Récriture d'Ordre Supérieur

Jean-Pierre Jouannaud
École Polytechnique
91400 Palaiseau, France

email: jouannaud@lix.polytechnique.fr
http: //w^3.lix.polytechnique.fr/Labo/Jean-Pierre.Jouannaud

Project LogiCal, Pôle Commun de Recherche en Informatique du Plateau de Saclay, CNRS, École Polytechnique, INRIA, Université Paris-Sud.

January 3, 2005

Outline

Données initiales

1. une partition $Mul \uplus Lex$ of $\mathcal{F} \cup \mathcal{S}$ tq $S \subseteq Lex$;
2. un préordre bien-fondé $\geq_{\mathcal{FS}}$ sur $\mathcal{F} \cup \{@\} \cup \mathcal{S} \cup \{\rightarrow\}$, la *précédence*, telle que:
   (i) $\geq_{\mathcal{FS}}$ est l'union de ses restrictions à $\mathcal{F} \cup \{@\}$ et $\mathcal{S} \cup \{\rightarrow\}$,
   (ii) $\forall s \in \mathcal{S} : s \not\geq_{\mathcal{FS}} \rightarrow$;
   (iii) $\forall f \in \mathcal{F} : f >_{\mathcal{FS}} @$;
   (iv) si $f : \overline{\sigma} \rightarrow \sigma =_{\mathcal{FS}} g : \overline{\tau} \rightarrow \tau$, alors
   a. $\sigma =_{\mathcal{T_S}} \tau$, et
   b. $f \in Lex$ ssi $g \in Lex$,
   $arity(f) = arity(g) = n$ et $\overline{\sigma}\ (=_{\mathcal{FS}})^n\ \overline{\tau}$;
3. $>_{\mathcal{FS}}$ s'étend à $\mathcal{F} \cup \{@\} \cup \mathcal{X} \cup \mathcal{S} \cup \{\rightarrow\} \cup \mathcal{S}^{\forall}$ par ajout des paires $x \geq_{\mathcal{FS}} x \ \ \forall x \in \mathcal{X} \cup \mathcal{S}^{\forall}$.

Données initiales

1. une partition $\mathit{Mul} \uplus \mathit{Lex}$ of $\mathcal{F} \cup \mathcal{S}$ tq $S \subseteq \mathit{Lex}$;
2. un préordre bien-fondé $\geq_{\mathcal{F}\mathcal{S}}$ sur $\mathcal{F} \cup \{@\} \cup \mathcal{S} \cup \{\rightarrow\}$, la *précédence*, telle que:
   (i) $\geq_{\mathcal{F}\mathcal{S}}$ est l'union de ses restrictions à $\mathcal{F} \cup \{@\}$ et $\mathcal{S} \cup \{\rightarrow\}$,
   (ii) $\forall s \in \mathcal{S} : s \neq_{\mathcal{F}\mathcal{S}} \rightarrow$;
   (iii) $\forall f \in \mathcal{F} : f >_{\mathcal{F}\mathcal{S}} @$;
   (iv) si $f : \overline{\sigma} \rightarrow \sigma =_{\mathcal{F}\mathcal{S}} g : \overline{\tau} \rightarrow \tau$, alors
   a. $\sigma =_{\mathcal{T}_\mathcal{S}} \tau$, et
   b. $f \in \mathit{Lex}$ ssi $g \in \mathit{Lex}$,
   $\mathit{arity}(f) = \mathit{arity}(g) = n$ et $\overline{\sigma}\ (=_{\mathcal{F}\mathcal{S}})^n\ \overline{\tau}$;
3. $>_{\mathcal{F}\mathcal{S}}$ s'étend à $\mathcal{F} \cup \{@\} \cup \mathcal{X} \cup \mathcal{S} \cup \{\rightarrow\} \cup \mathcal{S}^\forall$ par ajout des paires $x \geq_{\mathcal{F}\mathcal{S}} x \ \ \forall x \in \mathcal{X} \cup \mathcal{S}^\forall$.

Données initiales

1. une partition $\mathit{Mul} \uplus \mathit{Lex}$ of $\mathcal{F} \cup \mathcal{S}$ tq $S \subseteq \mathit{Lex}$;
2. un préordre bien-fondé $\geq_{\mathcal{F}\mathcal{S}}$ sur $\mathcal{F} \cup \{@\} \cup \mathcal{S} \cup \{\rightarrow\}$, la *précédence*, telle que:
   (i) $\geq_{\mathcal{F}\mathcal{S}}$ est l'union de ses restrictions à $\mathcal{F} \cup \{@\}$ et $\mathcal{S} \cup \{\rightarrow\}$,
   (ii) $\forall s \in \mathcal{S} : s \not=_{\mathcal{F}\mathcal{S}} \rightarrow$;
   (iii) $\forall f \in \mathcal{F} : f >_{\mathcal{F}\mathcal{S}} @$;
   (iv) si $f : \overline{\sigma} \rightarrow \sigma =_{\mathcal{F}\mathcal{S}} g : \overline{\tau} \rightarrow \tau$, alors
   a. $\sigma =_{\mathcal{T}_\mathcal{S}} \tau$, et
   b. $f \in \mathit{Lex}$ ssi $g \in \mathit{Lex}$,
   $\mathit{arity}(f) = \mathit{arity}(g) = n$ et $\overline{\sigma}\ (=_{\mathcal{F}\mathcal{S}})^n\ \overline{\tau}$;
3. $>_{\mathcal{F}\mathcal{S}}$ s'étend à $\mathcal{F} \cup \{@\} \cup \mathcal{X} \cup \mathcal{S} \cup \{\rightarrow\} \cup \mathcal{S}^\forall$ par ajout des paires $x \geq_{\mathcal{F}\mathcal{S}} x \ \ \forall x \in \mathcal{X} \cup \mathcal{S}^\forall$.

Définition de HORPO sur $\mathcal{T} \cup \mathcal{T}_{\mathcal{S}^\forall}$

$s : \sigma \succ t : \tau$ ssi

Type : $\sigma = \tau = *$ ou $\sigma \succeq \tau$

et

Clôture: $s \doteq f(\overline{s})$ et $\exists u \in \mathcal{CC}(s,t)$: $u \succeq t$

Prec: $s \doteq f(\overline{s})$, $t \doteq g(\overline{t})$, $f >_{\mathcal{FS}} g$ et A

Prec@: $s \doteq f(\overline{s})$, $f \in \mathcal{F}$, $t \doteq @(\overline{t})$ et A

Precλ: $s \doteq f(\overline{s})$, $f \in \mathcal{F}$, $t \doteq \lambda x : \alpha.v$, $x \notin \mathcal{V}ar(v)$ et $s \succ v$

Mul: $s \doteq f(\overline{s})$, $t \doteq g(\overline{t})$, $f =_{\mathcal{FS}} g \in Mul$ et $\overline{s} \succ_{mul} \overline{t}$

Lex: $s \doteq f(\overline{s})$, $t \doteq g(\overline{t})$, $f =_{\mathcal{FS}} g \in Lex$, $\overline{s} \succ_{lex} \overline{t}$, A

Mul@: $s \doteq @(s_1, s_2)$, $t \doteq @(\overline{t})$ et $\{s_1, s_2\} \succ_{mul} \overline{t}$

Définition, suite

Monλ: $s \doteq \lambda x : \alpha.u,\ t \doteq \lambda y : \beta.v$, $\alpha =_{\mathcal{T}_\mathcal{S}} \beta$ et $u \succ v\{y \mapsto x\}$

Mon$\rightarrow$: $s \doteq \alpha \rightarrow \beta,\ t \doteq \alpha' \rightarrow \beta'$, $\alpha =_{\mathcal{T}_\mathcal{S}} \alpha'$ et $\beta \succ \beta'$

Reduction: $s \doteq @(\lambda x.u, v)$ et $u\{x \mapsto v\} \succeq t$

$A \doteq \forall v \in \bar{t}$: $s \succ v$ ou $\exists u \in \mathcal{CC}(s)$: $u \succeq v$

$\mathcal{CC}(f(\bar{s}), t) \doteq \bar{s} \quad \forall f \in \mathcal{F} \cup \{@\}$

$\mathcal{CC}(\lambda x : \sigma.u, t) \doteq \{u\}$ si $x \notin \mathcal{V}ar(t)$ sinon $\emptyset$

$\mathcal{CC}(\alpha \rightarrow \beta, \tau) \doteq \{\beta\}$

$@(\bar{t})$d est, une application étendue de t

$\succeq \doteq \succ \cup =$, où $=$ est l'équivalence sur $\mathcal{T}(\mathcal{F}, \mathcal{X})$ engendrée par permutation de $Mul \subseteq \mathcal{F}$

$=_{\mathcal{T}}$ et $=_{\mathcal{T}_\mathcal{S}}$ les restriction de $=$ à $\mathcal{T}(\mathcal{F}, \mathcal{X})$ et $\mathcal{T}_{\mathcal{S}^\forall}$

Bonne définition de HORPO

1. $\succ$ est défini par récurrence sur l'ordre $(\longrightarrow_\beta \cup \rhd, \rhd)_{lex}$ appliqué à la paire d'arguments (s, t).
2. Cet ordre bien-fondé sera utilisé dans certaines preuves sous le terme "récrurrence sur la définition" (de HORPO).

Bonne définition de HORPO

1. $\succ$ est défini par récurrence sur l'ordre $(\longrightarrow_{\beta} \cup \rhd, \rhd)_{lex}$ appliqué à la paire d'arguments (s, t).
2. Cet ordre bien-fondé sera utilisé dans certaines preuves sous le terme “récrurrence sur la définition” (de HORPO).

Exemple: terminaison d'un itérateur polymorphe

$\{x : \alpha,\ l : list(\alpha),\ X : \alpha \rightarrow \alpha\} \vdash$

1. **But:** $map(cons(x,l),X) \succ cons(@(X,x),map(l,X))$
2. **Prec:** $map(cons(x,l),X) \succ \{@(X,x),map(l,X)\}$
3. **Type:** $list(\alpha) \succ \alpha$ (par **Clôture**)
4. **Prec@:** $X \succeq X$ (trivial) et $cons(x,l) \succeq x$
5. **Type:** $list(\alpha) \succeq \alpha \rightarrow \alpha$ (par **Prec**)
6. **Prec:** $x \succeq x$ (trivial)
7. **Mul:** $\{cons(x,l),X\} \succ_{mul} \{l,X\}$
8. $X \succeq X$ (trivial)
9. **Prec:** $l \succeq l$ (trivial et FIN)

Exemple: terminaison d'un itérateur polymorphe

$\{x : \alpha,\ l : list(\alpha),\ X : \alpha \rightarrow \alpha\} \vdash$

1. **But:** $map(cons(x,l),X) \succ cons(@(X,x),map(l,X))$
2. **Prec:** $map(cons(x,l),X) \succ \{@(X,x),map(l,X)\}$
3. **Type:** $list(\alpha) \succ \alpha$ (par **Clôture**)
4. **Prec@:** $X \succeq X$ (trivial) et $cons(x,l) \succeq x$
5. **Type:** $list(\alpha) \succeq \alpha \rightarrow \alpha$ (par **Prec**)
6. **Prec:** $x \succeq x$ (trivial)
7. **Mul:** $\{cons(x,l),X\} \succ_{mul} \{l,X\}$
8. $X \succeq X$ (trivial)
9. **Prec:** $l \succeq l$ (trivial et FIN)

Exemple: terminaison d'un itérateur polymorphe

$\{x : \alpha,\ l : list(\alpha),\ X : \alpha \to \alpha\} \vdash$

1. **But:** $map(cons(x,l),X) \succ cons(@(X,x), map(l,X))$
2. **Prec:** $map(cons(x,l),X) \succ \{@(X,x), map(l,X)\}$
3. **Type:** $list(\alpha) \succ \alpha$ (par **Clôture**)
4. **Prec@:** $X \succeq X$ (trivial) et $cons(x,l) \succeq x$
5. **Type:** $list(\alpha) \succeq \alpha \to \alpha$ (par **Prec**)
6. **Prec:** $x \succeq x$ (trivial)
7. **Mul:** $\{cons(x,l),X\} \succ_{mul} \{l,X\}$
8. $X \succeq X$ (trivial)
9. **Prec:** $l \succeq l$ (trivial et FIN)

Exemple: terminaison d'un itérateur polymorphe

$\{x : \alpha,\ l : list(\alpha),\ X : \alpha \rightarrow \alpha\} \vdash$

1. **But:** $map(cons(x,l),X) \succ cons(@(X,x), map(l,X))$
2. **Prec:** $map(cons(x,l),X) \succ \{@(X,x), map(l,X)\}$
3. **Type:** $list(\alpha) \succ \alpha$ (par **Clôture**)
4. **Prec@:** $X \succeq X$ (trivial) et $cons(x,l) \succeq x$
5. **Type:** $list(\alpha) \succeq \alpha \rightarrow \alpha$ (par **Prec**)
6. **Prec:** $x \succeq x$ (trivial)
7. **Mul:** $\{cons(x,l), X\} \succ_{mul} \{l, X\}$
8. $X \succeq X$ (trivial)
9. **Prec:** $l \succeq l$ (trivial et FIN)

Exemple: terminaison d'un itérateur polymorphe

$\{x : \alpha,\ l : list(\alpha),\ X : \alpha \rightarrow \alpha\} \vdash$

1. **But:** $map(cons(x,l),X) \succ cons(@(X,x), map(l,X))$
2. **Prec:** $map(cons(x,l),X) \succ \{@(X,x), map(l,X)\}$
3. **Type:** $list(\alpha) \succ \alpha$ (par **Clôture**)
4. **Prec@:** $X \succeq X$ (trivial) et $cons(x,l) \succeq x$
5. **Type:** $list(\alpha) \succeq \alpha \rightarrow \alpha$ (par **Prec**)
6. **Prec:** $x \succeq x$ (trivial)
7. **Mul:** $\{cons(x,l), X\} \succ_{mul} \{l, X\}$
8. $X \succeq X$ (trivial)
9. **Prec:** $l \succeq l$ (trivial et FIN)

Exemple: terminaison d'un itérateur polymorphe

$\{x : \alpha,\ l : list(\alpha),\ X : \alpha \rightarrow \alpha\} \vdash$

1. **But:**
   $map(cons(x,l),X) \succ cons(@(X,x), map(l,X))$
2. **Prec:**
   $map(cons(x,l),X) \succ \{@(X,x), map(l,X)\}$
3. **Type:** $list(\alpha) \succ \alpha$ (par **Clôture**)
4. **Prec@:** $X \succeq X$ (trivial) et $cons(x,l) \succeq x$
5. **Type:** $list(\alpha) \succeq \alpha \rightarrow \alpha$ (par **Prec**)
6. **Prec:** $x \succeq x$ (trivial)
7. **Mul:** $\{cons(x,l),X\} \succ_{mul} \{l,X\}$
8. $X \succeq X$ (trivial)
9. **Prec:** $l \succeq l$ (trivial et FIN)

Exemple: terminaison d'un itérateur polymorphe

$\{x : \alpha,\ l : list(\alpha),\ X : \alpha \to \alpha\} \vdash$

1. **But:** $map(cons(x,l),X) \succ cons(@(X,x),map(l,X))$
2. **Prec:** $map(cons(x,l),X) \succ \{@(X,x),map(l,X)\}$
3. **Type:** $list(\alpha) \succ \alpha$ (par **Clôture**)
4. **Prec@:** $X \succeq X$ (trivial) et $cons(x,l) \succeq x$
5. **Type:** $list(\alpha) \succeq \alpha \to \alpha$ (par **Prec**)
6. **Prec:** $x \succeq x$ (trivial)
7. **Mul:** $\{cons(x,l),X\} \succ_{mul} \{l,X\}$
8. $X \succeq X$ (trivial)
9. **Prec:** $l \succeq l$ (trivial et FIN)

Exemple: terminaison d'un itérateur polymorphe

$\{x : \alpha,\ l : list(\alpha),\ X : \alpha \rightarrow \alpha\} \vdash$

1. **But:** $map(cons(x, l), X) \succ cons(@(X, x), map(l, X))$
2. **Prec:** $map(cons(x, l), X) \succ \{@(X, x), map(l, X)\}$
3. **Type:** $list(\alpha) \succ \alpha$ (par **Clôture**)
4. **Prec@:** $X \succeq X$ (trivial) et $cons(x, l) \succeq x$
5. **Type:** $list(\alpha) \succeq \alpha \rightarrow \alpha$ (par **Prec**)
6. **Prec:** $x \succeq x$ (trivial)
7. **Mul:** $\{cons(x, l), X\} \succ_{mul} \{l, X\}$
8. $X \succeq X$ (trivial)
9. **Prec:** $l \succeq l$ (trivial et FIN)

Exemple: terminaison d'un itérateur polymorphe

$\{x : \alpha,\ l : list(\alpha),\ X : \alpha \rightarrow \alpha\} \vdash$

1. **But:** $map(cons(x,l),X) \succ cons(@(X,x),map(l,X))$
2. **Prec:** $map(cons(x,l),X) \succ \{@(X,x),map(l,X)\}$
3. **Type:** $list(\alpha) \succ \alpha$ (par **Clôture**)
4. **Prec@:** $X \succeq X$ (trivial) et $cons(x,l) \succeq x$
5. **Type:** $list(\alpha) \succeq \alpha \rightarrow \alpha$ (par **Prec**)
6. **Prec:** $x \succeq x$ (trivial)
7. **Mul:** $\{cons(x,l),X\} \succ_{mul} \{l,X\}$
8. $X \succeq X$ (trivial)
9. **Prec:** $l \succeq l$ (trivial et FIN)

Propriété fondamentale de HORPO

Theorem (jouannaud and Rubio)

1. *La fermeture transitive de la restriction de $\succ$ à $\mathcal{T}_{\mathcal{S}^\forall}$ est un ordre de récriture du premier ordre;*
2. *La fermeture transitive de la restriction de $\succ$ à $\mathcal{T}(\mathcal{F}, \mathcal{X})$ est un ordre de récriture d'ordre supérieur.*

Méthode de preuve

L'inclusion de $\succ$ sur les types dans le rpo de Dershowitz (noté $\succ^{rpo}_{\mathcal{T}_{\mathcal{S}^\forall}}$) montre la bonne fondation sur les types qui joue un rôle crucial. Pour les termes, la preuve de normalisation est basée sur la méthode de Tait-Girard. On va donc introduire un prédicat de réductibilité, montrer qu'il implique la normalisation, et que tout terme est réductible. Notre prédicat sera défini comme plus petit point fixe d'une fonctionnelle positive sur le treillis des parties des termes "candidats", c-a-d typables à équivalence près pour la congruence $=_{\mathcal{T}_{\mathcal{S}}}$. Toutes les propriétés de HORPO seront prouvées pour les termes candidats.

Termes candidats

Variables :
$$\frac{x : \sigma \in \Gamma \quad \sigma' =_{\mathcal{T}_\mathcal{S}} \sigma}{\Gamma \vdash_{\mathcal{F}} x :_C \sigma'}$$

Fun :

$$f : \sigma_1 \times \ldots \times \sigma_n \rightarrow \sigma \in \mathcal{F}$$
ξ some type substitution of domain $\subseteq \bigcup_i \mathcal{V}ar(\sigma_i)$
$$\frac{\Gamma \vdash_{\mathcal{F}} t_1 :_C \sigma_1\xi \quad \ldots \quad \Gamma \vdash_{\mathcal{F}} t_n :_C \sigma_n\xi \quad \tau =_{\mathcal{T}_\mathcal{S}} \sigma\xi}{\Gamma \vdash_{\mathcal{F}} f(t_1, \ldots, t_n) :_C \tau}$$

Abstraction :
$$\frac{\Gamma \cdot \{x : \sigma\} \vdash_{\mathcal{F}} t :_C \tau \quad \theta =_{\mathcal{T}_\mathcal{S}} \sigma}{\Gamma \vdash_{\mathcal{F}} (\lambda x : \sigma . t) :_C \theta \rightarrow \tau}$$

Application :
$$\frac{\Gamma \vdash_{\mathcal{F}} s :_C \sigma \rightarrow \tau \quad \Gamma \vdash_{\mathcal{F}} t :_C \sigma}{\Gamma \vdash_{\mathcal{F}} @(s, t) :_C \tau}$$

Termes candidats, suite

Definition

Terms typable in the previous type system are called *candidate terms*.

Lemma

Let $\Gamma \vdash_{\mathcal{F}} s :_C \sigma$. *Then* $\Gamma \vdash_{\mathcal{F}} s :_C \tau$ *iff* $\sigma =_{\mathcal{T}_{\mathcal{S}}} \tau$.

Proof.

By induction on the type derivation, and by case on the last judgement applied. $\square$

Proof

Variables is trivial. For **Functions**, the if direction results from the transitivity of $=_{\mathcal{T}_\mathcal{S}}$. For the converse, let $\sigma_1\xi, \ldots, \sigma_n\xi$ be the types of $t_1, \ldots, t_n$ in the first type derivation, and $\sigma_1\xi', \ldots, \sigma_n\xi'$ the types of $t_1, \ldots, t_n$ in the second derivation. By induction hypothesis, $\sigma_i\xi' =_{\mathcal{T}_\mathcal{S}} \sigma_i\xi$, hence $\sigma\xi' =_{\mathcal{T}_\mathcal{S}} \sigma\xi$ by property of $=_{\mathcal{T}_\mathcal{S}}$, and we conclude by transitivity of $=_{\mathcal{T}_\mathcal{S}}$. **Abstraction** uses the fact that $\sigma \rightarrow \tau =_{\mathcal{T}_\mathcal{S}} \sigma' \rightarrow \tau'$ iff $\sigma' =_{\mathcal{T}_\mathcal{S}} \sigma$ and $\tau' =_{\mathcal{T}_\mathcal{S}} \tau$ and the induction hypothesis. **Application** uses the induction hypothesis on t to have $\tau' =_{\mathcal{T}_\mathcal{S}} \tau$ and the induction hypothesis on s to ajust the type for the application rule to make sense.

Basic properties of candidate terms

1. Because β-reduction is type preserving, the set of candidate terms is closed under β-reductions;
2. Because β-reduction is strongly normalizing for a typed λ-calculus with arbitrary constants, the set of candidate terms is well-founded with respect to β-reductions (consider a signature in which $f : \sigma'_1 \times \ldots \times \sigma'_n \to \sigma' \in \mathcal{F}$ provided $f : \sigma_1 \times \ldots \times \sigma_n \to \sigma \in \mathcal{F}$ with $\sigma'_i =_{\mathcal{T}_S} \sigma_i$ for every $i \in [1..n]$ and $\sigma' =_{\mathcal{T}_S} \sigma$);
3. HORPO applies to candidate terms as well, by keeping the same definition.

Propriétés de HORPO sur les candidats

1. $\succ$ est monotone;
   (récurence facile sur les contextes)
2. $\succ$ est stable;
   (récurence sur la définition)
3. $\succ$ est compatible;
   (découle des propriétés du typage)
4. $\succ$ est fonctionnel;
   (bati dans la définition)
5. $\succ$ est polymorphe;
   (récurence sur la définition)
6. Reste la bonne fondation.

Propriétés de HORPO sur les candidats

1. $\succ$ est monotone;
   (récurence facile sur les contextes)
2. $\succ$ est stable;
   (récurence sur la définition)
3. $\succ$ est compatible;
   (découle des propriétés du typage)
4. $\succ$ est fonctionnel;
   (bati dans la définition)
5. $\succ$ est polymorphe;
   (récurence sur la définition)
6. Reste la bonne fondation.

Propriétés de HORPO sur les candidats

1. $\succ$ est monotone;
   (récurence facile sur les contextes)
2. $\succ$ est stable;
   (récurence sur la définition)
3. $\succ$ est compatible;
   (découle des propriétés du typage)
4. $\succ$ est fonctionnel;
   (bati dans la définition)
5. $\succ$ est polymorphe;
   (récurence sur la définition)
6. Reste la bonne fondation.

Propriétés de HORPO sur les candidats

1. $\succ$ est monotone;
   (récurence facile sur les contextes)
2. $\succ$ est stable;
   (récurence sur la définition)
3. $\succ$ est compatible;
   (découle des propriétés du typage)
4. $\succ$ est fonctionnel;
   (bati dans la définition)
5. $\succ$ est polymorphe;
   (récurence sur la définition)
6. Reste la bonne fondation.

Propriétés de HORPO sur les candidats

1. $\succ$ est monotone;
   (récurence facile sur les contextes)
2. $\succ$ est stable;
   (récurence sur la définition)
3. $\succ$ est compatible;
   (découle des propriétés du typage)
4. $\succ$ est fonctionnel;
   (bati dans la définition)
5. $\succ$ est polymorphe;
   (récurence sur la définition)
6. Reste la bonne fondation.

Propriétés de HORPO sur les candidats

1. $\succ$ est monotone;
   (récurence facile sur les contextes)
2. $\succ$ est stable;
   (récurence sur la définition)
3. $\succ$ est compatible;
   (découle des propriétés du typage)
4. $\succ$ est fonctionnel;
   (bati dans la définition)
5. $\succ$ est polymorphe;
   (récurence sur la définition)
6. Reste la bonne fondation.

Interpretations

The family of *candidate interpretations* $\{[\![\sigma]\!]\}_{\sigma \in \mathcal{T}_\mathcal{S}}$ is the family of subsets of the set of candidates whose elements are the least sets satisfying the following properties:
(i) If σ is a data type, then $s :_C \sigma \in [\![\sigma]\!]$ iff $t \in [\![\tau]\!] \;\; \forall t :_C \tau$ such that $s \succ t$
(ii) If $s :_C \sigma = \tau \rightarrow \rho$ then $s \in [\![\sigma]\!]$ iff $@(s, t) \in [\![\rho]\!]$ for every $t \in [\![\tau]\!]$;
(iii) If σ is a variable, then $s :_C \sigma \in [\![\sigma]\!]$ iff $s\xi :_C \sigma\xi \in [\![\sigma\xi]\!] \; \forall \xi$ ground.
A candidate term s of type σ is said to be *computable* if $s \in [\![\sigma]\!]$. A vector $\overline{s}$ of terms is computable iff so are all its components.

Basic assumptions and properties

We assume without loss of generality that

1. types are ground
2. functional types are in canonical form: $\sigma = \sigma_1 \to \ldots \to \sigma_n \to \tau$ with τ basic and $n > 0$.

Two basic simple properties are:

1. (i) Assume $\sigma =_{\mathcal{T}_S} \tau$. Then $[\![\sigma]\!] = [\![\tau]\!]$.
2. (ii) Let $\sigma = \sigma_1 \to \ldots \to \sigma_n \to \tau$, where $n > 0$. Then $s \in [\![\sigma]\!]$ iff $@(s, t_1, \ldots, t_n) \in [\![\tau]\!]$ for all $t_1 \in [\![\sigma_1]\!], \ldots, t_n \in [\![\sigma_n]\!]$.

Basic assumptions and properties

We assume without loss of generality that

1. types are ground
2. functional types are in canonical form: $\sigma = \sigma_1 \to \ldots \to \sigma_n \to \tau$ with τ basic and $n > 0$.

Two basic simple properties are:

1. (i) Assume $\sigma =_{\mathcal{T}_\mathcal{S}} \tau$. Then $[\![\sigma]\!] = [\![\tau]\!]$.
2. (ii) Let $\sigma = \sigma_1 \to \ldots \to \sigma_n \to \tau$, where $n > 0$. Then $s \in [\![\sigma]\!]$ iff $@(s, t_1, \ldots, t_n) \in [\![\tau]\!]$ for all $t_1 \in [\![\sigma_1]\!], \ldots, t_n \in [\![\sigma_n]\!]$.

Computability Properties

1. (i) Every computable term is SN;
2. (ii) Assuming that s is computable and $s \succeq t$, then t is computable;
3. (iii) A neutral term s is computable iff t is computable for every t such that $s \succ t$;
4. (iv) If $\bar{t}$ be a vector of computable terms s.t. $@(\bar{t})$ is a candidate, then $@(\bar{t})$ is computable;
5. (v) $\lambda x : \sigma.u$ is computable iff $u\{x \mapsto w\}$ is computable for every computable $w :_C \sigma$;
6. (vi) Let $s :_C \sigma \in \mathcal{T}_S^{min}$. Then s is computable iff it is strongly normalizable.
7. (vii) Let $f : \overline{\sigma} \to \tau \in \mathcal{F}$ and $\overline{s} :_C \overline{\sigma}$ a vector of computable terms. Then $f(\overline{s})$ is computable.

Computability Properties

1. (i) Every computable term is SN;
2. (ii) Assuming that s is computable and $s \succeq t$, then t is computable;
3. (iii) A neutral term s is computable iff t is computable for every t such that $s \succ t$;
4. (iv) If $\bar{t}$ be a vector of computable terms s.t. $@(\bar{t})$ is a candidate, then $@(\bar{t})$ is computable;
5. (v) $\lambda x : \sigma.u$ is computable iff $u\{x \mapsto w\}$ is computable for every computable $w :_C \sigma$;
6. (vi) Let $s :_C \sigma \in \mathcal{T}_S^{min}$. Then s is computable iff it is strongly normalizable.
7. (vii) Let $f : \overline{\sigma} \to \tau \in \mathcal{F}$ and $\overline{s} :_C \overline{\sigma}$ a vector of computable terms. Then $f(\overline{s})$ is computable.

Computability Properties

1. (i) Every computable term is SN;
2. (ii) Assuming that s is computable and $s \succeq t$, then t is computable;
3. (iii) A neutral term s is computable iff t is computable for every t such that $s \succ t$;
4. (iv) If $\bar{t}$ be a vector of computable terms s.t. $@(\bar{t})$ is a candidate, then $@(\bar{t})$ is computable;
5. (v) $\lambda x : \sigma.u$ is computable iff $u\{x \mapsto w\}$ is computable for every computable $w :_C \sigma$;
6. (vi) Let $s :_C \sigma \in \mathcal{T}_S^{min}$. Then s is computable iff it is strongly normalizable.
7. (vii) Let $f : \overline{\sigma} \rightarrow \tau \in \mathcal{F}$ and $\overline{s} :_C \overline{\sigma}$ a vector of computable terms. Then $f(\overline{s})$ is computable.

Computability Properties

1. (i) Every computable term is SN;
2. (ii) Assuming that s is computable and $s \succeq t$, then t is computable;
3. (iii) A neutral term s is computable iff t is computable for every t such that $s \succ t$;
4. (iv) If $\bar{t}$ be a vector of computable terms s.t. $@(\bar{t})$ is a candidate, then $@(\bar{t})$ is computable;
5. (v) $\lambda x : \sigma.u$ is computable iff $u\{x \mapsto w\}$ is computable for every computable $w :_C \sigma$;
6. (vi) Let $s :_C \sigma \in \mathcal{T}_S^{min}$. Then s is computable iff it is strongly normalizable.
7. (vii) Let $f : \overline{\sigma} \rightarrow \tau \in \mathcal{F}$ and $\overline{s} :_C \overline{\sigma}$ a vector of computable terms. Then $f(\overline{s})$ is computable.

Computability Properties

1. (i) Every computable term is SN;
2. (ii) Assuming that s is computable and $s \succeq t$, then t is computable;
3. (iii) A neutral term s is computable iff t is computable for every t such that $s \succ t$;
4. (iv) If $\bar{t}$ be a vector of computable terms s.t. $@(\bar{t})$ is a candidate, then $@(\bar{t})$ is computable;
5. (v) $\lambda x : \sigma.u$ is computable iff $u\{x \mapsto w\}$ is computable for every computable $w :_C \sigma$;
6. (vi) Let $s :_C \sigma \in \mathcal{T}_S^{min}$. Then s is computable iff it is strongly normalizable.
7. (vii) Let $f : \overline{\sigma} \to \tau \in \mathcal{F}$ and $\overline{s} :_C \overline{\sigma}$ a vector of computable terms. Then $f(\overline{s})$ is computable.

Computability Properties

1. (i) Every computable term is SN;
2. (ii) Assuming that s is computable and $s \succeq t$, then t is computable;
3. (iii) A neutral term s is computable iff t is computable for every t such that $s \succ t$;
4. (iv) If $\bar{t}$ be a vector of computable terms s.t. $@(\bar{t})$ is a candidate, then $@(\bar{t})$ is computable;
5. (v) $\lambda x : \sigma.u$ is computable iff $u\{x \mapsto w\}$ is computable for every computable $w :_{C} \sigma$;
6. (vi) Let $s :_{C} \sigma \in \mathcal{T}_{\mathcal{S}}^{min}$. Then s is computable iff it is strongly normalizable.
7. (vii) Let $f : \overline{\sigma} \longrightarrow \tau \in \mathcal{F}$ and $\overline{s} :_{C} \overline{\sigma}$ a vector of computable terms. Then $f(\overline{s})$ is computable.

Computability Properties

1. (i) Every computable term is SN;
2. (ii) Assuming that s is computable and $s \succeq t$, then t is computable;
3. (iii) A neutral term s is computable iff t is computable for every t such that $s \succ t$;
4. (iv) If $\bar{t}$ be a vector of computable terms s.t. $@(\bar{t})$ is a candidate, then $@(\bar{t})$ is computable;
5. (v) $\lambda x : \sigma.u$ is computable iff $u\{x \mapsto w\}$ is computable for every computable $w :_C \sigma$;
6. (vi) Let $s :_C \sigma \in \mathcal{T}_{\mathcal{S}}^{min}$. Then s is computable iff it is strongly normalizable.
7. (vii) Let $f : \overline{\sigma} \to \tau \in \mathcal{F}$ and $\overline{s} :_C \overline{\sigma}$ a vector of computable terms. Then $f(\overline{s})$ is computable.

Lemme principal et Preuve du théorème

Lemma

Let γ be a computable substitution and t be an algebraic λ-term. Then $t\gamma$ is computable.

We can now show that HORPO is SN:

Proof.

Given an arbitrary term t, let γ be the identity substitution. Since γ is computable, $t = t\gamma$ is computable by Lemma 4, and strongly normalizable by Property 1 (i). $\square$

We are left with the proof of the lemma, which proceeds by induction on the size of t.

Preuve du lemme principal

1. $t \in \mathcal{X}$. $t\gamma$ is computable by assumption.
2. $t \doteq \lambda x.u$. By Property (v), $t\gamma$ is computable if so is $u\gamma\{x \mapsto w\}$ for every computable w. Since x may not occur in γ, we define $\delta \doteq \gamma \cup \{x \mapsto w\}$. Then $u\gamma\{x \mapsto w\} = u(\gamma \cup \{x \mapsto w\})$. As δ is computable and $|t| > |u|$, $u\delta$ is computable by induction hypothesis.
3. $t \doteq @(t_1, t_2)$. Then $t_1\gamma$ and $t_2\gamma$ are computable by induction hypothesis, hence t is computable by Property (iv).
4. $t \doteq f(t_1, \ldots, t_n)$. Then $t_i\gamma$ is computable by induction hypothesis, hence $t\gamma$ is computable by property (vii).

Preuve du lemme principal

1. $t \in \mathcal{X}$. $t\gamma$ is computable by assumption.
2. $t \doteq \lambda x.u$. By Property (v), $t\gamma$ is computable if so is $u\gamma\{x \mapsto w\}$ for every computable w. Since x may not occur in γ, we define $\delta \doteq \gamma \cup \{x \mapsto w\}$. Then $u\gamma\{x \mapsto w\} = u(\gamma \cup \{x \mapsto w\})$. As δ is computable and $|t| > |u|$, $u\delta$ is computable by induction hypothesis.
3. $t \doteq @(t_1, t_2)$. Then $t_1\gamma$ and $t_2\gamma$ are computable by induction hypothesis, hence t is computable by Property (iv).
4. $t \doteq f(t_1, \ldots, t_n)$. Then $t_i\gamma$ is computable by induction hypothesis, hence $t\gamma$ is computable by property (vii).

Preuve du lemme principal

1. $t \in \mathcal{X}$. $t\gamma$ is computable by assumption.
2. $t \doteq \lambda x.u$. By Property (v), $t\gamma$ is computable if so is $u\gamma\{x \mapsto w\}$ for every computable w. Since x may not occur in γ, we define $\delta \doteq \gamma \cup \{x \mapsto w\}$. Then $u\gamma\{x \mapsto w\} = u(\gamma \cup \{x \mapsto w\})$. As δ is computable and $|t| > |u|$, $u\delta$ is computable by induction hypothesis.
3. $t \doteq @(t_1, t_2)$. Then $t_1\gamma$ and $t_2\gamma$ are computable by induction hypothesis, hence t is computable by Property (iv).
4. $t \doteq f(t_1, \ldots, t_n)$. Then $t_i\gamma$ is computable by induction hypothesis, hence $t\gamma$ is computable by property (vii).

Preuve du lemme principal

1. $t \in \mathcal{X}$. $t\gamma$ is computable by assumption.
2. $t \doteq \lambda x.u$. By Property (v), $t\gamma$ is computable if so is $u\gamma\{x \mapsto w\}$ for every computable w. Since x may not occur in γ, we define $\delta \doteq \gamma \cup \{x \mapsto w\}$. Then $u\gamma\{x \mapsto w\} = u(\gamma \cup \{x \mapsto w\})$. As δ is computable and $|t| > |u|$, $u\delta$ is computable by induction hypothesis.
3. $t \doteq @(t_1, t_2)$. Then $t_1\gamma$ and $t_2\gamma$ are computable by induction hypothesis, hence t is computable by Property (iv).
4. $t \doteq f(t_1, \ldots, t_n)$. Then $t_i\gamma$ is computable by induction hypothesis, hence $t\gamma$ is computable by property (vii).

Propriétés (i,ii,iii)

Note first that the only if part of property (iii) is property (ii). We are left with (i), (ii) and the if part of (iii), which are together spelled out as follows:

Given a type σ, a term $s :_C \sigma \in [\![\sigma]\!]$, a term $t :_C \tau$ such that $s \succ t$, and a neutral term $u :_C \sigma$ such that $w :_C \theta \in [\![\theta]\!]$ for every w such that $u \succ w$, we prove by induction on the definition of $[\![\sigma]\!]$ that
(i) s is strongly normalizable,
(ii) t is computable,
(iii) u is computable.

Propriétés (i,ii,iii)

Note first that the only if part of property (iii) is property (ii). We are left with (i), (ii) and the if part of (iii), which are together spelled out as follows:

Given a type σ, a term $s :_C \sigma \in [\![\sigma]\!]$, a term $t :_C \tau$ such that $s \succ t$, and a neutral term $u :_C \sigma$ such that $w :_C \theta \in [\![\theta]\!]$ for every w such that $u \succ w$, we prove by induction on the definition of $[\![\sigma]\!]$ that
(i) s is strongly normalizable,
(ii) t is computable,
(iii) u is computable.

σ is a data type

(i) All reducts of s are computable by definition of the interpretations, hence strongly normalizable by induction hypothesis, and therefore, so is s.

(ii) By definition of the candidate interpretations.

(iii) By definition of the candidate interpretations.

σ is a data type

(i) All reducts of s are computable by definition of the interpretations, hence strongly normalizable by induction hypothesis, and therefore, so is s.

(ii) By definition of the candidate interpretations.

(iii) By definition of the candidate interpretations.

σ is a data type

(i) All reducts of s are computable by definition of the interpretations, hence strongly normalizable by induction hypothesis, and therefore, so is s.

(ii) By definition of the candidate interpretations.

(iii) By definition of the candidate interpretations.

σ is a functional type. Property (i)

Let $s :_C \sigma = \theta_0 \succ s_1 :_C \theta_1 \ldots \succ s_n :_C \theta_n \succ \ldots$ be a derivation issuing from s. $s_n \in [\![\theta_n]\!]$ by repeated applications of property (ii) (and assumption for $n = 0$). Such derivations are of two kinds:

(a) $\sigma >_{\mathcal{T}_\mathcal{S}} \theta_n$ for some n. Then, s_n is strongly normalizable by induction hypothesis and the derivation issuing from s is finite;

(b) $\theta_n =_{\mathcal{T}_\mathcal{S}} \sigma$ for all n. The sequence of terms $@(s_n, y :_C \sigma_1) :_C \sigma_2 \to \ldots \sigma_n \to \tau$ is well-typed, and strictly decreasing by monotonicity. Since $\sigma \succ^{rpo}_{\mathcal{T}_\mathcal{S}\forall} \sigma_1$, $y :_C \sigma_1$ is computable by induction hypothesis (iii). By definition, $@(s_n, y)$ is computable and the sequence finite by induction hypothesis (i).

σ is a functional type. Property (i)

Let $s :_C \sigma = \theta_0 \succ s_1 :_C \theta_1 \ldots \succ s_n :_C \theta_n \succ \ldots$ be a derivation issuing from s. $s_n \in [\![\theta_n]\!]$ by repeated applications of property (ii) (and assumption for $n = 0$). Such derivations are of two kinds:

(a) $\sigma >_{\mathcal{T}_\mathcal{S}} \theta_n$ for some n. Then, s_n is strongly normalizable by induction hypothesis and the derivation issuing from s is finite;

(b) $\theta_n =_{\mathcal{T}_\mathcal{S}} \sigma$ for all n. The sequence of terms $@(s_n, y :_C \sigma_1) :_C \sigma_2 \rightarrow \ldots \sigma_n \rightarrow \tau$ is well-typed, and strictly decreasing by monotonicity. Since $\sigma \succ^{rpo}_{\mathcal{T}_{\mathcal{S}^\forall}} \sigma_1$, $y :_C \sigma_1$ is computable by induction hypothesis (iii). By definition, $@(s_n, y)$ is computable and the sequence finite by induction hypothesis (i).

Property (ii)

Let $\sigma = \theta \rightarrow \rho$. By arrow preservation and arrow decreasing properties, there are two cases:

(a) $\rho \geq_{\mathcal{T}_S} \tau$. Since s is computable, $@(s, u)$ is computable for every $u \in [\![\theta]\!]$. Let $y :_C \theta$. By induction hypothesis (iii), $y \in [\![\theta]\!]$, hence $@(u, y)$ is computable. Since $@(s, y) :_C \rho \succ t :_C \tau$ by case **Clôture**, t is computable induction hypothesis (ii).

(b) $\tau = \theta' \rightarrow \rho'$, with $\theta =_{\mathcal{T}_S} \theta'$ and $\rho \geq_{\mathcal{T}_S} \rho'$. Since s is computable, given $u \in [\![\theta]\!]$, then $@(s, u) \in [\![\rho]\!]$, hence, by induction hypothesis (ii) $@(t, u) \in [\![\rho']\!]$. Since $[\![\theta]\!] = [\![\theta']\!]$ by Lemma 1, $t \in [\![\tau]\!]$ by definition of $[\![\tau]\!]$.

Property (ii)

Let $\sigma = \theta \to \rho$. By arrow preservation and arrow decreasing properties, there are two cases:

(a) $\rho \geq_{\mathcal{T}_S} \tau$. Since s is computable, $@(s, u)$ is computable for every $u \in [\![\theta]\!]$. Let $y :_C \theta$. By induction hypothesis (iii), $y \in [\![\theta]\!]$, hence $@(u, y)$ is computable. Since $@(s, y) :_C \rho \succ t :_C \tau$ by case **Clôture**, t is computable induction hypothesis (ii).

(b) $\tau = \theta' \to \rho'$, with $\theta =_{\mathcal{T}_S} \theta'$ and $\rho \geq_{\mathcal{T}_S} \rho'$. Since s is computable, given $u \in [\![\theta]\!]$, then $@(s, u) \in [\![\rho]\!]$, hence, by induction hypothesis (ii) $@(t, u) \in [\![\rho']\!]$. Since $[\![\theta]\!] = [\![\theta']\!]$ by Lemma 1, $t \in [\![\tau]\!]$ by definition of $[\![\tau]\!]$.

Propriété (iii)

Let $\sigma = \sigma_1 \rightarrow \ldots \rightarrow \sigma_n \rightarrow \tau$, with $n > 0$ and τ a data type. t is computable iff $@(t, u_1, \ldots, u_n)$ is computable for arbitrary $u_1 \in [\![\sigma_1]\!], \ldots, u_n \in [\![\sigma_n]\!]$, hence SN by Property (i), and
$@(t, u_1, \ldots, u_n) :_C \tau$ is computable iff so are its reducts.
We prove by induction on multiset $\{u_1, \ldots, u_n\}$ of computable terms ordered by $(\succeq)_{mul}$ the property (H) stating that terms w strictly smaller than $@(t, u_1, \ldots, u_i)$ in $\succ$ are computable. The property corresponds to $i = n$. If $i = 0$, terms strictly smaller than t are computable by assumption. If $(i + 1) \leq n$, we consider all w strictly smaller than $@(@(t, u_1, \ldots, u_i), u_{i+1})$.

First case:

$@(@(t, u_1, \ldots, u_i), u_{i+1}) \succ w$ by Case **Clôture**.
There are again two possibilities:

- $@(t, u_1, \ldots, u_i) \succeq w$, and therefore $@(t, u_1, \ldots, u_i) \succ w$ for type reason since w is also a reduct of $@(t, u_1, \ldots, u_{i+1})$. We then conclude by induction hypothesis (H).
- $u_{i+1} \succeq w$. We conclude by assumption and induction property (ii).

First case:

$@(@(t, u_1, \ldots, u_i), u_{i+1}) \succ w$ by Case **Clôture**.
There are again two possibilities:

- $@(t, u_1, \ldots, u_i) \succeq w$, and therefore $@(t, u_1, \ldots, u_i) \succ w$ for type reason since w is also a reduct of $@(t, u_1, \ldots, u_{i+1})$. We then conclude by induction hypothesis (H).
- $u_{i+1} \succeq w$. We conclude by assumption and induction property (ii).

Second case:

$@(@(t, u_1, \ldots, u_i), u_{i+1}) \succ w \doteq @(\overline{w})$ by Case **Mul@**. By definition of multiset extension and for type reasons, there are two possibilities:

- for all $v \in \overline{w}$, either $@(t, u_1, \ldots, u_i) \succ v$, and v is computable by induction hypothesis (H), or $u_{i+1} \succeq v$, in which case v is computable by assumption and induction property (ii). It follows that w is computable by Property (iv).
- $w_1 = @(t, u_1, \ldots, u_i)$ and $u_{i+1} \succ w_2$, implying that w_2 is computable by assumption and induction property (ii). By induction property (H), all reducts of w are computable. Since w and t have the same (data) type, w is computable by induction property (iii).

Second case:

$@(@(t, u_1, \ldots, u_i), u_{i+1}) \succ w \doteq @(\overline{w})$ by Case **Mul@**. By definition of multiset extension and for type reasons, there are two possibilities:

- for all $v \in \overline{w}$, either $@(t, u_1, \ldots, u_i) \succ v$, and v is computable by induction hypothesis (H), or $u_{i+1} \succeq v$, in which case v is computable by assumption and induction property (ii). It follows that w is computable by Property (iv).
- $w_1 = @(t, u_1, \ldots, u_i)$ and $u_{i+1} \succ w_2$, implying that w_2 is computable by assumption and induction property (ii). By induction property (H), all reducts of w are computable. Since w and t have the same (data) type, w is computable by induction property (iii).

Propriété (v)

The only if part follows from property (ii) and the definition of $[\![]\!]$. Assume conversely that $u\{x \mapsto s\}$ is computable for an arbitrary computable s. We show that $@(\lambda x.u, w)$ is computable for an arbitrary computable w. Since variables are computable by property (iii), $u = u\{x \mapsto x\}$ is computable by assumption. By property (i), u and w are SN. Since $\lambda x.u \succeq v$ implies $u \succeq v$ or $v = \lambda x.u'$ and $u \succeq u'$, $\lambda x.u$ is SN by induction on u. We prove that $@(\lambda x.u, w)$ is computable by induction on the pair $\{\lambda x.u, w\}$ ordered by $(\succ)_{lex}$. By property (iii), the neutral term $@(\lambda x.u, w)$ is computable iff so are all v s.t. $@(\lambda x.u, w) \succ v$.

Cas **Clôture**

1. If $w \succeq v$, we conclude by property (ii).
2. If $\lambda x.u \succ v$, there are two cases. If $u \succeq v$ by case **Clôture**, we conclude by property (ii) again. Otherwise, $v = \lambda x.u'$ and $u \succ u'$, hence $u\{x \mapsto w\} \succ u'\{x \mapsto w\}$ by stability property. By assumption and property (ii), $u'\{x \mapsto w\}$ is therefore computable. Hence, $@(v, w)$ is computable by induction hypothesis applied to the pair $(v = \lambda x.u', w)$. We then conclude by definition of the interpretations that v is computable.

Cas Clôture

1. If $w \succeq v$, we conclude by property (ii).
2. If $\lambda x.u \succ v$, there are two cases. If $u \succeq v$ by case **Clôture**, we conclude by property (ii) again. Otherwise, $v = \lambda x.u'$ and $u \succ u'$, hence $u\{x \mapsto w\} \succ u'\{x \mapsto w\}$ by stability property. By assumption and property (ii), $u'\{x \mapsto w\}$ is therefore computable. Hence, $@(v, w)$ is computable by induction hypothesis applied to the pair $(v = \lambda x.u', w)$. We then conclude by definition of the interpretations that v is computable.

Cas **Reduction** et **Mul@**

1. If $@(\lambda x.u, w) \succ v$ by case **Reduction**, then $u\{x \mapsto w\} \succeq v$. By assumption, $u\{x \mapsto w\}$ is computable, and hence v is computable by property (ii).
2. If the comparison is by case **Mul@**, then $v = @(\overline{v})$ and all terms in $\{\overline{v}\}$ are smaller than w or $\lambda x.u$. There are two cases.

Cas **Reduction** et **Mul@**

1. If $@(\lambda x.u, w) \succ v$ by case **Reduction**, then $u\{x \mapsto w\} \succeq v$. By assumption, $u\{x \mapsto w\}$ is computable, and hence v is computable by property (ii).
2. If the comparison is by case **Mul@**, then $v = @(\overline{v})$ and all terms in $\{\overline{v}\}$ are smaller than w or $\lambda x.u$. There are two cases.

Cas **Mul@**, suite

1. $v_1 = \lambda x.u$ and $w \succ v_i$ for $i > 1$. Then v_i is computable by property (ii) and, since $u\{x \mapsto v_2\}$ is computable by the main assumption, $@(v_1, v_2)$ is computable by induction hypothesis. If $v = @(v_1, v_2)$, we are done, and we conclude by bqasic property (ii) otherwise.
2. For all other cases, terms in $\overline{v}$ are reducts of $\lambda x.u$ and w. Reducts of w and reducts of $\lambda x.u$ which are themselves reducts of u are computable by property (ii).

Cas **Mul@**, suite

1. $v_1 = \lambda x.u$ and $w \succ v_i$ for $i > 1$. Then v_i is computable by property (ii) and, since $u\{x \mapsto v_2\}$ is computable by the main assumption, $@(v_1, v_2)$ is computable by induction hypothesis. If $v = @(v_1, v_2)$, we are done, and we conclude by bqasic property (ii) otherwise.
2. For all other cases, terms in $\overline{v}$ are reducts of $\lambda x.u$ and w. Reducts of w and reducts of $\lambda x.u$ which are themselves reducts of u are computable by property (ii).

Cas **Mul@**, fin

1. If all terms in $\overline{v}$ are such reducts, v is computable by basic property (ii).
2. Otherwise, for typing reason, v_1 is a reduct of $\lambda x.u$ of the form $\lambda x.u'$ with $u \succ u'$, and all other terms in $\overline{v}$ are reducts of the previous kind. By the main assumption, $u\{x \mapsto v''\}$ is computable. Besides, $u\{x \mapsto v''\} \succ u'\{x \mapsto v''\}$ by stability property of the ordering. Therefore $u'\{x \mapsto v''\}$ is computable by Property (ii). By induction hypothesis, $@(v_1, v_2)$ is again computable. If $v = @(v_1, v_2)$, we are done, otherwise v is computable by basic property (ii).

Cas **Mul@**, fin

1. If all terms in $\overline{v}$ are such reducts, v is computable by basic property (ii).
2. Otherwise, for typing reason, v_1 is a reduct of $\lambda x.u$ of the form $\lambda x.u'$ with $u \succ u'$, and all other terms in $\overline{v}$ are reducts of the previous kind. By the main assumption, $u\{x \mapsto v''\}$ is computable. Besides, $u\{x \mapsto v''\} \succ u'\{x \mapsto v''\}$ by stability property of the ordering. Therefore $u'\{x \mapsto v''\}$ is computable by Property (ii). By induction hypothesis, $@(v_1, v_2)$ is again computable. If $v = @(v_1, v_2)$, we are done, otherwise v is computable by basic property (ii).

Propriété (vi)

The only if direction is property (i). For the if direction, let s be a strongly normalizable term of type $\sigma \in \mathcal{T}_{\mathcal{S}}^{min}$. We prove that s is computable by induction on $\succ$. Since σ is a data type, s must be neutral. Let now $s \succ t :_C \tau$, hence $\sigma \geq_{\mathcal{T}_{\mathcal{S}}} \tau$. By definition of $\mathcal{T}_{\mathcal{S}}^{min}$, $\tau =_{\mathcal{T}_{\mathcal{S}}} \sigma$, hence, τ is a data type by definition of $=_{\mathcal{T}_{\mathcal{S}}}$, and since σ is minimal, so is τ, hence $\tau \in \mathcal{T}_{\mathcal{S}}^{min}$. By assumption on s, t must be strongly normalizable, and by induction hypothesis, it is therefore computable. Since this is true of all reducts of s, by definition s is computable.

Preuve de (vii)

Since terms in $\overline{s}$ are computable, by Property (i), they are strongly normalizable. We use this remark to build our induction argument: we prove that $f(\overline{s})$ is computable by induction on the pair $(f, \overline{s})$ ordered lexicographically by $(>_{\mathcal{F}}, (\succ)_{stat_f})_{lex}$. Since $f(\overline{s})$ is neutral, by

Property (iii), it is computable iff every t such that $f(\overline{s}) \succ t$ is computable, which we prove by an inner induction on the size of t. We discuss by cases according to the definition of $\succ$.

Preuve de (vii)

Since terms in $\overline{s}$ are computable, by Property (i), they are strongly normalizable. We use this remark to build our induction argument: we prove that $f(\overline{s})$ is computable by induction on the pair $(f, \overline{s})$ ordered lexicographically by $(>_{\mathcal{F}}, (\succ)_{stat_f})_{lex}$. Since $f(\overline{s})$ is neutral, by

Property (iii), it is computable iff every t such that $f(\overline{s}) \succ t$ is computable, which we prove by an inner induction on the size of t. We discuss by cases according to the definition of $\succ$.

Cas **Clôture** et **Prec**

Let $f(\overline{s}) \succ t$ by case **Clôture**, hence $s_i \succeq t$ for some $s_i \in \overline{s}$. Since s_i is computable, t is computable by Property (ii).

Let $s = f(\overline{s}) \succ t$ by case **Prec**. Then $t = g(\overline{t})$, $f >_{\mathcal{F}} g$ and for every $v \in \overline{t}$ either $s \succ v$, in which case v is computable by the inner induction hypothesis, or $u \succeq v$ for some $u \in \overline{s}$ and v is computable by Property (ii). Therefore, $\overline{t}$ is computable, and since $f >_{\mathcal{F}} g$, t is computable by the outer induction hypothesis.

Cas **Clôture** et **Prec**

Let $f(\overline{s}) \succ t$ by case **Clôture**, hence $s_i \succeq t$ for some $s_i \in \overline{s}$. Since s_i is computable, t is computable by Property (ii).

Let $s = f(\overline{s}) \succ t$ by case **Prec**. Then $t = g(\overline{t})$, $f >_{\mathcal{F}} g$ and for every $v \in \overline{t}$ either $s \succ v$, in which case v is computable by the inner induction hypothesis, or $u \succeq v$ for some $u \in \overline{s}$ and v is computable by Property (ii). Therefore, $\overline{t}$ is computable, and since $f >_{\mathcal{F}} g$, t is computable by the outer induction hypothesis.

Cas **Mul** et **Lex**

If $f(\overline{s}) \succ t$ by case **Mul**, then $t = g(\overline{t})$, $f =_{\mathcal{F}} g$, and $\overline{s}(\succ)_{mul}\overline{t}$. By definition of the multiset comparison, for every $t_i \in \overline{t}$ there is some $s_j \in \overline{s}$, s.t. $s_j \succeq t_i$, hence, by Property (ii), t_i is computable. This allows us to conclude by the outer induction hypothesis that t is computable.

If $f(\overline{s}) \succ t$ by case **Lex**, then $t = g(\overline{t})$, $f =_{\mathcal{F}} g$, $\overline{s}(\succ)_{lex}\overline{t}$ and for every $v \in \overline{t}$ either $f(\overline{s}) \succ v$ or $u \succeq v$ for some $u \in \overline{s}$. As in the precedence case, this implies that $\overline{t}$ is computable. Then, since $\overline{s}(\succ)_{lex}\overline{t}$, t is computable by the outer induction hypothesis.

Cas **Mul** et **Lex**

If $f(\overline{s}) \succ t$ by case **Mul**, then $t = g(\overline{t})$, $f =_{\mathcal{F}} g$, and $\overline{s}(\succ)_{mul}\overline{t}$. By definition of the multiset comparison, for every $t_i \in \overline{t}$ there is some $s_j \in \overline{s}$, s.t. $s_j \succeq t_i$, hence, by Property (ii), t_i is computable. This allows us to conclude by the outer induction hypothesis that t is computable.

If $f(\overline{s}) \succ t$ by case **Lex**, then $t = g(\overline{t})$, $f =_{\mathcal{F}} g$, $\overline{s}(\succ)_{lex}\overline{t}$ and for every $v \in \overline{t}$ either $f(\overline{s}) \succ v$ or $u \succeq v$ for some $u \in \overline{s}$. As in the precedence case, this implies that $\overline{t}$ is computable. Then, since $\overline{s}(\succ)_{lex}\overline{t}$, t is computable by the outer induction hypothesis.

Cas **Prec@** et **Prec**λ

If $f(\overline{s}) \succ t$ by case **Prec@**, let $@(t_1, \ldots, t_n)$ be the partial left-flattening of t used in that proof. By the same token as in case **Prec**, every term in $\overline{t}$ is computable, hence t is computable by Property (iv).

If $f(\overline{s}) \succ t$ by case **Prec**λ, then $t = \lambda x.u$ with $x \notin \mathcal{V}ar(u)$, and $f(\overline{s}) \succ u$. By the inner induction hypothesis, u is computable. Hence, $u\{x \mapsto w\} = u$ is computable for any computable w, and therefore, $t = \lambda x.u$ is computable by Property (v).

Cas **Prec@** et **Prec**λ

If $f(\overline{s}) \succ t$ by case **Prec@**, let $@(t_1, \ldots, t_n)$ be the partial left-flattening of t used in that proof. By the same token as in case **Prec**, every term in $\overline{t}$ is computable, hence t is computable by Property (iv).

If $f(\overline{s}) \succ t$ by case **Prec**λ, then $t = \lambda x.u$ with $x \notin \mathcal{V}ar(u)$, and $f(\overline{s}) \succ u$. By the inner induction hypothesis, u is computable. Hence, $u\{x \mapsto w\} = u$ is computable for any computable w, and therefore, $t = \lambda x.u$ is computable by Property (v).

Clôture calculable

Le rôle de la clôture calculable $\mathcal{CC}(s, t)$ du terme s dans les preuves (t joue un rôle très marginal) est de collecter des termes réductibles au sens de Tait-Girard. Nous allons donc profiter des propriétés de réductibilité pour construire une définition inductive de cette clôture (l' ancienne définition sera le cas de base) de manière à accroître son expressivité et utiliser le cas **Clôture** de la définition du HORPO de manière plus systématique.