables.

Lemma 14 *If l is a pattern and $p_1, \ldots, p_n$ are the leaf positions of l then, for all substitutions σ valid wrt l, we have $\widehat{\sigma}(l) \leftarrow^*_{\beta,l,p_1} \ldots \leftarrow^*_{\beta,l,p_n} l\sigma$.*

[36] Note that $\to_\eta$ cannot be postponed after $\to_{\beta_0}$ as shown by the following example: $(\lambda x a)(\lambda y x y) \to_\eta (\lambda x a)x \to_{\beta_0} a$.

[37] In CRSs, $\to_{\mathcal{R}}$ is defined as the closure of $\mathcal{R}$ by context and valuation (extended to all terms).

PROOF. Let p be a leaf position of l. By definition of patterns, there are terms $\vec{t}$ and pairwise distinct variables $x, \vec{y}$ such that $l|_p = x\vec{t}$, $x \in \mathrm{FV}(l)$ and $\vec{t} \to_\eta^* \vec{y}$. Since σ is valid wrt l, there is a such that $x\sigma = \lambda\vec{y}a$ and $\widehat{\sigma}(l)|_p = a\{y_1 \mapsto t_1, \ldots, y_n \mapsto t_n\} \leftarrow_{\beta,l,p}^* (\lambda\vec{y}a)\vec{t} = l\sigma|_p$. ■

Hence, valuation preserves typing: $\tau(\widehat{\sigma}(t)) = \tau(t)$.

We now introduce our definition of rewriting with matching modulo $\beta\eta$:

Definition 26 (Rewriting with pattern matching modulo $\beta\eta$). Given a set $\mathcal{R}$ of rewrite rules of the form $\mathsf{f}\vec{l} \to r$ with $\mathsf{f}\vec{l}$ a pattern, let $t \to_{\mathcal{R},\beta\eta} u$ if there are $p \in \mathrm{Pos}(t)$, $l \to r \in \mathcal{R}$ and σ such that $\tau(t|_p) = \tau(l)$, σ is valid wrt l, $t|_p =_\eta \widehat{\sigma}(l)$ and $u = t[r\sigma]_p$.

Lemma 15 *The relation $\to_{\mathcal{R},\beta\eta}$ is monotone and stable by substitution.*

PROOF. Monotony is straightforward. We check that it is stable by substitution. Assume that $t \to_{\mathcal{R},\beta\eta} u$ and let θ be a substitution. There are $p \in \mathrm{Pos}(t)$, $l \to r \in \mathcal{R}$ and σ such that $\tau(t|_p) = \tau(l)$, σ is valid wrt l, $t|_p =_\eta \widehat{\sigma}(l)$ and $u = t[r\sigma]_p$. We have $\tau(t\theta|_p) = \tau(t|_p) = \tau(l)$, $\sigma\theta$ valid wrt l and $t\theta|_p =_\eta \widehat{\sigma}(l)\theta$. We now prove that $\widehat{\sigma}(l)\theta =_\eta \widehat{\sigma\theta}(l)$. Let $q \in \mathrm{LPos}(l)$. Since l is a pattern, $l|_q = x\vec{t}$ where x and $\vec{t}{\downarrow_\eta}$ are pairwise distinct variables and $\{\vec{t}{\downarrow_\eta}\} \subseteq \mathrm{BV}(l, q)$. Since σ is valid wrt l, there is a such that $x\sigma = \lambda\vec{y}a$ and $\widehat{\sigma}(l|_q) =_\eta a$. Wlog we can assume that θ is away from $\{\vec{y}\}$. Therefore, $x\sigma\theta = \lambda\vec{y}a\theta$ and $\widehat{\sigma}(l|_q)\theta =_\eta a\theta =_\eta \widehat{\sigma\theta}(l|_q)$. Therefore, $t\theta \to_{\mathcal{R},\beta\eta} u\theta$. ■

6.1. Definition of computability

Computability is straightforwardly extended to this new form of rewriting as follows:

Definition 27 (Computability predicates for rewriting with matching modulo $\beta\eta$). Given a set $\mathcal{R}$ of rewrite rules of the form $\mathsf{f}\vec{l} \to r$ with $\mathsf{f}\vec{l}$ a pattern, a term is *neutral* if it is of the form $x\vec{v}$, $(\lambda xt)u\vec{v}$ or $\mathsf{f}\vec{v}$ with $\mathsf{f} \in \mathcal{D}(\mathcal{R}) - \mathcal{M}(\mathcal{R})$[38] and $|\vec{v}| \geq \alpha_\mathsf{f} = sup\{|\vec{l}| \mid \exists r, \mathsf{f}\vec{l} \to r \in \mathcal{R}\}$. Given a type T, let $\mathbf{Red}^T_{\mathcal{R},\beta\eta}$ be the set of all the sets $P \subseteq \mathcal{L}^T$ such that:

(R1) $P \subseteq \mathrm{SN}(\to)$ where $\to = \to_\beta \cup \to_{\mathcal{R},\beta\eta}$;
(R2) P is stable by $\to$;
(R3) if $t : T$ is neutral and $\to(t) \subseteq P$, then $t \in P$.

Lemma 16 *For all type T, $\mathbf{Red}^T_{\mathcal{R},\beta\eta}$ is stable by non-empty intersection and admits SN^T as greatest element. Moreover, for all $T, U \in \mathcal{T}$, $P \in \mathbf{Red}^T_{\mathcal{R},\beta\eta}$ and $Q \in \mathbf{Red}^U_{\mathcal{R},\beta\eta}$, $\propto(P, Q) \in \mathbf{Red}^{T\Rightarrow U}_{\mathcal{R},\beta\eta}$.*

PROOF. The proof is similar to the one of Lemma 1. One can easily check the stability by non-empty intersection and the fact that $\mathrm{SN}^T \in \mathbf{Red}^T_{\mathcal{R},\beta\eta}$. For the stability by $\propto$, there is no change for (R1) and (R2). We now detail (R3). Let $T, U \in \mathcal{T}$, $P \in \mathbf{Red}^T_{\mathcal{R},\beta\eta}$, $Q \in \mathbf{Red}^U_{\mathcal{R},\beta\eta}$, $v : T \Rightarrow U$ neutral such that $\to(v) \subseteq \propto(P, Q)$ and $t \in P$. We now show that $vt \in Q$ by well-founded induction on t with $\to$ as well-founded relation ($t \in \mathrm{SN}$ by (R1)). Since vt is neutral, by (R3), it suffices to prove that every reduct w of vt is in Q:

- $w = v't$ with $v \to v'$. By assumption, $v' \in \propto(P, Q)$. Therefore, $w \in Q$.
- $w = vt'$ with $t \to t'$. By the induction hypothesis, $w \in Q$.
- There are $\mathsf{f}\vec{l} \to r \in \mathcal{R}$ and σ such that $vt =_\eta \widehat{\sigma}(\mathsf{f}\vec{l}) = \mathsf{f}\widehat{\sigma}(\vec{l})$ and $w = r\sigma$. By confluence of $\to_\eta$, $(vt){\downarrow_\eta}$ is of the form $\mathsf{f}\vec{m}$ with $|\vec{m}| = |\vec{l}|$. Since v is neutral, v is of the form $x\vec{t}$, $(\lambda xa)b\vec{t}$ or $\mathsf{g}\vec{m}$ with $\alpha_\mathsf{g} \leq |\vec{m}|$. We discuss these cases in turn:

[38] See Definition 15.

– $v = x\vec{t}$. Then, $v{\downarrow_\eta}$ is of the form $x\vec{u}$. So, this case is not possible.

– $v = (\lambda x a)b\vec{t}$. Then, $a =_\eta cx$ with $x \notin \mathrm{FV}(c)$ and $cb\vec{t}u =_\eta \widehat{\sigma}(\mathsf{f}\vec{l})$. Hence, $v \to_\beta v' = a^b_x\vec{t}$, $v't =_\eta (cx)^b_x\vec{t}u = cb\vec{t}u =_\eta \widehat{\sigma}(\mathsf{f}\vec{l})$ and $v't \to w$. Therefore, $w \in Q$ since $v' \in \propto(P, Q)$, $t \in P$ and Q satisfies (R2).

– $v = \mathsf{g}\vec{m}$ with $\alpha_\mathsf{g} \leq |\vec{m}|$. Then, $\mathsf{g} = \mathsf{f}$ and $|\vec{m}| < |\vec{m}u| = |\vec{l}|$. Since v is neutral, $\alpha_\mathsf{f} \leq |\vec{m}|$. By definition of α_f, $|\vec{l}| \leq \alpha_\mathsf{f}$. So, this case is not possible. ■

We now check that Lemma 2 still holds if the following condition is satisfied:

Definition 28. A set $\mathcal{R}$ of rules is *β-complete* if, for all rules $l \to r \in \mathcal{R}$ and types T, U such that $l : T \Rightarrow U$, there is $x \in \mathcal{X} - \mathrm{FV}(l)$ such that $\tau(x) = T$ and:

- $lx \to s^x_y \in \mathcal{R}$ if $r = \lambda y s$;[39]
- $lx \to rx \in \mathcal{R}$ otherwise.

For instance, $\mathcal{R}$ is β-complete if, for every rule $l \to r \in \mathcal{R}$, l is of base type. On the other hand, the set $\mathcal{R} = \{\mathsf{f} \to \lambda x x\}$ is not β-complete since $\mathsf{f}x \to x \notin \mathcal{R}$.

Lemma 17 *Assume that $\mathcal{R}$ is β-complete. Given $T \in \mathcal{T}$ and $P \in \mathbf{Red}^T_{\mathcal{R},\beta\eta}$, $(\lambda x t)u\vec{v} \in P$ iff $(\lambda x t)u\vec{v} : T$, $t^u_x\vec{v} \in P$ and $u \in \mathrm{SN}$.*

Proof. Assume that $(\lambda x t)u\vec{v} \in P$. By (R2), $t^u_x\vec{v} \in P$. By (R1), $(\lambda x t)u\vec{v} \in \mathrm{SN}$. Therefore, $u \in \mathrm{SN}$.

Assume now that $t^u_x\vec{v} \in P$ and $u \in \mathrm{SN}$. By (R1), $t^u_x\vec{v} \in \mathrm{SN}$. Therefore, $\vec{v} \in \mathrm{SN}$, $t^u_x \in \mathrm{SN}$ and $t \in \mathrm{SN}$. We now prove that, for all $t, u, \vec{v} \in \mathrm{SN}$, $(\lambda x t)u\vec{v} \in P$, by induction on $\to_{\mathrm{prod}}$. Since $(\lambda x t)u\vec{v}$ is neutral, by (R3), it suffices to prove that every reduct w of $(\lambda x t)u\vec{v}$ belongs to P. Since rules are of the form $\mathsf{f}\vec{l} \to r$, there are three possible cases:

- $w = t^u_x\vec{v}$. Then, $w \in P$ by assumption.
- $w = (\lambda x t')u'\vec{v}'$ and $tu\vec{v} \to_{\mathrm{prod}} t'u'\vec{v}'$. Then, $w \in P$ by the induction hypothesis.
- There are $\mathsf{f}\vec{l} \to r \in \mathcal{R}$ and σ such that $\lambda x t =_\eta \widehat{\sigma}(\mathsf{f}\vec{l})$ and $w = r\sigma u\vec{v}$. By confluence of $\to_\eta$, there is a such that $t \to^*_\eta ax$, $x \notin \mathrm{FV}(a)$ and $a =_\eta \widehat{\sigma}(\mathsf{f}\vec{l})$. Wlog we can assume that $x \notin \mathrm{FV}(l)$ and σ is away from $\{x\}$. Hence, $t =_\eta \widehat{\sigma}(\mathsf{f}\vec{l}x)$. Since $\mathcal{R}$ is β-complete, there are two cases:
  - $r = \lambda y s$ and $\mathsf{f}\vec{l}x \to s^x_y \in \mathcal{R}$. Then, $t \to_{\mathcal{R},\beta\eta} s^x_y\sigma$. By monotony and stability by substitution, $t^u_x\vec{v} \to_{\mathcal{R},\beta\eta} (s^x_y\sigma)^u_x\vec{v}$. Hence, $(s^x_y\sigma)^u_x\vec{v} \in P$ by (R2). Therefore, by the induction hypothesis, $(\lambda x s^x_y\sigma)u\vec{v} \in P$. Wlog we can assume that σ is away from $\{y\}$. Hence, $(\lambda x s^x_y\sigma)u\vec{v} =_\alpha r\sigma u\vec{v}$.
  - r is not an abstraction and $\mathsf{f}\vec{l}x \to rx \in \mathcal{R}$. Then, $t \to_{\mathcal{R},\beta\eta} (rx)\sigma = r\sigma x$. By monotony and stability by substitution, $t^u_x\vec{v} \to_{\mathcal{R},\beta\eta} r\sigma u\vec{v}$. Hence, $r\sigma u\vec{v} \in P$ by (R2). ■

Corollary 4 *Assume that $\mathcal{R}$ is β-complete. Given $T, U \in \mathcal{T}$, $P \in \mathbf{Red}^T_{\mathcal{R},\beta\eta}$ and $Q \in \mathbf{Red}^U_{\mathcal{R},\beta\eta}$, $\lambda x t \in \propto(Q, P)$ iff $\lambda x t : U \Rightarrow T$ and, for all $u \in Q$, $t^u_x \in P$.*

Proof. Assume that $\lambda x t \in \propto(Q, P)$ and $u \in Q$. Then, by definition of $\propto$, $(\lambda x t)u \in P$. Therefore, by (R2), $t^u_x \in P$. Assume now that, for all $u \in P$, $t^u_x \in P$. By definition of $\propto$, $\lambda x t \in \propto(Q, P)$ if, for all $u \in Q$, $(\lambda x t)u \in P$. By (R1), $u \in \mathrm{SN}$. Therefore, by Lemma 17, $(\lambda x t)u \in P$. ■

But β-completeness is not a real restriction from the point of view of termination since:

[39] This case is not necessary for Lemma 17 to hold but avoids adding rules whose right-hand sides are β-redexes.

Lemma 18 *For every (finite) set of rules $\mathcal{S}$, there is a (finite) β-complete set of rules $\mathcal{R} \supseteq \mathcal{S}$ such that $\mathsf{f}\vec{l} \to r \in \mathrm{CC}_{\mathsf{f}}(\vec{l})$ for every $\mathsf{f}\vec{l} \to r \in \mathcal{R}$ if $\mathsf{f}\vec{l} \to r \in \mathrm{CC}_{\mathsf{f}}(\vec{l})$ for every $\mathsf{f}\vec{l} \to r \in \mathcal{S}$.*

Proof. Let F_β be the function on the powerset of $\mathcal{T}^2$ such that, for all $\mathcal{R} \subseteq \mathcal{T}^2$, $F_\beta(\mathcal{R})$ is the smallest set such that $\mathcal{R} \subseteq F_\beta(\mathcal{R})$ and, for all $l \to r \in \mathcal{R}$ and T, U such that $l : T \Rightarrow U$, there is $x \in \mathcal{X} - \mathrm{FV}(l)$ such that $\tau(x) = T$, $lx \to s \in F_\beta(\mathcal{R})$ if $r = \lambda xs$, and $lx \to rx \in F_\beta(\mathcal{R})$ otherwise. Since F_β is extensive (*i.e.* $\mathcal{R} \subseteq F_\beta(\mathcal{R})$), by Hessenberg's fixpoint theorem [Hes09], F_β has a fixpoint $\mathcal{R}$ such that $\mathcal{S} \subseteq \mathcal{R}$. Since $\mathcal{R} = F_\beta(\mathcal{R})$, $\mathcal{R}$ is β-complete.

Now, if $\mathcal{S} = \{l_1 \to r_1, \ldots, l_n \to r_n\}$ and, for every $i \in [1, n]$, $l_i : \vec{T^i} \Rightarrow \mathsf{A}_i$ with $\mathsf{A}_i \in \mathcal{B}$, then $\mathrm{card}(\mathcal{R}) \leq n + \Sigma_{i=1}^n |\vec{T^i}|$.

Assume now that $\mathsf{f}\vec{l} \to r \in \mathrm{CC}_{\mathsf{f}}(\vec{l})$ for every $\mathsf{f}\vec{l} \to r \in \mathcal{S}$, and that there are $\mathsf{f}\vec{l} \to r \in \mathcal{R}$ and T, U such that $\mathsf{f}\vec{l} : T \Rightarrow U$. By assumption, $r \in \mathrm{CC}_{\mathsf{f}}(\vec{l})$. Let now $x \in \mathcal{X} - \mathrm{FV}(l)$. Wlog, we can assume that $x \notin \mathrm{BV}(r)$. Hence, $r \in \mathrm{CC}_{\mathsf{f}}(\vec{lx})$. By (arg), $x \in \mathrm{CC}_{\mathsf{f}}(\vec{lx})$. Therefore, by (app), $rx \in \mathrm{CC}_{\mathsf{f}}(\vec{lx})$. Now, if $r = \lambda ys$, then $s_y^x \in \mathrm{CC}_{\mathsf{f}}(\vec{lx})$ by (red). ■

Note moreover that $\mathcal{S} \subseteq \mathcal{R} \subseteq \to_{\mathcal{S}} \to_{\beta_0}^{=}$. Therefore, $\to_\beta \cup \to_{\mathcal{R},\beta\eta}$ and $\to_\beta \cup \to_{\mathcal{S},\beta\eta}$ have the same normal forms and, if $=_{\mathcal{R}\beta}$ (resp. $=_{\mathcal{S}\beta}$) is the smallest congruence containing $\to_\beta$ and $\mathcal{R}$ (resp. $\mathcal{S}$), then $=_{\mathcal{R}\beta}$ is equal to $=_{\mathcal{S}\beta}$.

6.2. Preservation of computability by η-equivalence

In this section, we prove that computability is preserved by η-equivalence if $\leftarrow_\eta \to_{\mathcal{R},\beta\eta} \subseteq \to_{\mathcal{R},\beta\eta} =_\eta$. Then, we give sufficient conditions for this commutation property to hold.

Lemma 19 *Let $>_\mathcal{T}$ be the smallest transitive relation on types containing $>_\mathcal{B}$ and such that $T \Rightarrow U >_\mathcal{T} T$ and $T \Rightarrow U >_\mathcal{T} U$. The relation $>_\mathcal{T}$ is well-founded.*

Proof. Wlog we can assume that the symbol $\Rightarrow$ is not a type constant. Then, let $\succ$ be the smallest transitive relation on $\mathcal{B} \cup \{\Rightarrow\}$ containing $>_\mathcal{B}$ and such that $\Rightarrow \succ \mathsf{A}$ for all $\mathsf{A} \in \mathcal{B}$. The relation $\succ$ is well-founded for $>_\mathcal{B}$ is well-founded. Hence, $>_\mathcal{T}$ is well-founded for it is included in the recursive path ordering (RPO) built over $\succ$ [Der79]. ■

Lemma 20 *Let $\mathcal{R}$ be a set of rules such that $\leftarrow_\eta \to_{\mathcal{R},\beta\eta} \subseteq \to_{\mathcal{R},\beta\eta} =_\eta$, and assume that types are interpreted as in Section 4.7. If $t : T$ is computable, $t =_\eta u$ and $u : T$, then u is computable.*

Proof. Note that, by Lemma 12, $t \to_\eta^* \leftarrow_\eta^* u$. Since t and u are well-typed and $\to_\eta$ preserves typing, all terms between t and u are of type T.

We then proceed by induction on (1) the type of t ordered with $>_\mathcal{T}$ (well-founded by Lemma 19), (2) the rank of t (see Definition 12) if t is of base type, (3) t ordered by $\to$ ($t \in \mathrm{SN}$ by (R1)), and (4) the number of $\leftrightarrow_\eta$-steps between t and u.

If $T = V \to T'$, then u is computable if, for all computable $v : V$, $uv : T'$ is computable. By monotony, $tv =_\eta uv$. Since $tv : T'$ and $T >_\mathcal{T} T'$, uv is computable by the induction hypothesis.

If $t = u$, then u is computable. Assume now that $t =_\eta t' \leftrightarrow_\eta u$. By the induction hypothesis, t' is computable. Therefore, we are left to prove the lemma when $=_\eta$ is replaced by $\leftrightarrow_\eta$.

Assume now that T is a type constant A. By Lemma 4, a term $a : \mathsf{A}$ is computable iff all its reducts are computable and, for all $\mathsf{f} \in \mathcal{M}(\mathcal{R})$, $i \in \mathrm{Acc}(\mathsf{f})$ and $\vec{a}$ such that $a = \mathsf{f}\vec{a}$, a_i is computable.

We first prove that, for all $\mathsf{f} \in \mathcal{M}(\mathcal{R})$, $i \in \mathrm{Acc}(\mathsf{f})$ and $\vec{u}$ such that $u = \mathsf{f}\vec{u}$, $u_i : \vec{V} \Rightarrow \mathsf{B}$ is computable. Since t is of base type and $t \leftrightarrow_\eta u = \mathsf{f}\vec{u}$, there are $\vec{t}$ such that $t = \mathsf{f}\vec{t}$ and $\vec{t}\ (\leftrightarrow_\eta)_{\mathrm{prod}}\ \vec{u}$. Now, u_i is computable if, for all computable $\vec{v} : \vec{V}$, $u_i\vec{v}$ is computable. By monotony, $t_i\vec{v} \leftrightarrow_\eta^= u_i\vec{v}$ and $t_i\vec{v}$ has a type or a rank smaller than the type or rank of $\mathsf{f}\vec{t}$ (for $i \in \mathrm{Acc}(\mathsf{f})$). Therefore, $u_i\vec{v}$ is computable by the induction hypothesis.

We now prove that all the reducts v of u are computable.

- $t \xrightarrow{p}_\eta u \xrightarrow{q}_\beta v$.[40] We now prove that there is t' such that $t \to_\beta^+ t' \to_\eta^* v$, so that we can conclude by the induction hypothesis:
  - $p \# q$. In this case, $t \to_\beta \to_\eta v$.
  - $p \leq q$. There are a and a' such that $t|_p = \lambda xax$, $x \notin \mathrm{FV}(a)$, $u = t[a]_p$, $a \to_\beta a'$ and $v = t[a']_p$. Thus, $t \to_\beta t[\lambda xa'x]_p \to_\eta v$.
  - $p > q$. There are a and b such that $u|_q = (\lambda xa)b$ and $v = u[a^b_x]_q$.
    - $p \geq q1$. There is d such that $t = u[(\lambda xa)d]_q$ and $d \to_\eta b$. Thus, $t \to_\beta u[a^d_x] \to_\eta^* v$.
    - $p = q0$. Then, $t = u[(\lambda x(\lambda xa)x)b]_q$. Thus, $t \to_{\beta_0} u \to_\beta v$.
    - $p > q0$. There is d such that $t = u[(\lambda xd)b]_q$ and $d \to_\eta a$. Thus, $t \to_\beta u[d^b_x] \to_\eta v$.
- $t \xrightarrow{p}_\eta u \xrightarrow{q}_{\mathcal{R},\beta\eta} v$. We now prove that there is t' such that $t \to_{\mathcal{R},\beta\eta} t' \to_\eta^= v$, so that we can conclude by the induction hypothesis.
  - $p \# q$. Then, $t \to_{\mathcal{R},\beta\eta} t' \to_\eta v$.
  - $p \geq q$. Then, $t \to_{\mathcal{R},\beta\eta} v$.
  - $p < q$. There are a and a' such that $t|_p = \lambda xax$, $x \notin \mathrm{FV}(a)$, $u = t[a]_p$, $a \to_{\mathcal{R},\beta\eta} a'$ and $v = t[a']_p$. Thus, $t \to_{\mathcal{R},\beta\eta} t[\lambda xa'x]_p \to_\eta v$.
- $t \xleftarrow{p}_\eta u \xrightarrow{q}_{\mathcal{R},\beta\eta} v$. By assumption, there is t' such that $t \to t' =_\eta v$, so that we can conclude by the induction hypothesis.
- $t \xleftarrow{p}_\eta u \xrightarrow{q}_\beta v$. We now prove that, either $v = t$ and v is computable for t is computable, or there is t' such that $t \to_\beta t' \leftarrow_\eta^* v$ and we can conclude by the induction hypothesis:
  - $p \# q$. Then, $t \to_\beta t' \leftarrow_\eta v$.
  - $p = q$. Not possible.
  - $p > q$. There are a and b such that $u|_q = (\lambda xa)b$ and $v = u[a^b_x]_q$.
    - $p = q0$. There is d such that $a = dx$, $x \notin \mathrm{FV}(d)$ and $t = u[db]_q$. Thus, $t = v$.
    - $p > q0$. There is a' such that $a \to_\eta a'$ and $t = u[(\lambda xa')b]_q$. Thus, $t \to_\beta u[a'^b_x]_q \leftarrow_\eta v$.
    - $p \geq q1$. There is b' such that $b \to_\eta b'$ and $t = u[(\lambda xa)b']_q$. Thus, $t \to_\beta u[a^{b'}_x]_q \leftarrow_\eta^* v$.
  - $p < q$. There is a such that $u|_p = \lambda xax$, $x \notin \mathrm{FV}(a)$ and $t = u[a]_p$.
    - $p0 = q$. There is b such that $a = \lambda yb$ and $v = u[\lambda xb^x_y]_p$. As already mentioned in Lemma 13, since u is well-typed, we can assume wlog that $y = x$. Thus, $t = v$.
    - $p0 < q$. There is a' such that $a \to_\beta a'$ and $v = u[\lambda xa'x]_p$. Thus, $t \to_\beta u[a']_p \leftarrow_\eta v$. ■

In the previous proof, we have seen that $\to_\eta \to_{\mathcal{R},\beta\eta} \subseteq \to_{\mathcal{R},\beta\eta} \to_\eta^=$. Hence, if we also have $\leftarrow_\eta \to_{\mathcal{R},\beta\eta} \subseteq \to_{\mathcal{R},\beta\eta} =_\eta$, then $\leftrightarrow_\eta \to_{\mathcal{R},\beta\eta} \subseteq \to^+_{\mathcal{R},\beta\eta} =_\eta$, a property that, after [JM84], we call:

Definition 29. A relation R *locally η-commutes* if $\leftrightarrow_\eta R \subseteq R^+ =_\eta$.

We now provide sufficient conditions for this property to hold:

Definition 30. A set $\mathcal{R}$ of rules is *η-complete* if, for all l, k, r, x such that $lk \to r \in \mathcal{R}$, $k \to_\eta^* x$ and $x \in \mathcal{X} - \mathrm{FV}(l)$, we have:

[40] This case could be simplified and dealt with by (R2) if $\to_\eta$ was included in $\to$. But, then, we would have to check Lemma 16 again. The present proof shows that this is not necessary.

- $l \to s \in \mathcal{R}$ if $r = sk'$, $k' \to_\eta^* x$ and $x \notin \mathrm{FV}(s)$;[41]
- $l \to \lambda xr \in \mathcal{R}$ otherwise.

Lemma 21 *If $\mathcal{R}$ is η-complete, then $\leftarrow_\eta \to_{\mathcal{R},\beta\eta} \subseteq \to_{\mathcal{R},\beta\eta} =_\eta$ and $\to_{\mathcal{R},\beta\eta}$ locally η-commutes.*

Proof. Assume that $t \overset{p}{\leftarrow}_\eta u \overset{q}{\to}_{\mathcal{R},\beta\eta} v$.

- $p \# q$. Then, $t \to_{\mathcal{R},\beta\eta} \leftarrow_\eta v$.
- $p \geq q$. Then, $t \to_{\mathcal{R},\beta\eta} v$.
- $p < q$. There is a such that $u|_p = \lambda xax$, $x \notin \mathrm{FV}(a)$ and $t = u[a]_p$.
  - $p01 \leq q$. Not possible since the rules are of the form $\mathsf{f}\vec{l} \to r$.
  - $p00 \leq q$. There is a' such that $v = u[\lambda xa'x]_p$ and $a \to_{\mathcal{R},\beta\eta} a'$. Then, $t \to_{\mathcal{R},\beta\eta} u[a']_p \leftarrow_\eta v$.
  - $p0 = q$. There are $\mathsf{f}\vec{l} \to r \in \mathcal{R}$ and σ such that $ax =_\eta \widehat{\sigma}(\mathsf{f}\vec{l})$ and $v = u[\lambda xr\sigma]_p$. By confluence of $\to_\eta$, there are $\vec{m}$ and k such that $\vec{l} = \vec{m}k$, $a =_\eta \widehat{\sigma}(\mathsf{f}\vec{m})$ and $x =_\eta \widehat{\sigma}(k)$. Since k is a pattern, there is $y \in \mathcal{X}$ such that $k \to_\eta^* y$ and $y\sigma \to_\eta^* x$. Wlog we can assume that $y = x$. Let θ be the restriction of σ on $\mathrm{FV}(\mathsf{f}\vec{m})$. Since $x \notin \mathrm{FV}(a)$ and the set of free variables of a term is invariant by $=_\eta$, we have $x \notin \mathrm{FV}(\vec{m})$ and θ away from $\{x\}$. Now, since $\mathcal{R}$ is η-complete, there are two cases:
    - $r = sk'$, $k' \to_\eta^* x$, $x \notin \mathrm{FV}(s)$ and $\mathsf{f}\vec{m} \to s \in \mathcal{R}$. Then, $a \to_{\mathcal{R},\beta\eta} s\theta$ and $\mathrm{FV}(s\theta) \subseteq \mathrm{FV}(a)$. Since $x \notin \mathrm{FV}(a)$, $x \notin \mathrm{FV}(s\theta)$ and $s\theta \leftarrow_\eta \lambda xs\theta x$. Since $x \leftarrow_\eta^* x\sigma$ and $x \leftarrow_\eta^* k'$, we have $x \leftarrow_\eta^* k'\sigma$. Therefore, $t \to_{\mathcal{R},\beta\eta} \leftarrow_\eta^* u[\lambda xs\theta k'\sigma] = v$.
    - Otherwise, $\mathsf{f}\vec{m} \to \lambda xr \in \mathcal{R}$. Hence, $a \to_{\mathcal{R},\beta\eta} (\lambda xr)\theta$. Since θ is away from $\{x\}$, $(\lambda xr)\theta = \lambda xr\theta$. Since $x = x\theta \leftarrow_\eta^* x\sigma$, $r\theta \leftarrow_\eta^* r\sigma$. Therefore, $t \to_{\mathcal{R},\beta\eta} \leftarrow_\eta^* v$. ■

For instance, $\mathcal{R} = \{\mathsf{f}x \to x\}$ is not η-complete since $\mathsf{f} \to \lambda xx \notin \mathcal{R}$ and, indeed, the relation $\leftarrow_\eta$ does not commute with $\to_{\mathcal{R},\beta\eta}$ because of the non-joinable critical pair $\mathsf{f} \leftarrow_\eta \lambda x\mathsf{f}x \to_{\mathcal{R}} \lambda xx$. Adding the rule $\mathsf{f} \to \lambda xx$ allows us to recover commutation.

But η-completeness is not a real restriction from the point of view of termination since:

Lemma 22 *For every (finite) set of rules $\mathcal{S}$, there is an η-complete (finite) set of rules $\mathcal{R} \supseteq \mathcal{S}$ such that, using the rules of Figure 8, $\mathsf{f}\vec{l} \to r \in \mathrm{CC}_\mathsf{f}(\vec{l})$ for every $\mathsf{f}\vec{l} \to r \in \mathcal{R}$ if $\mathsf{f}\vec{l} \to r \in \mathrm{CC}_\mathsf{f}(\vec{l})$ for every $\mathsf{f}\vec{l} \to r \in \mathcal{S}$.*

Proof. Let F_η be the function on the powerset of $\mathcal{T}^2$ such that, for all $\mathcal{R} \subseteq \mathcal{T}^2$, $F_\eta(\mathcal{R})$ is the smallest set such that $\mathcal{R} \subseteq F_\eta(\mathcal{R})$ and, for all l, k, r, x such that $lk \to r \in \mathcal{R}$, $k \to_\eta^* x$ and $x \in \mathcal{X} - \mathrm{FV}(l)$, $l \to s \in F_\eta(\mathcal{R})$ if $r = sk'$, $k' \to_\eta^* x$ and $x \notin \mathrm{FV}(s)$, and $l \to \lambda xr \in F_\eta(\mathcal{R})$ otherwise.

Since F_η is extensive (*i.e.* $\mathcal{R} \subseteq F_\eta(\mathcal{R})$), by Hessenberg's fixpoint theorem [Hes09], F_η has a fixpoint $\mathcal{R}$ such that $\mathcal{S} \subseteq \mathcal{R}$. Since $\mathcal{R} = F_\eta(\mathcal{R})$, $\mathcal{R}$ is η-complete.

If $\mathcal{S} = \{l_1 \to r_1, \ldots, l_n \to r_n\}$ and, for every $i \in [1, n]$, $l_i : \vec{T^i} \Rightarrow \mathsf{A}_i$ with $\mathsf{A}_i \in \mathcal{B}$, then $\mathrm{card}(\mathcal{R}) \leq n + \Sigma_{i=1}^n |\vec{T^i}|$.

Assume now that $\mathsf{f}\vec{l} \to r \in \mathrm{CC}_\mathsf{f}(\vec{l})$ for every $\mathsf{f}\vec{l} \to r \in \mathcal{S}$, and that there are $\mathsf{f}\vec{l}k \to r \in \mathcal{R}$ and $x \in \mathcal{X} - \mathrm{FV}(\vec{l})$ such that $k \to_\eta^* x$. By assumption, $r \in \mathrm{CC}_\mathsf{f}(\vec{l}k)$. By (var), $x \in \mathrm{CC}_\mathsf{f}(\vec{l})$. Therefore, by (eta), $k \in \mathrm{CC}_\mathsf{f}(\vec{l})$. Now, since $x \in \mathcal{X} - \mathrm{FV}(\vec{l})$, we can get $r \in \mathrm{CC}_\mathsf{f}(\vec{l})$ by replacing, everywhere in the derivation proof of $r \in \mathrm{CC}_\mathsf{f}(\vec{l}k)$, $\mathrm{CC}_\mathsf{f}(\vec{l}k)$ by $\mathrm{CC}_\mathsf{f}(\vec{l})$, and the proofs of $k \in \mathrm{CC}_\mathsf{f}(\vec{l}k)$ obtained with (arg), by the proof of $k \in \mathrm{CC}_\mathsf{f}(\vec{l})$ obtained with (var) and (eta). Therefore, by (abs), $\lambda xr \in \mathrm{CC}_\mathsf{f}(\vec{l})$. Now, if $r = sk'$ with $k' \to_\eta^* x$ and $x \notin \mathrm{FV}(s)$, then $s \in \mathrm{CC}_\mathsf{f}(\vec{l})$ by (eta). ■

[41] This case is not necessary for Lemma 20 to hold but avoids adding rules whose right-hand sides are η-redexes.

The fact that the rules of Figure 8 are valid computability closure operations is proved in [Bla00].

Note that $\mathcal{S} \subseteq \mathcal{R} \subseteq \leftarrow_\eta^* \mathcal{S} \rightarrow_\eta^*$. Hence, if $=_{\mathcal{R}\beta\eta}$ (resp. $=_{\mathcal{S}\beta\eta}$) is the smallest congruence containing $\rightarrow_\eta$, $\rightarrow_\beta$ and $\mathcal{R}$ (resp. $\mathcal{S}$), then $=_{\mathcal{R}\beta\eta}$ is equal to $=_{\mathcal{S}\beta\eta}$. Moreover, $\rightarrow_{\mathcal{R},\beta\eta}$ and $\rightarrow_{\mathcal{S},\beta\eta}$ have the same normal forms on η-long terms.

We have seen that termination of rewriting with matching modulo $\beta\eta$ relies on commutation properties between $\rightarrow_{\mathcal{R},\beta\eta}$ and $\leftrightarrow_{\beta\eta}$. Such conditions are well-known in first-order rewriting theory: the notion of compatibility of Peterson and Stickel [PS81], the notion of local E-commutation of Jouannaud and Muñoz [JM84] and, more generally, the notion of local coherence modulo E of Jouannaud and Kirchner [JK86]. Similarly, the addition of *extension rules* to make a system compatible, locally commute or locally coherent is also well-known since Lankford and Ballantyne [LB77].

6.3. Preservation of computability by leaf-β-expansion

We now prove that computability is preserved by leaf-β-expansion, but for patterns containing undefined symbols *only*.

Definition 31. Let v be a term, $p \in \mathrm{LPos}(v)$ and $\vec{q}$ be the leaf positions of v distinct from p. We say that a term t is *valid* wrt (v,p) if there are $\vec{t}$ and u such that $t = v[\vec{t}]_{\vec{p}}[u]_p$ and, for all $\vec{y}$, a and $\vec{b}$ such that $u = (\lambda\vec{y}a)\vec{b}$ and $|\vec{y}| = |\vec{b}|$, we have $\vec{b} \in [\![\tau(\vec{y})]\!]$ and, for all $j \in [1, |\vec{b}|]$, either $b_j{\downarrow_\eta} \in \mathrm{BV}(l,p)$ or $\mathrm{FV}(b_j) \cap \mathrm{BV}(l,p) = \emptyset$.

Note that, if l is a pattern and σ is valid wrt l, then every term t such that $\widehat{\sigma}(l) \leftarrow^*_{\beta,l,p_1} \ldots \leftarrow^*_{\beta,l,p_k} t$, where $p_1, \ldots, p_k$ are leaf positions of l, is valid wrt $(l,p_1), \ldots, (l,p_k)$ (for $b_j{\downarrow_\eta} \in \mathrm{BV}(l,p_i)$ in this case).

Lemma 23 *Let $\mathcal{R}$ be a β and η-complete set of rules, and assume that types are interpreted as in Section 4.7. Let l be a term containing undefined symbols only, and let p be a leaf position of l. If $t \in [\![\tau(l)]\!]$, $t \leftarrow_{\beta,l,p} u$ and u is valid wrt (l,p), then $u \in [\![\tau(l)]\!]$.*

Proof. Let $S = [\![\tau(l)]\!]$. Note that l does not need to be a pattern, a property that cannot be preserved when instantiating bound variables. In fact, the complete structure of l is not relevant. Because we look at leaf-β-expansions, only the top part of l that is above the leaf positions is relevant. Hence, let $||\_||$ be the measure on terms defined as follows:

- $||l|| = 1 + ||m||$ if $l = \lambda z m$,
- $||l|| = 1 + \mathsf{sup}\{||l_1||, \ldots, ||l_n||\}$ if $l = \mathsf{f} l_1 \ldots l_n$ and $n \geq 1$,
- $||l|| = 0$ otherwise.

We prove the lemma by induction on (1) $||l||$, (2) $\tau(l)$, (3) t ordered by $\rightarrow$ (for $t \in \mathrm{SN}$ by (R1)), and (4) the terms $\vec{b}$ such that $u|_p = (\lambda\vec{y}a)\vec{b}$ (for u is valid wrt (l,p)) ordered by $\rightarrow$ (for $\vec{b} \in \mathrm{SN}$ by (R1)). We proceed by case on l:

- $||l|| = 0$. Then, there are $a, x, e, \vec{b}$ such that $t = a^e_x\vec{b}$ and $u = (\lambda x a)e\vec{b}$. Since u is valid, $e \in [\![\tau(x)]\!]$. Hence, $e \in \mathrm{SN}$ by (R1). Therefore, by Lemma 17, $u \in S$.
- $l = \lambda z m$. Then, there are r, s, q and M such that $l : \tau(z) \Rightarrow M$, $t = \lambda z r$, $u = \lambda z s$ and $r \leftarrow_{\beta,m,q} s$. That is, $S = \propto([\![\tau(z)]\!], [\![M]\!])$ and there are $\vec{t}$, a, x, e, $\vec{b}$ such that $r = m[\vec{t}]_{\vec{k}}[a^e_x\vec{b}]_q$ and $s = m[\vec{t}]_{\vec{k}}[(\lambda x a)e\vec{b}]_q$, where $\vec{k}$ are all the leaf positions of m distinct from q. By Corollary 4, $u \in S$ if, for all $g \in [\![\tau(z)]\!]$, $s^g_z = m[\vec{t^g_z}]_{\vec{k}}[(\lambda x a)^g_z e^g_z \vec{b^g_z}]_q \in [\![M]\!]$. So, let $g \in [\![\tau(z)]\!]$. By Corollary 4, $r^g_z = m[\vec{t^g_z}]_{\vec{k}}[(a^e_x)^g_z \vec{b^g_z}]_q \in [\![M]\!]$. Let $b_0 = e$. Since u is valid and $z \in \mathrm{BV}(l,p)$, for all $i \in [0, |\vec{b}|]$, either $b_i \rightarrow^*_\eta z$ and $b_i(^g_z) \rightarrow^*_\eta g$, or $z \notin \mathrm{FV}(b_i)$ and $b_i(^g_z) = b_i$. Therefore, s^g_z is valid. Wlog we can assume that $x \neq z$ and $x \notin \mathrm{FV}(g)$. Hence, $(\lambda x a)^g_z = \lambda x a^g_z$ and $(a^e_x)^g_z = (a^g_z)^{e^g_z}_x$. Therefore, $r^g_z \leftarrow_{\beta,m,q} s^g_z$ and, by the induction hypothesis (1), $s^g_z \in [\![M]\!]$.
- $l = \mathsf{f}\vec{l}$ with $\tau(\mathsf{f}) = \vec{T} \Rightarrow U$. We proceed by case on $\tau(l)$:

– $\tau(l) = V \Rightarrow W$. By definition of computability, $u \in S$ if, for all $v \in [\![V]\!]$, $uv \in [\![W]\!]$. So, let $v \in [\![V]\!]$. Then, $tv \in [\![W]\!]$ and $tv \leftarrow_{\beta,lx,0p} uv$. Moreover, uv is valid wrt $(lx, 0p)$ and $||lx|| = ||l||$. Therefore, by the induction hypothesis (2), $uv \in [\![W]\!]$.

– $\tau(l) \in \mathcal{B}$. Since $t \leftarrow_{\beta,l,p} u$, there are $i, q, \vec{t}, \vec{u}$ such that $p = 0^{|\vec{l}|-i}1q$, $t = \mathsf{f}\vec{t}$, $u = \mathsf{f}\vec{u}$ and $t_i \leftarrow_{\beta,l_i,q} u_i$, that is, there are $a, x, e, \vec{b}$ such that $t_i|_q = a^e_x\vec{b}$ and $u_i|_q = (\lambda xa)e\vec{b}$.
By Lemma 4, $u \in S$ if all its reducts are in S and, if $\mathsf{f} \in \mathcal{M}(\mathcal{R})$ and $i \in \mathrm{Acc}(\mathsf{f})$, then $u_i \in [\![T_i]\!]$. Assume that $\mathsf{f} \in \mathcal{M}(\mathcal{R})$ and $i \in \mathrm{Acc}(\mathsf{f})$. By Lemma 4, $t_i \in [\![T_i]\!]$. If $t_i = u_i$ then $u_i \in [\![T_i]\!]$. Otherwise, $t_i \leftarrow_{\beta,l_i,q} u_i$. Therefore, since u_i is valid wrt (l_i, q), by the induction hypothesis (1), $u_i \in [\![T_i]\!]$. We now prove that, if $u \xrightarrow{q} v$, then $v \in S$:

  * $p\#q$. Then, $t \to t' \leftarrow_{\beta,l,p} v$. By (R2), $t' \in S$. Since t' is valid wrt (l, p), by the induction hypothesis (3), $v \in S$.
  * $p > q$. Not possible since l contains undefined symbols only.
  * There are $\mathsf{f}\vec{l} \to r \in \mathcal{R}$ and θ such that $\tau(\lambda xa) = \tau(\mathsf{f}\vec{l})$, $\lambda xa =_\eta \widehat{\theta}(\mathsf{f}\vec{l})$ and $v = r\theta e\vec{b}$. Wlog we can assume that $x \notin \mathrm{FV}(\mathsf{f}\vec{l})$ and θ is away from $\{x\}$. Then, as already seen in the proof of Lemma 17, $a =_\eta \widehat{\theta}(\mathsf{f}\vec{l}x)$. Since $\mathcal{R}$ is β-complete, there are two cases:
    · There is s such that $r = \lambda xs$. Then, $\mathsf{f}\vec{l}x \to s \in \mathcal{R}$ and $a \to_{\mathcal{R},\beta\eta} s\theta$. Hence, $t \to t' = u[(s\theta)^e_x\vec{b}]_p \leftarrow_{\beta,l,p} u[(\lambda xs\theta)e\vec{b}]_p = v$. By (R2), $t' \in S$. Since v is valid wrt (l, p), by the induction hypothesis (3), $v \in S$.
    · Otherwise, $\mathsf{f}\vec{l}x \to rx \in \mathcal{R}$ and $a \to_{\mathcal{R},\beta\eta} r\theta x$. Hence, $t \to t' = u[(r\theta x)^e_x\vec{b}]_p \leftarrow_{\beta,l,p} u' = u[(\lambda xr\theta x)e\vec{b}]_p \to_\eta v$. By (R2), $t' \in S$. Since u' is valid wrt (l, p), by the induction hypothesis (3), $u' \in S$. Therefore, by Lemma 20, $v \in S$.
  * There is a' such that $a \to a'$ and $v = u[(\lambda xa')e\vec{b}]_p$. Then, $t \to t' = u[a'^e_x\vec{b}]_p \leftarrow_{\beta,l,p} v$. By (R2), $t' \in S$. Since v is valid wrt (l, p), by the induction hypothesis (3), $v \in S$.
  * There is e' such that $e \to e'$ and $v = u[(\lambda xa)e'\vec{b}]_p$. Then, $t \to^* t' = u[a^{e'}_x\vec{b}]_p \leftarrow_{\beta,l,p} v$. By (R2), $t' \in S$. Since u is valid wrt (l, p), $e \in [\![\tau(x)]\!]$. By (R2), $e' \in [\![\tau(x)]\!]$. Therefore, v is valid wrt (l, p) and, by the induction hypothesis (4), $v \in S$.
  * There is $\vec{b}'$ such that $\vec{b} \to_{\mathrm{prod}} \vec{b}'$ and $v = u[(\lambda xa)e\vec{b}']_p$. Then, $t \to t' = u[a^e_x\vec{b}']_p \leftarrow_{\beta,l,p} v$. Since u is valid wrt (l, p), $\vec{b}$ are computable. Thus, by (R2), $\vec{b}'$ are computable and v is valid. Therefore, by the induction hypothesis (3), v is computable. ■

Finally, we check that β and η-completion commute when left and right-hand sides are $\beta\eta$-normal. Hence, any (finite) set of rules $\mathcal{S}$ whose left-hand and right-hand sides are $\beta\eta$-normal can be completed into a (finite) β and η-complete set of rules $\mathcal{R} \supseteq \mathcal{S}$.

Lemma 24 *β-completion (resp. η-completion) preserves η-completeness (resp. β-completeness when left-hand and right-hand sides are $\beta\eta$-normal).*

Proof. • We will say that $\mathcal{R}$ is $\beta\eta$-normal if, for every rule $l \to r \in \mathcal{R}$, both l and r are $\beta\eta$-normal. We first prove that the function F_η defined in the proof of Lemma 22 preserves β-completeness and $\beta\eta$-normality. Let $\mathcal{R}$ be a $\beta\eta$-normal and β-complete set of rules. We have to prove that $F_\eta(\mathcal{R})$ is $\beta\eta$-normal and β-complete, that is, if there are $l \to r \in F_\eta(\mathcal{R})$ and $T, U \in \mathcal{T}$ such that $l : T \Rightarrow U$, then there is $x \in \mathcal{X} - \mathrm{FV}(l)$ such that $\tau(x) = T$ and, either $r = \lambda ys$ and $lx \to s^x_y \in F_\eta(\mathcal{R})$, or $lx \to rx \in F_\eta(\mathcal{R})$. Let $l \to r \in F_\eta(\mathcal{R}) - \mathcal{R}$ and assume that there is $gk \to d \in \mathcal{R}$ such that $k \to^*_\eta x \in \mathcal{X} - \mathrm{FV}(g)$. Then, either:

– $d = sk'$, $k' \to^*_\eta x \in \mathcal{X} - \mathrm{FV}(s)$ and $l \to r = g \to s$. Since $\mathcal{R}$ is $\beta\eta$-normal, $k = k' = x$ and r is not an abstraction. Therefore, $lx \to rx \in F_\eta(\mathcal{R})$ since $lx = gk$, $rx = sk' = d$ and $gk \to d \in \mathcal{R}$. Moreover, l is $\beta\eta$-normal since $l = g$ and gk is $\beta\eta$-normal, and r is $\beta\eta$-normal since $r = s$ and $d = sk'$ is $\beta\eta$-normal.

– $l \to r = g \to \lambda xd$. Since $\mathcal{R}$ is $\beta\eta$-normal, $k = x$. Therefore, $lx \to d \in F_\eta(\mathcal{R})$ since $lx = gk$ and $gk \to d \in \mathcal{R}$. Moreover, l is $\beta\eta$-normal since $l = g$ and g is $\beta\eta$-normal, and r is $\beta\eta$-normal since $r = \lambda xd$, d is $\beta\eta$-normal and d is not of the form sk' with $k' \to^*_\eta x \in \mathcal{X} - \mathrm{FV}(s)$.

- We now prove that the function F_β defined in the proof of Lemma 18 preserves η-completeness. Let $\mathcal{R}$ be an η-complete set of rules. We have to prove that $F_\beta(\mathcal{R})$ is η-complete, that is, if $lk \to r \in F_\beta(\mathcal{R})$ and $k \to_\eta^* x \in \mathcal{X} - \mathrm{FV}(l)$ then, either $r = tk'$, $k' \to_\eta^* x \in \mathcal{X} - \mathrm{FV}(t)$ and $l \to t \in F_\beta(\mathcal{R})$, or $l \to \lambda x r \in F_\beta(\mathcal{R})$. Let $l \to r \in F_\beta(\mathcal{R}) - \mathcal{R}$ and assume that there are $g \to d \in \mathcal{R}$ and $T, U \in \mathcal{T}$ such that $g : T \Rightarrow U$. Then, there is $x \in \mathcal{X} - \mathrm{FV}(g)$ such that $x : T$ and either:
  - $d = \lambda y s$ and $lk \to r = gx \to s_y^x$. Wlog we can assume that $y = x$. If $r = tk'$ and $k' \to_\eta^* x \in \mathcal{X} - \mathrm{FV}(t)$, then d is not $\beta\eta$-normal. Therefore, $l \to \lambda x r \in F_\beta(\mathcal{R})$ since $l = g$, $r = s$ and $g \to \lambda x s \in \mathcal{R}$.
  - $lk \to r = gx \to dx$. Therefore, $l \to d \in F_\beta(\mathcal{R})$ since $l = g$ and $g \to d \in \mathcal{R}$. ■

6.4. Handling the subterms of a pattern

We now show that Theorem 5 extends to rewriting with pattern matching modulo $\beta\eta$:

Figure 8: Computability closure operations VI

(subterm-abs)	if $\lambda x t \in \mathrm{CC}_\mathsf{f}(\vec{l})$ and $x \in \mathcal{X} - \mathrm{FV}(\vec{l})$, then $t \in \mathrm{CC}_\mathsf{f}(\vec{l})$
(subterm-app)	if $tx \in \mathrm{CC}_\mathsf{f}(\vec{l})$ and $x \in \mathcal{X} - (\mathrm{FV}(t) \cup \mathrm{FV}(\vec{l}))$, then $t \in \mathrm{CC}_\mathsf{f}(\vec{l})$
(eta)	if $t \in \mathrm{CC}_\mathsf{f}(\vec{l})$, $t =_\eta u$ and $\tau(t) = \tau(u)$, then $u \in \mathrm{CC}_\mathsf{f}(\vec{l})$

Theorem 8 *Given a set of rules $\mathcal{R}$ that is both β and η-complete, the relation $\to_\beta \cup \to_{\mathcal{R},\beta\eta}$ terminates on well-typed terms if there is an $\mathcal{F}$-quasi-ordering $\geq$ valid wrt the interpretation of Section 4.7 such that, for every rule $\mathsf{f}\vec{l} \to r \in \mathcal{R}$, $\vec{l}$ are patterns containing undefined symbols only and $r \in \mathrm{CC}_\mathsf{f}(\vec{l})$, where* CC *is the smallest computability closure closed by the operations I to VI.*

Proof. We proceed as for Theorem 5 by showing that, for all $(\mathsf{f}, \vec{t}) \in \Sigma_{\mathsf{max}}$, every reduct t of $\mathsf{f}\vec{t}$ is computable, by well-founded induction on $> \cup \to_{\mathrm{prod}}$. There are two cases:

- There is $\vec{u}$ such that $t = \mathsf{f}\vec{u}$ and $\vec{t} \to_{\mathrm{prod}} \vec{u}$. By (R2), $\vec{u}$ is computable. Therefore, by the induction hypothesis, $\mathsf{f}\vec{u}$ is computable.
- There are $\vec{s}$, $\vec{w}$, $\mathsf{f}\vec{l} \to r \in \mathcal{R}$ and σ such that $\vec{t} = \vec{s}\vec{w}$, $\vec{s} =_\eta \widehat{\sigma}(\vec{l})$ and $t = r\sigma\vec{w}$. By Lemma 20, $\widehat{\sigma}(\vec{l})$ are computable. Let now $i \in [1, |\vec{l}|]$ and $p_1, \ldots, p_n$ be the leaf positions of l_i. By Lemma 14, we have $\widehat{\sigma}(l_i) \leftarrow_{\beta,l,p_1}^* \ldots \leftarrow_{\beta,l,p_n}^* l_i\sigma$. Since all the terms between $\widehat{\sigma}(l_i)$ and $l_i\sigma$ are valid, by Lemma 23, $l_i\sigma$ is computable. Since $r \in \mathrm{CC}_\mathsf{f}(\vec{l})$ and CC is stable by substitution (for $>$ is stable by substitution), we have $r\sigma \in \mathrm{CC}_\mathsf{f}(\vec{l\sigma})$. Now, Lemma 3 is easily extended with the rules of Figure 8 for destructuring patterns [Bla00]. Therefore, $r\sigma$ is computable since, for all $(\mathsf{g}, \vec{u}) \in \Sigma_{\mathsf{max}}$, if $(\mathsf{f}, \vec{t}) > (\mathsf{g}, \vec{u})$, then $\mathsf{g}\vec{u}$ is computable by the induction hypothesis. ■

For instance, let us check that these conditions are satisfied by the formal derivation rule given at the beginning of the section. Let $l = \lambda x\, \mathsf{sin}(Fx)$ and assume that $\mathsf{D} >_\mathcal{F} \times$. By (arg), $l \in \mathrm{CC} = \mathrm{CC}_\mathsf{D}(l)$. By (var), $x \in \mathrm{CC}$. By (subterm-abs), $\mathsf{sin}(Fx) \in \mathrm{CC}$. By (subterm-acc), $Fx \in \mathrm{CC}$. By (subterm-app), $F \in \mathrm{CC}$. By (undef), $\mathsf{cos}(Fx) \in \mathrm{CC}$. By (rec), $\mathsf{D}Fx \in \mathrm{CC}$ for $l \rhd_\mathsf{s} F$. By (rec), $(\mathsf{D}Fx) \times (\mathsf{cos}(Fx)) \in \mathrm{CC}$ for $\mathsf{D} >_\mathcal{F} \times$. Therefore, by (abs), $\lambda x(\mathsf{D}Fx) \times (\mathsf{cos}(Fx)) \in \mathrm{CC}$.

6.5. Application to CRSs and HRSs

CRSs [Klo80, KvOvR93] can be seen as an extension of the untyped λ-calculus with no object-level application symbol but, instead, symbols of fixed arity defined by rules using a matching mechanism equivalent to matching modulo $\beta\eta$ on Miller patterns.

In HRSs [Nip91, MN98], one considers simply-typed λ-terms in β-normal η-long form with symbols defined by rules using Miller's pattern-matching mechanism.

Note that, although HRS terms are simply typed, one can easily encode the untyped λ-calculus in it by considering an object-level application symbol. Similarly, in CRSs, one easily recovers the untyped λ-calculus by considering an object-level application symbol. Such a CRS is called β-CRS in [Bla00].

In HALs [JO91], Jouannaud and Okada consider arbitrary typed λ-terms with function symbols of fixed arity defined by rewrite rules, and computation is defined as the combination of β-reduction and rewriting.

These three approaches can be seen as operating on the same term algebra (λ-calculus with symbols of fixed arity, which is a sub-algebra of the one we consider here) with different reduction strategies wrt β-reduction [vOvR93]: in HALs, there is no restriction; in CRSs, every rewrite step is followed by a β-development of the substituted variables (see the notion of valuation in Definition 25); finally, in HRSs, terms are β-normalized.

More precisely, in a CRS, a term is either a variable x, an abstraction λxt, or the application of a function symbol f to a fixed number of terms. A CRS term is therefore in β-normal form. The set of CRS terms is a subset of the set of terms that is stable by reduction or expansion (if matching substitutions are restricted to CRS terms). Only rewrite rules can contain terms of the form $x\vec{t}$, but every rewrite step is followed by a β-development. Hence, the termination of a CRS can be reduced to the termination of the corresponding HAL, because a rewrite step in a CRS is included in the relation $\to_{\mathcal{R},\beta\eta} \cup \to_{\beta}^{*}$.

In an HRS, terms are in β-normal η-long form and, after a rewrite step, terms are β-normalized and η-expanded if necessary [MN98]. Hence, in an HRS, the reduction relation is $\to_{\mathcal{R}}\to_{\beta}^{!}\to_{\overline{\eta}}^{!}$, where $\to_{\overline{\eta}}$ is the relation of *η-expansion*[42] and $R^{!}$ denotes normalization wrt R. Hence, our results can directly apply to HRSs if the set of terms in η-long form is stable by rewriting for, in this case, no η-expansion is necessary[43]. This is in particular the case if the right-hand side of every rule is in η-long form [Hue76]. Otherwise, one needs to extend our results by proving the termination of $\to \cup \to_{\overline{\eta}}$ instead (see [CK96] for the case where $\to = \to_{\beta} \cup \to_{\mathcal{R}}$ and $\mathcal{R}$ is a set of algebraic rewrite rules).

7. Conclusion

We have provided a new, more general, presentation of the notion of computability closure [BJO02] and how it can be extended to deal with different kinds of rewrite relations (rewriting modulo some equational theory and rewriting with matching modulo $\beta\eta$) and applied to other frameworks for higher-order rewriting (Section 6.5). In particular, for dealing with recursive function definitions, we introduced a new more general rule (Figure 4) based on the notion of $\mathcal{F}$-quasi-ordering compatible with application (Definition 6).

Parts of this work have been formalized in the proof assistant Coq [Coq14]: *pure* λ-terms[44], computability predicates on (untyped) λ-terms, simply-typed λ-terms using typing environments, the interpretation of types using accessible arguments as in Section 4.7, and the smallest computability closure closed by the operations I, II, IV and V [Bla13].[45] Therefore, the complete formalization of the results presented in this paper is not out of reach. In particular, the operations III and VI, and the computability closure for rewriting modulo some equational theory. On the other hand, the computability closure for rewriting with matching modulo $\beta\eta$ seems more difficult.

For the sake of simplicity, we have presented this work in Church simply-typed λ-calculus [Chu40] but, at the price of heavier notations, a special care for type variables, and assuming that $\alpha_{\mathsf{f}} = \mathsf{sup}\{|\vec{l}| \mid \mathsf{f}\vec{l} \to r \in \mathcal{R}\}$

[42] That is, the relation $\leftarrow_{\eta}$ restricted to terms not of the form λxt and to contexts not of the form $C[[]u]$ [CK96].

[43] The set of terms in η-long form is stable by β-reduction [Hue76].

[44] Using named variables and explicit α-equivalence [CF58] which is closer to informal practice than de Bruijn indices [dB72].

[45] The definitions and theorems without their proofs are available on `http://color.inria.fr/doc/main.html`. In particular, λ-calculus is formalized in the files `LTerm.v`, `LSubs.v`, `LAlpha.v`, `LBeta.v` and `LSimple.v`; computability is formalized in `LComp.v`, `LCompRewrite.v` and `LCompSimple.v`; the interpretation of type constants as in Section 4.7 is formalized in `LCompInt.v`; the notion of $\mathcal{F}$-quasi-ordering is formalized in `LCall.v`; and the notion of computability closure is formalized in `LCompClos.v`. As an example, Gödel system T is proved terminating in `LSystemT.v` by using the lexicographic status $\mathcal{F}$-quasi-ordering $(\trianglerighteq_{\mathsf{s}})_{\mathrm{lex}}$.

is finite,[46] these results can be extended to polymorphic and dependent types, and type-level rewriting (*e.g.* strong elimination), following the techniques developed in [Bla05].

But the notion of computability closure has other interesting properties or applications:

- As shown in [Bla06a], it has some important relationship with the notion of dependency pair [AG00] and can indeed be used to improve the static approach to higher-order dependency pairs [KISB09].
- The notion of computability closure and Jouannaud and Rubio's higher-order recursive path ordering (HORPO) [JR99, JR07] share many similarities. The notion of computability closure is even used in HORPO for strengthening it. HORPO is potentially more powerful than CC because, when comparing the left-hand side of a rule $\mathsf{f}\vec{l}$ with its corresponding right-hand side r, in CC, the subterms of r must be compared with $\mathsf{f}\vec{l}$ itself while, in HORPO, the subterms of r may be compared with subterms of $\mathsf{f}\vec{l}$. However, in [Bla06b], I showed that HORPO is *included* in the monotone closure of the least *fixpoint* of the monotone function $\mathcal{R} \mapsto \{(\mathsf{f}\vec{l}, r) \mid r \in \mathrm{CC}_{\mathsf{f}}(\vec{l}), \tau(\mathsf{f}\vec{l}) = \tau(r), \mathrm{FV}(r) \subseteq \mathrm{FV}(l)\}$ (where CC is the smallest computability closure defined by the rules I to IV), and that Dershowitz' first-order recursive path ordering [Der82] is *equal* to this fixpoint (when CC is restricted to first-order terms). This and the fact that HORPO could not handle the examples of Section 4.6 motivated a series of papers culminating in the definition of the computability path ordering (CPO) subsuming both HORPO and CC, but currently limited to matching modulo α-equivalence [BJR08, BJR15].
- In Section 4.6, we have seen that, on non-strictly positive inductive types, the computability closure can handle recursors (by using an elimination-based interpretation of types), but cannot handle arbitrary function definitions (*e.g.* the function ex). This can however be achieved by extending the type system with size annotations (interpreted as ranks) and using an $\mathcal{F}$-quasi-ordering comparing size annotations. This line of research was initiated independently by Giménez [Gim96] and Hughes, Pareto and Sabry [HPS96], and further developed by Xi [Xi02], Abel [Abe04], Barthe *et al* [BFG$^+$04] and myself [Bla04]. By considering explicit quantifications and constraints on size annotations, one can even handle conditional rewrite rules [BR06]. Moreover, in [BR09], Roux and I showed that these developments can to some extent be seen as an instance of higher-order semantic labeling [Zan95, Ham07], a technique which consists in annotating function symbols with the semantics of theirs arguments in some model of the rewrite system.
- In [JR06], using a complex notion of "neutralization" that requires the introduction of new function symbols, Jouannaud and Rubio provide a general method for building a reduction ordering for rewriting with matching modulo $\beta\eta$ on *β-normal* terms from a reduction ordering for rewriting with matching modulo α-equivalence on arbitrary terms, if the latter satisfies some conditions. Then, they provide a restriction of HORPO satisfying the required conditions. A precise comparison between this approach and the one developed in Section 6 remains to be done. It could perhaps shed some light on this notion of neutralization.

Acknowledgements. The author thanks very much the anonymous referees for their very careful reading and many suggestions, and Ali Assaf and Ronan Saillard for their comments on Section 6.

References

[Abe04] A. Abel. Termination checking with types. *Theoretical Informatics and Applications*, 38(4):277–319, 2004.

[Abe06] A. Abel. *A polymorphic lambda-calculus with sized higher-order types.* PhD thesis, Ludwig-Maximilians-Universität München, Germany, 2006.

[ACCL91] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[AG00] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.

[46] Because, in this case, α_{f} may be infinite if $\mathcal{R}$ is infinite, which may be the case if one considers the rewrite relation generated by a conditional rewrite system, or applies some semantic labeling to a finite rewrite system [Zan95].

[ARCT11] A. Asperti, W. Ricciotti, C. Sacerdoti Coen, and E. Tassi. The Matita interactive theorem prover. In *Proceedings of the 23rd International Conference on Automated Deduction*, Lecture Notes in Computer Science 6803, 2011.

[Bar84] H. Barendregt. *The lambda calculus: its syntax and semantics*. North-Holland, 2nd edition, 1984.

[Bar92] H. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of logic in computer science. Volume 2. Background: computational structures*, pages 117–309. Oxford University Press, 1992.

[BDN09] A. Bove, P. Dybjer, and U. Norell. A brief overview of Agda - a functional language with dependent types. In *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science 5674, 2009.

[BFG97] F. Barbanera, M. Fernández, and H. Geuvers. Modularity of strong normalization in the algebraic-λ-cube. *Journal of Functional Programming*, 7(6):613–660, 1997.

[BFG+04] 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.

[BJO99] 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.

[BJO02] F. Blanqui, J.-P. Jouannaud, and M. Okada. Inductive-data-type systems. *Theoretical Computer Science*, 272:41–68, 2002.

[BJR08] F. Blanqui, J.-P. Jouannaud, and A. Rubio. The computability path ordering: the end of a quest. In *Proceedings of the 22nd International Conference on Computer Science Logic*, Lecture Notes in Computer Science 5213, 2008. Invited paper.

[BJR15] F. Blanqui, J.-P. Jouannaud, and A. Rubio. The computability path ordering. *Logical Methods in Computer Science*, ?(?):?–?, 2015. To appear.

[BKR05] E. Bonelli, D. Kesner, and A. Ríos. Relating higher-order and first-order rewriting. *Journal of Logic and Computation*, 15:901–947, 2005.

[Bla00] 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.

[Bla03] 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.

[Bla04] F. Blanqui. A type-based termination criterion for dependently-typed higher-order rewrite systems. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 3091, 2004.

[Bla05] F. Blanqui. Definitions by rewriting in the calculus of constructions. *Mathematical Structures in Computer Science*, 15(1):37–92, 2005.

[Bla06a] F. Blanqui. Higher-order dependency pairs. In *8th International Workshop on Termination*, 2006.

[Bla06b] F. Blanqui. (HO)RPO revisited. Technical Report 5972, INRIA, France, 2006.

[Bla07] F. Blanqui. Computability closure: ten years later. In *Rewriting, Computation and Proof – Essays Dedicated to Jean-Pierre Jouannaud on the Occasion of His 60th Birthday*, volume 4600 of *Lecture Notes in Computer Science*, 2007.

[Bla13] F. Blanqui. A formalization in Coq of the notion of computability closure for proving the termination of rewrite relations on λ-terms, 2013.

[BR06] F. Blanqui and C. Riba. Combining typing and size constraints for checking the termination of higher-order conditional rewrite systems. In *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science 4246, 2006.

[BR09] F. Blanqui and C. Roux. On the relation between sized-types based termination and semantic labelling. In *Proceedings of the 23rd International Conference on Computer Science Logic*, Lecture Notes in Computer Science 5771, 2009.

[BTG89] 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.

[BTG91] V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic strong normalization. *Theoretical Computer Science*, 83(1):3–28, 1991.

[CC79] P. Cousot and R. Cousot. Constructive versions of Tarski's fixed point theorems. *Pacific Journal of Mathematics*, 82(1):43–57, 1979.

[CF58] H. B. Curry and R. Feys. *Combinatory logic*. North-Holland, 1958.

[CH84] T. Coquand and G. Huet. A theory of constructions, 1984. Paper presented at the International Symposium on Semantics of Data Types but not published in the proceedings.

[CH88] T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76(2-3):95–120, 1988.

[Chu40] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[CJ03] H. Comon and J.-P. Jouannaud. Les termes en logique et en programmation, 2003. Lecture notes.

[CK96] R. Di Cosmo and D. Kesner. Combining algebraic rewriting, extensional lambda calculi, and fixpoints. *Theoretical Computer Science*, 169(2):201–220, 1996.

[CK01a] H. Cirstea and C. Kirchner. The rewriting calculus - part I. *Logic Journal of the Interest Group in Pure and applied Logic*, 9(3):339–375, 2001.

[CK01b] H. Cirstea and C. Kirchner. The rewriting calculus - part II. *Logic Journal of the Interest Group in Pure and*

applied Logic, 9(3):377–410, 2001.

[Coq92] T. Coquand. Pattern matching with dependent types. In *Proceedings of the International Workshop on Types for Proofs and Programs*, 1992.

[Coq14] INRIA, France. *The Coq reference manual, version 8.4pl5*, 2014.

[CPM88] T. Coquand and C. Paulin-Mohring. Inductively defined types. In *Proceedings of the International Conference on Computer Logic*, Lecture Notes in Computer Science 417, 1988.

[dB72] N. G. de Bruijn. Lambda-calculus notation with nameless dummies: a tool for automatic formula manipulation with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34(5):381–392, 1972.

[dB78] N. G. de Bruijn. A namefree lambda calculus with facilities for internal definition of expressions and segments. TH-Report 78-WSK-03, Department of Mathematics, Technological University Eindhoven, NL, 1978.

[Der79] N. Dershowitz. Orderings for term rewriting systems. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, 1979.

[Der82] N. Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science. Volume B: formal models and methods*, chapter 6, pages 243–320. North-Holland, 1990.

[DM79] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.

[Dou91] 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.

[Dou92] D. Dougherty. Adding algebraic rewriting to the untyped lambda calculus. *Information and Computation*, 101(2):251–267, 1992.

[Eke96] S. Eker. Fast matching in combinations of regular equational theories. In *Proceedings of the 1st International Workshop on Rewriting Logic and Applications*, Electronic Notes in Theoretical Computer Science 4, 1996.

[Erw96] M. Erwig. Active patterns. In *Proceedings of the 8th International Workshop on Implementation of Functional Languages*, Lecture Notes in Computer Science 1268, 1996.

[FJ94] M. Fernández and J.-P. Jouannaud. Modular termination of term rewriting systems revisited. In *Proceedings of the 10th International Workshop on Specification of Abstract Data Types*, Lecture Notes in Computer Science 906, 1994.

[FK12] C. Fuhs and C. Kop. Polynomial interpretations for higher-order rewriting. In *Proceedings of the 23rd International Conference on Rewriting Techniques and Applications*, Leibniz International Proceedings in Informatics 15, 2012.

[FPT99] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Proceedings of the 14th IEEE Symposium on Logic in Computer Science*, 1999.

[Gal90] J. Gallier. On Girard's "candidats de réductibilité". In P.-G. Odifreddi, editor, *Logic and Computer Science*, number 31 in APIC Studies in Data Processing, pages 123–203. Academic Press, 1990.

[Gan80] R. O. Gandy. Proofs of strong normalization. In J. R. Hindley and J. P. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 457–477. Academic Press, 1980.

[Gim96] E. Giménez. *Un calcul de constructions infinies et son application à la vérification de systèmes communiquants*. PhD thesis, ENS Lyon, France, 1996.

[Gim98] E. Giménez. Structural recursive definitions in type theory. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 1443, 1998.

[Gir71] 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, *Proceedings of the 2nd Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 63–92. North-Holland, 1971.

[Gir72] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris 7, France, 1972.

[GKK05] J. Glauert, D. Kesner, and Z. Khasidashvili. Expression reduction systems and extensions: an overview. In *Processes, Terms and Cycles: Steps to the Road of Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday*, volume 3838 of *Lecture Notes in Computer Science*, 2005.

[GLT88] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and types*. Cambridge University Press, 1988.

[Göd58] K. Gödel. über einer bisher noch nicht benützte erweiterung des finiten standpunktes. *Dialectica*, 12:280–287, 1958. Reprinted in [Göd90].

[Göd90] K. Gödel. *Collected works - vol. 2: publications 1938-1974*. Oxford University Press, 1990.

[Gra91] B. Gramlich. A structural analysis of modular termination of term rewriting systems. SEKI-Report SR-91-15, Fachbereich Informatik, Universität Kaiserslautern, Germany, 1991.

[Gra94] B. Gramlich. Generalized sufficient conditions for modular termination of rewriting. *Applicable Algebra in Engineering Communication and Computing*, 5(3-4):131–158, 1994.

[Ham06] M. Hamana. An initial algebra approach to term rewriting systems with variable binders. *Journal of Higher-Order and Symbolic Computation*, 19(2-3):231–262, 2006.

[Ham07] M. Hamana. Higher-order semantic labelling for inductive datatype systems. In *Proceedings of the 9th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, 2007.

[Hes09] G. Hessenberg. Kettentheorie and Wohlordnung. *Journal für die reine und angewandte Mathematik*, 135:81–133, 1909.

[Hof95] M. Hofmann. Approaches to recursive data types - a case study. Unpublished note cited in [Mat00] p. 61, 1995.

[HPS96] J. Hughes, L. Pareto, and A. Sabry. Proving the correctness of reactive systems using sized types. In *Proceedings of the 23th ACM Symposium on Principles of Programming Languages*, 1996.

[Hue76] G. Huet. Résolution d'équations dans les langages d'ordre 1, 2, ..., ω, 1976. Thèse d'État, Université Paris 7, France.

[Hue80] G. Huet. Confluent reductions: abstract properties and applications to term-rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.

[Jay04] C. B. Jay. The pattern calculus. *ACM Transactions on Programming Languages and Systems*, 26(6):911–937, 2004.

[JB04] N. D. Jones and N. Bohr. Termination analysis of the untyped lambda-calculus. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 3091, 2004.

[JK86] 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.

[JK09] C. B. Jay and Delia Kesner. First-class patterns. *Journal of Functional Programming*, 19(2):191–225, 2009.

[JM84] J.-P. Jouannaud and M. Muñoz. Termination of a set of rules modulo a set of equations. In *Proceedings of the 7th International Conference on Automated Deduction*, Lecture Notes in Computer Science 170, 1984.

[JO91] J.-P. Jouannaud and M. Okada. A computation model for executable higher-order algebraic specification languages. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, 1991.

[JO97a] J.-P. Jouannaud and M. Okada. Abstract data type systems. *Theoretical Computer Science*, 173(2):349–391, 1997.

[JO97b] J.-P. Jouannaud and M. Okada. Inductive data type systems: strong normalization and all that. Draft, 1997.

[JR99] 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.

[JR06] 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.

[JR07] J.-P. Jouannaud and A. Rubio. Polymorphic higher-order recursive path orderings. *Journal of the ACM*, 54(1):1–48, 2007.

[KB70] D. Knuth and P. Bendix. Simple word problems in universal algebra. In *Computational problems in abstract algebra, Proceedings of a Conference held at Oxford in 1967*, pages 263–297. Pergamon Press, 1970. Reproduced in [SW83].

[Kes07] D. Kesner. The theory of explicit substitutions revisited. In *Proceedings of the 21th International Conference on Computer Science Logic*, Lecture Notes in Computer Science 4646, 2007.

[Kha90] Z. Khasidashvili. Expression reduction systems. Technical Report 36, I. Vekua Institute of Applied Mathematics of Tbilisi State University, 1990.

[KISB09] K. Kusakari, Y. Isogai, M. Sakai, and F. Blanqui. Static dependency pair method based on strong computability for higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E92-D(10):2007–2015, 2009.

[KL80] S. Kamin and J.-J. Lévy. Attempts for generalizing the recursive path orderings. Unpublished note, 1980.

[Klo80] J. W. Klop. *Combinatory reduction systems.* PhD thesis, Utrecht Universiteit, NL, 1980. Published as Mathematical Center Tract 129.

[KM01] H. Kirchner and P.-E. Moreau. Promoting rewriting to a programming language: a compiler for non-deterministic rewrite programs in associative-commutative theories. *Journal of Functional Programming*, 11(2):207–251, 2001.

[KvOdV08] J. W. Klop, V. van Oostrom, and R. de Vrijer. Lambda calculus with patterns. *Theoretical Computer Science*, 398(1-3):16–31, 2008.

[KvOvR93] J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993.

[LB77] D. Lankford and A. Ballantyne. Decision procedures for simple equational theories with commutative-associative axioms: complete sets of commutative-associative reductions. Technical Report ATP-039, Automatic Theorem Proving Project, University of Texas, Austin, USA, 1977.

[Loa03] R. Loader. Higher-order β-matching is undecidable. *Logic Journal of the Interest Group in Pure and applied Logic*, 11(1):51–68, 2003.

[LSS92] 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.

[Mat98] R. Matthes. *Extensions of system F by iteration and primitive recursion on monotone inductive types.* PhD thesis, Ludwig Maximilians Universität, München, Germany, 1998.

[Mat00] R. Matthes. *Lambda calculus: a case for inductive definitions*, 2000.

[Men87] N. P. Mendler. *Inductive definition in type theory.* PhD thesis, Cornell University, USA, 1987.

[Mil91] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[MN98] R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(2):3–29, 1998.

[Ned73] R. Nederpelt. *Strong normalization in a typed lambda calculus with lambda structured types.* PhD thesis, Technische Universiteit Eindhoven, NL, 1973.

[Nip91] T. Nipkow. Higher-order critical pairs. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, 1991.

[Oka89] M. Okada. Strong normalizability for the combined system of the typed lambda calculus and an arbitrary convergent term rewrite system. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, 1989.

[Par93] M. Parigot. Strong normalization for second order classical natural deduction. In *Proceedings of the 8th IEEE Symposium on Logic in Computer Science*, 1993.

[Par97] M. Parigot. Proofs of strong normalization for second order classical natural deduction. *Journal of Symbolic Logic*, 62(4):1461–1479, 1997.

[Pau86] L. Paulson. Constructing recursion operators in intuitionistic type theory. *Journal of Symbolic Computation*, 2(4):325–355, 1986.

[Pot78] G. Pottinger. Proofs of the normalization and Church-Rosser theorems for the typed λ-calculus. *Notre Dame Journal of Formal Logic*, 19(3):445–451, 1978.

[PS81] G. Peterson and M. Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM*, 28(2):233–264, 1981.

[Qia93] Z. Qian. Linear unification of higher-order patterns. In *Proceedings of the 5th International Joint Conference on Theory and Practice of Software Development*, Lecture Notes in Computer Science 668, 1993.

[Rib07a] C. Riba. On the stability by union of reducibility candidates. In *Proceedings of the 10th International Conference on Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science 4423, 2007.

[Rib07b] C. Riba. Strong normalization as safe interaction. In *Proceedings of the 22nd IEEE Symposium on Logic in Computer Science*, 2007.

[Rou11] C. Roux. *Size-based termination: semantics and generalizations*. PhD thesis, Université Henri Poincaré, Nancy, France, 2011.

[San67] L. E. Sanchis. Functionals defined by recursion. *Notre Dame Journal of Formal Logic*, 8:161–174, 1967.

[Sta79] R. Statman. The typed λ-calculus is not elementary recursive. *Theoretical Computer Science*, 9:73–81, 1979.

[Sti09] C. Stirling. Decidability of higher-order matching. *Logical Methods in Computer Science*, 5(3):1–52, 2009.

[Str07] L. Straßburger. A characterisation of Medial as rewriting rule. In *Proceedings of the 18th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 4533, 2007.

[SW83] J. H. Siekmann and G. Wrightson, editors. *Automation of reasoning. 2: classical papers on computational logic 1967-1970*. Symbolic computation. Springer, 1983.

[SWS01] M. Sakai, Y. Watanabe, and T. Sakabe. An extension of dependency pair method for proving termination of higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E84-D(8):1025–1032, 2001.

[Tai67] W. W. Tait. Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967.

[Tai75] 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.

[Tak95] M. Takahashi. Parallel reductions in λ-calculus. *Information and Computation*, 118:120–127, 1995.

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[TeR03] TeReSe. *Term rewriting systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

[Toy87] Y. Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25(3):141–143, 1987.

[Tro73] A. S. Troelstra. Models and Computability. In A. S. Troelstra, editor, *Metamathematical investigation of intuitionistic arithmetic and analysis*, volume 344 of *Lecture Notes in Mathematics*, pages 97–174. Springer, 1973.

[Tul10] M. Tullsen. First class patterns. In *Proceedings of the 2nd International Symposium on Practical Aspects of Declarative Languages*, Lecture Notes in Computer Science 1753, 2010.

[Tur37] A. M. Turing. Computability and λ-definability. *Journal of Symbolic Logic*, 2(153-163), 1937.

[vD80] D. van Daalen. *The language theory of Automath*. PhD thesis, Eindhoven University of Technology, NL, 1980.

[vdP96] J. van de Pol. *Termination of higher-order rewrite systems*. PhD thesis, Utrecht Universiteit, NL, 1996.

[VM04] J. Vouillon and P.-A. Melliès. Semantic types: a fresh look at the ideal model for types. In *Proceedings of the 31st ACM Symposium on Principles of Programming Languages*, 2004.

[vO90] V. van Oostrom. Lambda calculus with patterns. Technical Report IR 228, Vrije Universiteit, Amsterdam, NL, 1990.

[vO94] V. van Oostrom. *Confluence for abstract and higher-order rewriting*. PhD thesis, Vrije Universiteit Amsterdam, NL, 1994.

[vOvR93] V. van Oostrom and F. van Raamsdonk. Comparing combinatory reduction systems and 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.

[vR96] F. van Raamsdonk. *Confluence and normalization for higher-order rewriting*. PhD thesis, Vrije University Amsterdam, NL, 1996.

[Wer94] B. Werner. *Une théorie des constructions inductives*. PhD thesis, Université Paris 7, France, 1994.

[Xi02] H. Xi. Dependent types for program termination verification. *Journal of Higher-Order and Symbolic Computation*, 15(1):91–131, 2002.

[Zan95] H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.