Rapport de DEA - Paris, 18 Septembre 1998

The Calculus of Algebraic and Inductive Constructions

Frédéric Blanqui

Responsables: Jean-Pierre Jouannaud (LRI) et Maribel Fernández (ENS)
Lieu: Laboratoire de Recherche en Informatique (Université Paris XI - Orsay)

DEA Sémantique, Preuve et Programmation - Paris VII

1 Introduction

This work is part of a long-term effort started by the DEMONS team at the Laboratoire de Recherche en Informatique (Université Paris-Sud) and the Coq team at INRIA-Rocquencourt.

Each team has developed a different approach to specification languages and adapted tools: rewriting and CiME for the first, constructive higher-order logic and Coq for the second.

Each approach has important advantages, but also some disadvantages. Hence, the idea has emerged to put together the efforts of both teams in order to build a system that would enjoy the advantages of both approaches in a safe and coherent way.

Coq is an implementation of the calculus of inductive constructions. As any proof-assistant, it offers a set of powerful automated tactics, that the user can extend at will. A first advantage is that the specifications are written in a declarative style, freeing the user from thinking in operational terms. Secondly, although the system relies on a small kernel which can easily be proved correct, making the whole system a safe one: each resulting proof is typed-checked, hence has to be correct. Thirdly, whatever complex are the proofs, it is always possible to extract from them an executable code that is correct with respect to the specification.

In return, a first disadvantage is that it is difficult to integrate decision procedures. Secondly, the authorized data types are too restrictive: on the one hand, the constructors must be free, and on the other hand, the functions must be defined by higher-order primitive recursion which is not really user-friendly. Thirdly, the system does not yet offer a module system for the management of theories and proofs.

For the rewriting approach to specification languages, the tools are mainly automatic; this is a first advantage. A second is that it is possible to execute a specification, allowing the user to perform experiments. Thirdly, it is easy to integrate decision procedures expressed by a convergent set of rewrite rules.

In return, a first disadvantage is that this approach forces the user to think in operational terms. Secondly, as the system is easily extensible by decision procedures, the user has to check meta-theoretic properties to ensure the correctness of a specification. Thirdly, the tools developed for this approach are often too automatic, hence do not allow useful interventions from the user, necessary by the complexity of proofs.

A first solution to extend Coq with decision procedures is to encode the results of specialized tools by Coq-terms that can be checked by the kernel. Another solution is to code the decision procedure by a tactic. Both approaches require an important implementation work (typically, several months), and result in proofs whose size may not be manageable.

Another approach is to abandon the idea that every proof must be entirely checked by the kernel, and to trust specialized tools by delegating them the checking of some sub-proofs, e.g. some equational sub-proofs.

Our work addresses two questions, namely, the integration of decision procedures defined by sets of rewrite rules, and the definition of a more powerful notion of inductive type, in which functions can be defined by sets of rewrite rules (pattern matching definitions). As can be expected, the same technical novelty serves both purposes. The result is the Calculus of Algebraic and Inductive Constructions, which merges together, in a coherent way, the calculus of constructions, inductive types, first-order and higher-order rewriting.

2 The Calculus of Algebraic and Inductive Constructions

The aim of this section is to present the *Calculus of Algebraic and Inductive Constructions* (CAIC), an extension of the Calculus of Constructions (a typed λ-calculus with polymorphism, dependent types and type constructors) [CH88] with strictly positive inductive types, uncurried function symbols and rewrite rules (either first-order or higher-order).

It is an extension of the Calculus of Algebraic Constructions of Barbanera, Fernández and Geuvers [BFG94] based on the recent work of Jouannaud and Okada [JO97b] for the strong normalization and confluence of an algebraic simply typed λ-calculus with positive inductive types (not necessarily strict).

Improvements in comparison to [BFG94] are the following:

· Inductive types and constructors are made available.
· Abstractions can occur in the rewriting rules (either in the lefthand side or in the righthand side).
· In higher-order rewrite rules, recursive calls can be compared through a combination of multi-set and lexicographic orderings instead of just a multi-set ordering.
· An adpated version of the new "General schema" of Jouannaud and Okada [JO97b] catches the recursor rules of any strictly positive inductive type.
· For the λ-calculus part, we use a much shorter and simpler strong normalization proof inspired from Geuvers [Geu95].
· For the reducibility of higher-order function symbols, we simplify and improve the proof of Jouannaud and Okada [JO97b].

Definition 2.1 (Algebraic types) Given a set $\mathcal{S}$ of *sorts*, the set $\mathcal{T}_\mathcal{S}$ of *algebraic types* is inductively defined by the following grammar rule:

$$s := \mathtt{s} \mid (s \rightarrow s)$$

where $\mathtt{s}$ ranges over $\mathcal{S}$. $\rightarrow$ associates to the right such that $s_1 \rightarrow (s_2 \rightarrow s_3)$ can be written as $s_1 \rightarrow s_2 \rightarrow s_3$. An algebraic type $s_1 \rightarrow \ldots \rightarrow s_n$ is *first-order* if $s_i \in \mathcal{S}$ $(1 \le i \le n)$, otherwise it is *higher-order*. We denote by $\mathcal{T}^n_\mathcal{S}$ $(n \ge 0)$ the set of algebraic types of the form $s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s$.

Definition 2.2 (Extended algebraic types) Given an infinite set $\mathit{Var}^\Box$ of variables, the set of *extended algebraic types* is inductively defined by the following grammar rule:

$$s := \alpha \mid \mathtt{s} \mid (s \rightarrow s)$$

where $\mathtt{s}$ ranges over $\mathcal{S}$ and α over $\mathit{Var}^\Box$. As for algebraic types, $\rightarrow$ associates to the right.

Definition 2.3 (Constructors) We assume that each sort $\mathtt{s}$ has an associated set $\mathcal{C}(\mathtt{s})$ of *constructors*. Each constructor C is equipped with an algebraic type $\tau(C)$ of the form $s_1 \rightarrow \ldots \rightarrow s_n \rightarrow \mathtt{s}$ $(n \ge 0)$: n is called its *arity* and $\mathtt{s}$ its *output type*. We denote by $\mathcal{C}^n$ $(n \ge 0)$ the set of constructors of arity n.

A constructor C is *first-order* if its type is first-order, otherwise it is *higher-order*.

Here are some familiar examples of sorts:
· the sort `bool` of booleans whose constructors are `true`:`bool` and `false`:`bool`
· the sort `nat` of natural numbers whose constructors are $\mathtt{0}_\mathtt{nat}$:`nat` and $\mathtt{s}_\mathtt{nat}$:$\mathtt{nat} \rightarrow \mathtt{nat}$
· the sort $\mathtt{list}_t$ of lists of elements of an algebraic type t whose constructors are $\mathtt{nil}_t$:$\mathtt{list}_t$ and $\mathtt{cons}_t$: $t \rightarrow \mathtt{list}_t \rightarrow \mathtt{list}_t$,
· the sort `ord` of ordinals whose constructors are $\mathtt{0}_\mathtt{ord}$:`ord`, $\mathtt{s}_\mathtt{ord}$:$\mathtt{ord} \rightarrow \mathtt{ord}$ and $\mathtt{lim}_\mathtt{ord}$:$(\mathtt{nat} \rightarrow \mathtt{ord}) \rightarrow \mathtt{ord}$
· the sort `mon` of Montague' semantical objects whose constructors are $\mathtt{0}_\mathtt{mon}$:`mon` and $\mathtt{C}_\mathtt{mon}$:$((\mathtt{mon} \rightarrow \mathtt{bool}) \rightarrow \mathtt{mon}) \rightarrow \mathtt{mon}$.

Definition 2.4 (Sort ordering) We define the following quasi-ordering on sorts: $\mathtt{s} \ge_\mathcal{S} \mathtt{t}$ if and only if $\mathtt{t}$ occurs in the type of a constructor belonging to $\mathcal{C}(\mathtt{s})$.

Definition 2.5 (Function symbols) Given an algebraic type $t = s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s$ $(n \ge 0)$, we denote by $\mathcal{F}^n_t$ the set of *function symbols* of *arity* n, of *type* $\tau(f) = t$ and of *output type* s. We denote by $\mathcal{F}$ the set of all the function symbols, usually called the user' signature.

Function symbols with a first-order (resp. higher-order) type are called *first-order* (resp. *higher-order*). We denote by $\mathcal{F}_1$ (resp. $\mathcal{F}_\omega$) the set of first-order (resp. higher-order) function symbols.

Here are some familiar examples of functions:
· the function $\mathtt{if}_t$ of arity 3 and type $\mathtt{bool} \rightarrow t \rightarrow t \rightarrow t$,
· the function $\mathtt{+}_\mathtt{nat}$ of arity 2 and type $\mathtt{nat} \rightarrow \mathtt{nat} \rightarrow \mathtt{nat}$,
· the functions $\mathtt{0}_\mathtt{int}$ of arity 0 and type `int`, $\mathtt{s}_\mathtt{int}$ of arity 1 and type $\mathtt{int} \rightarrow \mathtt{int}$, $\mathtt{p}_\mathtt{int}$ of arity 1 and type $\mathtt{int} \rightarrow \mathtt{int}$, and $\mathtt{+}_\mathtt{int}$ of arity 2 and type $\mathtt{int} \rightarrow \mathtt{int} \rightarrow \mathtt{int}$,
· the function $\mathtt{append}_t$ of arity 2 and type $\mathtt{list}_t \rightarrow \mathtt{list}_t \rightarrow \mathtt{list}_t$,
· the function $\mathtt{map}_{t,t'}$ of arity 2 and type $(t \rightarrow t') \rightarrow \mathtt{list}_t \rightarrow \mathtt{list}_{t'}$.

Definition 2.6 (Terms) The set *Term* of the *terms* of CAIC is inductively defined by the following grammar rule:

$$a := x \mid \mathtt{s} \mid \star \mid \Box \mid \lambda x{:}a.a \mid \Pi x{:}a.a \mid (a\,a) \mid C(a_1,\ldots,a_n) \mid f(a_1,\ldots,a_n)$$

where $\mathtt{s}$ ranges over $\mathcal{S}$, C over $\mathcal{C}^n$ ($n \geq 0$), f over $\mathcal{F}^n_t$ ($t \in \mathcal{T}^n_{\mathcal{S}}, n \geq 0$) and x ranges over an infinite set *Var* of variables made of two disjoint infinite sets, $\mathit{Var}^{\Box}$ and $\mathit{Var}^{\star}$. The application $(a\,b)$ associates to the left such that $(a_1\,a_2)\,a_3$ can be written $a_1\,a_2\,a_3$.

A sequence of terms $(a_1,\ldots,a_n)$ or $a_1\ldots a_n$ ($n \geq 0$) is denoted by a vector $\vec{a}$ whose length is $|\vec{a}| = n$. A term $C(\vec{a})$ (resp. $f(\vec{a})$) is said to be *constructed headed* (resp. *function headed*).

We inductively define the set $\mathit{Pos}(a)$ of the *positions* in a term a as a language on the alphabet $\{1, 2, \ldots\}$ as follows:

- $\mathit{Pos}(x) = \mathit{Pos}(s) = \mathit{Pos}(\star) = \mathit{Pos}(\Box) = \{\epsilon\}$
- $\mathit{Pos}(\lambda x{:}a.b) = \mathit{Pos}(\Pi x{:}a.b) = \mathit{Pos}(a\,b) = \{\epsilon\} \cup \{1\}\cdot \mathit{Pos}(a) \cup \{2\}\cdot \mathit{Pos}(b)$
- $\mathit{Pos}(C(a_1,\ldots,a_n)) = \mathit{Pos}(f(a_1,\ldots,a_n)) = \{\epsilon\} \cup \{1\}\cdot \mathit{Pos}(a_1) \cup \ldots \cup \{n\}\cdot \mathit{Pos}(a_n)$

where ϵ denotes the empty word and $\cdot$ the concatenation. The *subterm* of a term a at position $p \in \mathit{Pos}(a)$ is denoted by $a_{|m}$ and the term obtained by replacing it by a term b is denoted by $a[b]_m$.

Bound and free variables are defined as usual. $\mathit{Var}(a)$ denotes the set of variables occurring in a, while $\mathit{FV}(a)$ (resp. $\mathit{BV}(a)$) denotes that of free (resp. bound) variables.

$a\{\vec{x} \mapsto \vec{b}\}$ denotes the simultaneous *substitution* of the terms b_i for the variables x_i in the term a ($0 \leq i \leq |\vec{x}| = |\vec{b}|$). Substitutions are denoted by $\theta, \theta', \ldots$, and are written in postfixed notation. $\theta \cup \{x \mapsto a\}$ denotes the substitution θ' such that $x\theta' = a$ and $y\theta' = y\theta$ if $y \neq x$. The domain of a substitution θ is $\mathit{dom}(\theta) = \{x \in \mathit{Var} \mid x\theta \neq x\}$.

As in the untyped λ-calculus, terms that only differ from each other in their bound variables will be identified, an operation called *α-conversion*. By convention, *bound and free variables will always be assumed different.*

Finally, we use the traditional abbreviations: $\lambda\vec{x}:\vec{a}.b$ where $|\vec{x}| = |\vec{a}|$ denotes b if $\vec{x}$ is an empty sequence, and the term $\lambda x_1{:}a_1.(\lambda x_2{:}a_2.(\ldots(\lambda x_n{:}a_n.b)\ldots))$ otherwise. $\Pi\vec{x}{:}\vec{a}.b$ where $|\vec{x}| = |\vec{a}|$ denotes b if $\vec{x}$ is an empty sequence, and the term $\Pi x_1{:}a_1.(\Pi x_2{:}a_2.(\ldots(\Pi x_n{:}a_n.b)\ldots))$ otherwise. If $|\vec{a}| = n$, $b\,\vec{a}$ denotes b if $\vec{a}$ is an empty sequence, and the term $(\ldots((b\,a_1)\,a_2)\ldots)\,a_n$ otherwise. We also write $a \to b$ for the term $\Pi x{:}a.b$ where $x \notin \mathit{FV}(b)$. This abbreviation allows us to see extended algebraic types as terms of our calculus.

Definition 2.7 (β-reduction relation) The *βh-rewrite relation* (β-head rewrite relation) on terms is defined as follows:

$$(\lambda x{:}a.b)\,c \rightsquigarrow_{\beta h} b\{x \mapsto c\}$$

The *β-rewrite relation* is its compatible closure:

$$a \rightsquigarrow_{\beta h} a' \;\Rightarrow\; a \rightsquigarrow_{\beta} a'$$
$$a \rightsquigarrow_{\beta} a',\; m \in \mathit{Pos}(b) \;\Rightarrow\; b[a]_m \rightsquigarrow_{\beta} b[a']_m$$

A term is in *βh-normal form (β-normal form)* if it cannot be βh-reduced (β-reduced).

The βh-reduction (β-reduction) relation is the reflexive and transitive closure of the βh-rewrite (β-rewrite) relation.

Lemma 2.8 (β-normal forms) A term a is in β-normal form if and only if it is of the form $\mu_1 x_1{:}b_1\ldots\mu_n x_n{:}b_n.c\,\vec{d}$ where $\mu_i \in \{\lambda, \Pi\}$ ($1 \leq i \leq n, n \geq 0$), terms in $\vec{b}$ and $\vec{d}$ are in β-normal form, and c is $\star$, $\Box$, a sort $\mathtt{s}$, a variable y, a constructor headed term $C(\vec{e})$ or a function headed term $f(\vec{e})$ where terms in $\vec{e}$ are in β-normal form.

Definition 2.9 (Algebraic terms) The set *Alg* of the *algebraic terms* is inductively defined by the following grammar rule:

$$a := x^{\star} \mid C(a_1,\ldots,a_n) \mid f(a_1,\ldots,a_n)$$

where $x^{\star}$ ranges over $\mathit{Var}^{\star}$, C over $\mathcal{C}^n$ ($n \geq 0$) and f over $\mathcal{F}^n_t$ ($t \in \mathcal{T}^n_{\mathcal{S}}, n \geq 0$).

An algebraic term is *first-order* if its function symbols and constructors are first-order, otherwise it is *higher-order*.

Definition 2.10 (Rule terms) The set RT of *rule terms* is inductively defined by the following grammar rule:

$$a := x^\star \mid \lambda x^\star{:}s.a \mid (a\,a) \mid C(a_1,\ldots,a_n) \mid f(a_1,\ldots,a_n)$$

where $x^\star$ ranges over $Var^\star$, s over $\mathcal{T}_\mathcal{S}$, C over $\mathcal{C}^n$ $(n \geq 0)$ and f over $\mathcal{F}^n_t$ $(t \in \mathcal{T}^n_\mathcal{S}, n \geq 0)$.

A rule term is *first-order* if it is a first-order *algebraic* term, otherwise it is *higher-order*.

Lemma 2.11 (β-normal forms of rule terms) A rule term a is in β-normal form if and only if it is of the form $\lambda\vec{x}{:}\vec{s}.c\,\vec{d}$ where terms in $\vec{d}$ are in β-normal form and c is a variable y, a constructor headed term $C(\vec{e})$ or a function headed term $f(\vec{e})$ where terms in $\vec{e}$ are in β-normal form.

Definition 2.12 (Rewrite rules) [1] A *rewrite rule* is a pair $l \rightsquigarrow r$ of rule terms such that l is headed by a function symbol and $FV(r) \subseteq FV(l)$. A function symbol is *defined* by a rewrite rule if it is the head function symbol of the lefthand side of that rule.

A term a rewrites to a term b at position $m \in Pos(a)$ with the rule $l \rightsquigarrow r$ if $a_{|m} = l\theta$ and $b = a[r\theta]_m$ for some substitution θ. Given a set R of rules, a term rewrites by R if it rewrites by one of the rules of R. Such a rewrite relation is denoted by $\rightsquigarrow_R$. The R-reduction relation is the reflexive and transitive closure of the R-rewrite relation.

A rewrite rule is *first-order* if l and r are both first-order, otherwise it is *higher-order*. We denote by R_1 (resp. R_ω) the set of first-order (resp. higher-order) rules.

A first-order rewrite rule $l \rightsquigarrow r$ is *conservative* if no (free) variable has more occurrences in r than in l.

From now on, we will assume that first-order (resp. higher-order) function symbols are defined only by first-order (resp. higher-order) rewrite rules. Actually, it will always be possible to treat a first-order function symbol as an higher-order one.

Here are some examples of rewrite rules:

$\mathtt{if}_t(\mathtt{true},u,v) \rightsquigarrow u$
$\mathtt{if}_t(\mathtt{false},u,v) \rightsquigarrow v$

$\mathtt{+}_\mathtt{nat}(x,\mathtt{0}_\mathtt{nat}) \rightsquigarrow x$
$\mathtt{+}_\mathtt{nat}(x,\mathtt{s}_\mathtt{nat}(y)) \rightsquigarrow \mathtt{s}_\mathtt{nat}(\mathtt{+}_\mathtt{nat}(x,y))$
$\mathtt{+}_\mathtt{nat}(\mathtt{+}_\mathtt{nat}(x,y),z) \rightsquigarrow \mathtt{+}_\mathtt{nat}(x,\mathtt{+}_\mathtt{nat}(y,z))$

$\mathtt{append}_t(\mathtt{nil}_t,l) \rightsquigarrow l$
$\mathtt{append}_t(\mathtt{cons}_t(x,l'),l) \rightsquigarrow \mathtt{cons}_t(x,\mathtt{append}_t(l',l))$

$\mathtt{map}_{t,t'}(f,\mathtt{nil}_t) \rightsquigarrow \mathtt{nil}_{t'}$
$\mathtt{map}_{t,t'}(f,\mathtt{cons}_t(x,l)) \rightsquigarrow \mathtt{cons}_t(f\,x,\mathtt{map}_{t,t'}(f,l))$

With these examples, one can see that CAIC has more expressive power than a first-order algebraic language, or the calculus of constructions. Indeed, in a first-order algebraic language, a function like $\mathtt{map}_{t,t'}$ cannot be defined at all. Yet, it is a powerful and useful function.

In the calculus of constructions, the addition must be defined by induction and therefore cannot include the associativity property. This property has to be proved afterward. Another example is given by the definition of the Ackermann's function. In CAIC, it can be defined as follows:

- $\mathtt{ack}(\mathtt{0}_\mathtt{nat},y) \rightsquigarrow \mathtt{s}_\mathtt{nat}(y)$
- $\mathtt{ack}(\mathtt{s}_\mathtt{nat}(x),\mathtt{0}_\mathtt{nat}) \rightsquigarrow \mathtt{ack}(x,\mathtt{s}_\mathtt{nat}(\mathtt{0}_\mathtt{nat}))$
- $\mathtt{ack}(\mathtt{s}_\mathtt{nat}(x),\mathtt{s}_\mathtt{nat}(y)) \rightsquigarrow \mathtt{ack}(x,\mathtt{ack}(\mathtt{s}_\mathtt{nat}(x),y))$

In Coq [BBC+98], an implementation of the calculus of inductive constructions [CPM90] [Wer94], it must be defined in the following way:

[1]The definition of admissible rewrite rules is postponed to Definition 2.17 on the following page, once typing rules are defined.

```
Fixpoint ack[n:nat]:nat->nat :=
   Cases n of
      O => [m:nat](S m)
      | (S n') => Fix ack2 {ack2/1:nat->nat :=
         [m:nat] Cases p of
            O => (ack n' (S O))
            | (S m') => (ack n' (ack2 m'))
         end}
   end.
```

Definition 2.13 (Function ordering) We define the following quasi-ordering on function symbols: $f \geq_{\mathcal{F}} g$ if and only if g occurs in a defining rule of f.

Definition 2.14 (Reduction relation) Given a set R of rewrite rules, the rewrite relation of CAIC is $\leadsto \; = \; \leadsto_\beta \cup \leadsto_R$. The reduction relation of CAIC is its reflexive and transitive closure denoted by $\leadsto^*$. Its transitive closure is denoted by $\overset{+}{\leadsto}$. Its reflexive, symmetric and transitive closure is denoted by $\overset{*}{\leftrightsquigarrow}$.

A term a reduces to a term a' at position $m \in Pos(a)$, $a \overset{m}{\leadsto} a'$, if $a' = a[b']_m$ and $a_{|m}$ reduces at its root to b'. A term is in *normal form* if it cannot be either β-reduced or R-reduced.

An *expansion* is the inverse of a reduction: a expanses to b if b reduces to a.

Definition 2.15 (Typing rules) A *declaration* is a pair $x{:}a$ made of a variable x and a term a.

An *environment* is an ordered sequence of declarations. If Γ is an environment $x_1{:}a_1, \dots, x_n{:}a_n$ $(n \geq 0)$ then the domain of Γ is $dom(\Gamma) = \{x_1, \dots, x_n\}$, its free variables are $FV(\Gamma) = \bigcup_{x:a \in \Gamma} FV(a)$ and $\Gamma(x_i) = a_i$ $(1 \leq i \leq n)$.

A *typing judgement* is a triple $\Gamma \vdash a : b$ made of an environment Γ and two terms a, b. A term a has *type* b *in an environment* Γ if the judgement $\Gamma \vdash a : b$ can be deduced by the rules of Figure 1 on the next page.

A term is *well-typed in an environment* Γ or is a Γ-term if it has a type in the environment Γ. We denote by WTT the set of the well-typed terms of CAIC. An environment is *valid* if $\star$ is typable in it. An environment is *algebraic* (resp. *extended algebraic*) if, for each of its declarations $x{:}c$, c is an algebraic type (resp. an extended algebraic type or $\star$).

One may be surprised by the conversion rule. In the usual calculus of constructions, its side condition is $b \overset{*}{\leftrightsquigarrow} b'$. So, a priori, our system is not equivalent to the traditional one but to one with $b \overset{*}{\leftrightsquigarrow}_\Gamma b'$ as side condition, where $\overset{*}{\leftrightsquigarrow}_\Gamma$ is defined below. But, once the Church-Rosser's (CR) and subject reduction (SR) properties are established, both systems will appear to be equivalent, i.e. the usual conversion rule will be deducible from our system. Indeed, suppose that b and b' are two well-typed terms such that $b \overset{*}{\leftrightsquigarrow} b'$. Then, by CR, there exists a term c such that $b \leadsto^* c$ and $b' \leadsto^* c$. By SR, c is also well-typed and therefore $b \overset{*}{\leftrightsquigarrow}_\Gamma b'$.

Definition 2.16 (Γ- and Γ_Π-equivalences) [BFG97] Given an environment Γ, two Γ-terms a and b are Γ-*equivalent*, written $a \overset{*}{\leftrightsquigarrow}_\Gamma b$ [2], if they are equivalent through a chain of reductions/expansions $a \leadsto^*_{\rho_1} a_1 \; {}^{*}_{\rho_2}\!\!\leftsquigarrow a_2 \leadsto^*_{\rho_3} \dots a_n \; {}^{*}_{\rho_{n+1}}\!\!\leftsquigarrow b$ $(\rho_i \in \{\beta, R\}$, $1 \leq i \leq n+1$, $n \geq 0)$ whose *intermediate terms* $a_1, \dots, a_n$ are well-typed in Γ (nothing is known about the other terms).

Two well-typed terms a and b are Γ_Π-*equivalent* $(a \overset{*}{\leftrightsquigarrow}{}^{\Pi}_{\Gamma} b)$ if they are Γ-equivalent and their intermediate well-typed terms are products.

From now on, instead of the (conv) rule of Figure 1 on the following page, we will use its equivalent form:

$$\text{(conv')} \quad \frac{\Gamma \vdash a : b \quad \Gamma \vdash b' : p}{\Gamma \vdash a : b'} \quad (p \in \{\star, \Box\},\ b \overset{*}{\leftrightsquigarrow}_\Gamma b')$$

Definition 2.17 (Admissible rewrite rules) [3] A rewrite rule $l \leadsto r$, where l is headed by a function

[2] noted $\overset{c}{=}_{R\beta}$ in [BFG97]

[3] in [BFG97], similar conditions are defined that are called "cube-embeddability"

Figure 1: Typing rules of CAIC

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

$$\text{(sort)} \qquad \vdash \mathtt{s} : \star \qquad (\mathtt{s} \in \mathcal{S})$$

$$\text{(var)} \qquad \frac{\Gamma \vdash c : p}{\Gamma, x{:}c \vdash x : c} \qquad (x \in \mathit{Var}^p \setminus \mathit{dom}(\Gamma),\ p \in \{\star, \Box\})$$

$$\text{(weak)} \qquad \frac{\Gamma \vdash a : b \quad \Gamma \vdash c : p}{\Gamma, x{:}c \vdash a : b} \qquad (x \in \mathit{Var}^p \setminus \mathit{dom}(\Gamma),\ p \in \{\star, \Box\})$$

$$\text{(cons)} \qquad \frac{\Gamma \vdash a_1 : s_1 \quad \ldots \quad \Gamma \vdash a_n : s_n}{\Gamma \vdash C(a_1, \ldots, a_n) : s} \qquad (C \in \mathcal{C}^n,\ \tau(C) = s_1 \rightarrow \ldots \rightarrow s_n \rightarrow \mathtt{s},\ n \geq 0)$$

$$\text{(fun)} \qquad \frac{\Gamma \vdash a_1 : s_1 \quad \ldots \quad \Gamma \vdash a_n : s_n}{\Gamma \vdash f(a_1, \ldots, a_n) : s} \qquad (f \in \mathcal{F}^n_t,\ t = s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s \in \mathcal{T}_{\mathcal{S}},\ n \geq 0)$$

$$\text{(abs)} \qquad \frac{\Gamma, x{:}a \vdash b : c \quad \Gamma \vdash \Pi x{:}a.c : q}{\Gamma \vdash \lambda x{:}a.b : \Pi x{:}a.c} \qquad (x \notin \mathit{dom}(\Gamma),\ q \in \{\star, \Box\})$$

$$\text{(app)} \qquad \frac{\Gamma \vdash a : \Pi x{:}b.c \quad \Gamma \vdash d : b}{\Gamma \vdash a\, d : c\{x \mapsto d\}}$$

$$\text{(conv)} \qquad \frac{\Gamma \vdash a : b \quad \Gamma \vdash b' : p}{\Gamma \vdash a : b'} \qquad (p \in \{\star, \Box\},\ b \leadsto^*_\beta b' \text{ or } b' \leadsto^*_\beta b \text{ or } b \leadsto^*_R b' \text{ or } b' \leadsto^*_R b)$$

$$\text{(prod)} \qquad \frac{\Gamma \vdash a : p \quad \Gamma, x{:}a \vdash b : q}{\Gamma \vdash \Pi x{:}a.b : q} \qquad (x \notin \mathit{dom}(\Gamma),\ p, q \in \{\star, \Box\})$$

symbol whose output type is s, is *admissible* if and only if it satisfies the following conditions:

(Well-typedness) there exists a unique algebraic environment Γ_l such that $\Gamma_l \vdash l : s$,
(Algebraicity) for any environment Γ, $\Gamma \vdash l : s \Rightarrow \Gamma|_{FV(l)} \overset{*}{\leftrightsquigarrow}_{\Gamma} \Gamma_l$,
(Type-preservation) for any environment Γ, $\Gamma \vdash l : s \Rightarrow \Gamma \vdash r : s$.

The decidability of these conditions is studied in Section 8 on page 28. In comparison to [BFG97] where the decidability is proved only for algebraic terms, we prove it for a very large class of rule terms including abstractions and applied variables[4].

The admissibility condition of the set of rewrite rules will be explicitly mentioned each time it is needed, since most of the meta-theoretical results do not need this additional hypothesis.

As an example, let us prove the admissibility of the following rewrite rule: $\mathtt{+}_{\mathtt{nat}}(u, \mathtt{s}_{\mathtt{nat}}(v)) \leadsto \mathtt{s}_{\mathtt{nat}}(\mathtt{+}_{\mathtt{nat}}(u,v))$. The well-typedness condition is satisfied by taking $\Gamma = u : \mathtt{nat}, v : \mathtt{nat}$. The algebraicity follows from the fact that each variable is the argument of a function symbol. This implies also the type preservation condition since the righthand side of the rule is well-typed in Γ.

Now, imagine a rule whose lefthand side contains a subterm like $(x\, y)$ and whose righthand side contains only y. Then, the type preservation condition cannot be satisfied for all environments. Indeed, it suffices to consider an environment where the type of y is not equivalent to any algebraic type. It does not matter in the lefthand side, but the righthand side cannot be well-typed anymore since function symbols operate only on arguments whose types are algebraic.

[4] see Definition 8.1 on page 29

Definition 2.18 (Positive and negative type positions) Given an algebraic type s, its sets of positive and negative positions are inductively defined as follows:

- $Pos^+(s) = \epsilon$ if $s \in \mathcal{S}$
- $Pos^-(s) = \emptyset$ if $s \in \mathcal{S}$
- $Pos^+(s \to t) = 1 \cdot Pos^-(s) \,\cup\, 2 \cdot Pos^+(t)$
- $Pos^-(s \to t) = 1 \cdot Pos^+(s) \,\cup\, 2 \cdot Pos^-(t)$

Given an algebraic type t, we say that s *occurs positively* in t if s occurs in t, and each occurrence of s in t is at a positive position.

If s does not occur positively in t then, either s does not occur in t, or s occurs at a negative position in t.

Definition 2.19 (Inductive sorts) Let s be a sort whose constructors are $C_1, \dots, C_n$ $(n \geq 1)$ and suppose that C_i has type $s^i_1 \to \dots \to s^i_{n_i} \to s$ $(1 \leq i \leq n,\ n_i \geq 0)$. Then we say that:

- s is a *basic inductive sort* if each s^i_j is either s or a basic inductive sort strictly smaller than s in $\leq_{\mathcal{S}}$,
- s is a *strictly positive inductive sort* if each s^i_j is either a strictly positive inductive sort strictly smaller than s in $\leq_{\mathcal{S}}$ or of the form $s'_1 \to \dots \to s'_p \to s$ where each s'_k is built from strictly positive inductive sorts strictly smaller than s in $\leq_{\mathcal{S}}$,
- s is a *positive inductive sort* if s does not occur in negative position in each s^i_j.

For example, the sort $\mathtt{nat}$ whose constructors are $\mathtt{0}_{\mathtt{nat}}$:$\mathtt{nat}$ and $\mathtt{s}_{\mathtt{nat}}$:$\mathtt{nat} \to \mathtt{nat}$ is a basic sort. The sort $\mathtt{ord}$ whose constructors are $\mathtt{0}_{\mathtt{ord}}$:$\mathtt{ord}$, $\mathtt{s}_{\mathtt{ord}}$:$\mathtt{ord} \to \mathtt{ord}$ and $\mathtt{lim}_{\mathtt{ord}}$:$(\mathtt{nat} \to \mathtt{ord}) \to \mathtt{ord}$ is a strictly positive sort since $\mathtt{ord} >_{\mathcal{S}} \mathtt{nat}$.

The sort $\mathtt{mon}$ whose constructors are $\mathtt{0}_{\mathtt{mon}}$:$\mathtt{mon}$ and $\mathtt{C}_{\mathtt{mon}}$:$((\mathtt{mon} \to \mathtt{bool}) \to \mathtt{mon}) \to \mathtt{mon}$ is a positive sort. Indeed, $\mathtt{mon} >_{\mathcal{S}} \mathtt{bool}$, $Pos^+((\mathtt{mon} \to \mathtt{bool}) \to \mathtt{mon}) = \{2, 1 \cdot 1\}$ and $\mathtt{mon}$ occurs only at these positions.

Definition 2.20 (Strictly positive recursors) Let t be an algebraic type and s a strictly positive inductive sort generated by the constructors $C_1, \dots, C_n$ $(n \geq 1)$ of respective types $s^i_1 \to \dots \to s^i_{n_i} \to \mathsf{s}$ $(1 \leq i \leq n,\ n_i \geq 0)$. Its associated recursor rec^{s}_t is a function symbol of arity $n+1$ and type $\mathsf{s} \to t_1 \to \dots \to t_n \to t$ where $t_i = s^i_1 \to \dots \to s^i_{n_i} \to s^i_1\{\mathsf{s} \mapsto t\} \to \dots \to s^i_{n_i}\{\mathsf{s} \mapsto t\} \to t$. It is defined via the following rewrite rules:

$$rec^{\mathsf{s}}_t(C_i(\vec{a}), \vec{b}) \rightsquigarrow b_i\, \vec{a}\, \vec{d}$$

where $d_j = \lambda \vec{x}{:}\vec{v}.rec^{\mathsf{s}}_t(a_j\, \vec{x}, \vec{b})$ if $s^i_j = s'_1 \to \dots \to s'_p \to \mathsf{s}$ $(p \geq 0)$ and $d_j = a_j$ otherwise.

Definition 2.21 (Positive recursors) Let t be an algebraic type and s a strictly positive inductive sort generated by the constructors $C_1, \dots, C_n$ $(n \geq 1)$ of respective types $s^i_1 \to \dots \to s^i_{n_i} \to \mathsf{s}$ $(1 \leq i \leq n,\ n_i \geq 0)$. Its associated recursor rec^{s}_t is a function symbol of arity $n+1$ and type $\mathsf{s} \to t_1 \to \dots \to t_n \to t$ where $t_i = s^i_1 \to \dots \to s^i_{n_i} \to s^i_1\{\mathsf{s} \mapsto t\} \to \dots \to s^i_{n_i}\{\mathsf{s} \mapsto t\} \to t$. It is defined via the following rewrite rules:

$$rec^{\mathsf{s}}_t(C_i(\vec{a}), \vec{b}) \rightsquigarrow b_i\, \vec{a}\, \vec{d}$$

where $d_j = R^{\mathsf{s}}_t(a_j, s^i_j)$ is inductively defined as follows:

$$R^{\mathsf{s}}_t(e, u_1 \to \dots \to u_k \to \mathsf{s}') = \lambda x_1{:}\, u'_1 \dots \lambda x_k{:}\, u'_k. \begin{cases} rec^{\mathsf{s}}_t(e\, f_1 \dots f_k, \vec{b}) & \text{if } \mathsf{s}' = \mathsf{s} \\ e\, f_1 \dots f_k & \text{otherwise} \end{cases}$$

where $u'_j = u_j\{\mathsf{s} \mapsto t\}$ if s occurs positively in $u_1 \to \dots \to u_k \to \mathsf{s}'$, u_j otherwise, and $f_j = R^{\mathsf{s}}_t(x_j, u_j)$.

Via the Curry-Howard isomorphism [How80], a recursor of a sort s corresponds to the structural induction principle associated to the set of elements built from the constructors of s. Recursors for strictly positive types are found in many proof assistants based on the Curry-Howard isomorphism, e.g. in Coq [PM93].

Here are a few examples of recursors:

$\mathtt{rec}^t_{\mathtt{bool}}(\mathtt{true}, u, v) \rightsquigarrow u$

$\mathtt{rec}^t_{\mathtt{bool}}(\mathtt{false},u,v) \rightsquigarrow v$ (note that $\mathtt{rec}^t_{\mathtt{bool}}$ is the same as $\mathtt{if}_t$)

$\mathtt{rec}^t_{\mathtt{nat}}(0_{\mathtt{nat}},u,v) \rightsquigarrow u$
$\mathtt{rec}^t_{\mathtt{nat}}(\mathtt{s}_{\mathtt{nat}}(n),u,v) \rightsquigarrow v\; n\; \mathtt{rec}^t_{\mathtt{nat}}(n,u,v)$ (this is Gödel's higher-order primitive recursion)

$\mathtt{rec}^t_{\mathtt{ord}}(0_{\mathtt{ord}},u,v,w) \rightsquigarrow u$
$\mathtt{rec}^t_{\mathtt{ord}}(\mathtt{s}_{\mathtt{ord}}(n),u,v,w) \rightsquigarrow v\; n\; \mathtt{rec}^t_{\mathtt{ord}}(n,u,v,w)$
$\mathtt{rec}^t_{\mathtt{ord}}(\mathtt{lim}_{\mathtt{ord}}(f),u,v,w) \rightsquigarrow w\; f\; \lambda n{:}\mathtt{nat}.\mathtt{rec}^t_{\mathtt{ord}}(f\,n,u,v,w)$

$\mathtt{rec}^t_{\mathtt{mon}}(0_{\mathtt{mon}},u,v) \rightsquigarrow u$
$\mathtt{rec}^t_{\mathtt{mon}}(\mathtt{C}_{\mathtt{mon}}(f),u,v) \rightsquigarrow v\; f\; \lambda x{:}\, t \rightarrow \mathtt{bool}.\mathtt{rec}^t_{\mathtt{mon}}(f\; \lambda y{:}\mathtt{mon}.x\; \mathtt{rec}^t_{\mathtt{mon}}(y,u,v),\, u,v)$

3 Meta-theory

The aim of this section is to establish some structural properties of CAIC. Most of them are classical properties of pure type systems (PTS).

Definition 3.1 (Term classes)
- The set of *kinds* is defined by $Kind = \{K \in Term \mid \exists\Gamma,\, \Gamma \vdash K : \Box\}$.
- The set of *type constructors* is defined by $Constr = \{T \in Term \mid \exists\Gamma, K \in Kind,\, \Gamma \vdash T : K\}$.
- The set of *types* is defined by $Type = \{\tau \in Term \mid \exists\Gamma,\, \Gamma \vdash \tau : \star\}$.
- The set of *objects* is defined by $Obj = \{u \in Term \mid \exists\Gamma, \tau \in Type,\, \Gamma \vdash u : \tau\}$.

Besides, by Γ-kind, Γ-type, ... we mean a Γ-term belonging to the appropriate class.

Lexicography: In the sequel, we will use the following lexicographical conventions: K, K' will denote kinds, T, T' type constructors, τ, σ types, u, v objects, s, t algebraic types, p, q, $\star$ or $\Box$, f, g function symbols, C, C' constructors and a, b, c, d, e arbitrary terms.

As for the calculus of constructions, CAIC can be seen as a PTS. It is easy to see that the introduction of rewriting does not affect the structural properties shared by all the PTS, even though rewriting is defined on a larger class of terms. Hence we recall these properties without necessary providing their proof, that the interested reader can find in [GN91] or [Bar93].

Theorem 3.2 (PTS structural properties) [GN91][Bar93]

(free variables) If $\Gamma \vdash a : b$ and $\Gamma = x_1{:}c_1, \dots, x_n{:}c_n$ then all the variables x_i are distinct, $FV(a) \cup FV(b) \subseteq \{x_1, \dots, x_n\}$ and $FV(c_i) \subseteq \{x_1, \dots, x_{i-1}\}$ $(1 \le i \le n)$.

(substitution) If $\Gamma, x{:}a, \Delta \vdash b : c$ and $\Gamma \vdash d : a$ then $\Gamma, \Delta\{x \mapsto d\} \vdash b\{x \mapsto d\} : c\{x \mapsto d\}$.

(thinning) If $\Gamma \vdash a : b$ and $\Gamma \subseteq \Gamma'$ then $\Gamma' \vdash a : b$.

(stripping) [5] If $\Gamma \vdash a : b$ then
- $a = p \Rightarrow b \overset{*}{\leftrightsquigarrow}_\Gamma \Box,\ p = \star$
- $a = s \Rightarrow b \overset{*}{\leftrightsquigarrow}_\Gamma \star$
- $a = x \in Var^p \Rightarrow b \overset{*}{\leftrightsquigarrow}_\Gamma b',\ x{:}b' \in \Gamma,\ \Gamma \vdash b' : p$
- $a = C(a_1, \dots, a_n),\ \tau(C) = s_1 \rightarrow \dots \rightarrow s_n \rightarrow s \Rightarrow b \overset{*}{\leftrightsquigarrow}_\Gamma s,\ \Gamma \vdash a_i : s_i\ (1 \le i \le n)$
- $a = f(a_1, \dots, a_n),\ \tau(f) = s_1 \rightarrow \dots \rightarrow s_n \rightarrow s \Rightarrow b \overset{*}{\leftrightsquigarrow}_\Gamma s,\ \Gamma \vdash a_i : s_i\ (1 \le i \le n)$
- $a = \Pi x{:}c.d \Rightarrow b \overset{*}{\leftrightsquigarrow}_\Gamma q,\ \Gamma \vdash c : p,\ \Gamma, x{:}c \vdash d : q$
- $a = \lambda x{:}c.d \Rightarrow b \overset{*}{\leftrightsquigarrow}_\Gamma \Pi x{:}c.e,\ \Gamma, x{:}c \vdash d : e,\ \Gamma \vdash c : p,\ \Gamma, x{:}c \vdash e : q$
- $a = c\,d \Rightarrow b \overset{*}{\leftrightsquigarrow}_\Gamma e_2\{x \mapsto d\},\ \Gamma \vdash c : \Pi x{:}e_1.e_2,\ \Gamma \vdash d : e_1$

(well-foundedness) [6] If $\Gamma \vdash a : b$ then either $b = \Box$ or $\Gamma \vdash b : p$.

(type uniqueness) [7] If $\Gamma \vdash a : b$ and $\Gamma \vdash a : b'$ then $b \overset{*}{\leftrightsquigarrow}_\Gamma b'$.

[5] called "generation lemma" in [GN91]
[6] called "correctness" in [BFG97]
[7] only for singly sorted PTS [Bar93] (called functional PTS in [GN91]) which is the case of the calculus of constructions

Lemma 3.3 (Γ-unexpansivity of $\star$ and $\Box$) [BFG97] If a is a Γ-term and $a \overset{*}{\underset{\Gamma}{\leftrightsquigarrow}} p$ then $a = p$.

Proof. Suppose that $a \neq p$. Then, there exists a non empty chain of reductions/expansions from a to p. As p is irreducible, the chain terminates by a reduction $a' \rightsquigarrow_{\rho}^{*} p$ where a' is a well-typed intermediate term.

If this is an R-reduction then the last R-rewrite must take place at the root since p is not headed by a function symbol. Hence, $a' = f(\vec{c}\theta)$ and $p = e\theta$ where $f(\vec{c}) \rightsquigarrow e$ is the last applied rewrite rule. $e \neq p$ since p is not a rule term. Thus, $e = x \in Var^{\star}$ and $x\theta = p$, which is not possible for typing reasons.

If this is a β-reduction then a' must be of the form $(\lambda x{:}b.c)\,d$ with p occurring as a subterm of c or d. $p \neq \Box$ since $\Box$ is not typable. Hence, $p = \star$. Thanks to successive strippings, we can suppose that $c = \star$ or $d = \star$. If $c = \star$ then $\lambda x{:}b.c$ has type $\Pi x{:}b.\Box$ which is not typable. If $d = \star$ then b is of type $\Box$ which is not typable.

Hence, there cannot exist a non empty chain from a to p and $a = p$. $\diamond$

From now on, we will use a strengthened version of the stripping lemma where $b = p$ whenever $b \overset{*}{\underset{\Gamma}{\leftrightsquigarrow}} p$.

Lemma 3.4 (Kind structure) [BFG97] A term a is a Γ-kind if and only if a is a Γ-term of the form $\Pi\vec{x}{:}\vec{b}.\star$.

Proof. The condition is sufficient. The converse is proved by induction on the structure of the type derivation of a (the conversion rule cannot apply on kinds since $\Box$ is not expansible). $\diamond$

Lemma 3.5 (Γ-stability of kinds) [BFG97] Given a Γ-kind a and a Γ-term b, if $a \overset{*}{\underset{\Gamma}{\leftrightsquigarrow}} b$ then b is also a Γ-kind.

Proof. The property is proved by induction on the length of the chain of reductions/expansions between a and b. By the kind structure lemma, as a is a kind, it is a well-typed product over $\star$. If $a \rightsquigarrow_{\rho}^{*} b$ ($\rho \in \{\beta, R\}$) then b has the same structure as a and as it is well-typed, it must be a kind. If $b \rightsquigarrow_{\rho}^{*} a$ then, as in the proof of the unexpansivity lemma of $\star$ and $\Box$ (Lemma 3.3), b has the same structure as a. Hence b is also a kind. $\diamond$

Lemma 3.6 (Classification) $\mathit{Constr} \cap \mathit{Kind} = \mathit{Kind} \cap \mathit{Obj} = \mathit{Obj} \cap \mathit{Constr} = \emptyset$

Proof.

- $\mathit{Constr} \cap \mathit{Kind} = \emptyset$:
  If $\Gamma \vdash a : K$, $\Gamma \vdash K : \Box$ and $\Gamma' \vdash a : \Box$ then, by the kind structure lemma, $a = \Pi\vec{x}{:}\vec{b}.\star$ for a non empty sequence of terms $\vec{b}$ since a cannot be equal to $\star$. Hence, by stripping on $\Gamma \vdash a{:}K$, $K = p \in \{\star, \Box\}$. But, as $\Gamma \vdash K{:}\Box$, $K = \star$. So, by stripping again, $\Gamma, \vec{x}{:}\vec{b} \vdash \star{:}\star$ which is not possible.
- $\mathit{Kind} \cap \mathit{Obj} = \emptyset$:
  If $\Gamma \vdash a{:}\tau$, $\Gamma \vdash \tau{:}\star$ and $\Gamma' \vdash a{:}\Box$ then, by the kind structure lemma, $a = \Pi x{:}b.c$ for some terms b and c since a cannot be equal to $\star$. Hence, by stripping, $\tau = \Box$ which is not possible.
- $\mathit{Obj} \cap \mathit{Constr} = \emptyset$: [BFG97]
  By induction on $\Gamma \vdash a{:}\tau$ such that $\Gamma \vdash \tau{:}\star$, one can show it is not possible that $\Gamma' \vdash a{:}K$ and $\Gamma' \vdash K{:}\Box$ for any environment Γ' and term K, thanks to the stripping lemma.

$\diamond$

Lemma 3.7 (Typed structure) The term classes have the following *typed structure*:

- $K := \star \mid \Pi x{:}\tau.K \mid \Pi\alpha{:}K.K$
- $T := \mathsf{s} \mid \alpha \mid \Pi x{:}\tau.\tau \mid \Pi\alpha{:}K.\tau \mid \lambda x{:}\tau.T \mid \lambda\alpha{:}K.T \mid (T\,u) \mid (T\,T)$
- $u := x \mid C(u_1, \ldots, u_n) \mid f(u_1, \ldots, u_n) \mid \lambda x{:}\tau.u \mid \lambda\alpha{:}K.u \mid (u\,u) \mid (u\,T)$

Proof. All these conditions are necessary. They are sufficient for kinds. We prove they are sufficient for type constructors and objects by induction on the type derivation. The only non straightforward case is that of the conversion rule. Suppose that T is a type constructor such that $\Gamma \vdash T{:}K'$, $\Gamma \vdash K'{:}\Box$, $\Gamma \vdash T{:}K$ and $K \overset{*}{\underset{\Gamma}{\leftrightsquigarrow}} K'$. Then, by Γ-stability of kinds, $\Gamma \vdash K : \Box$ and we can apply the induction hypothesis. Suppose now that u is an object such that $\Gamma \vdash u{:}\tau'$, $\Gamma \vdash \tau'{:}\star$, $\Gamma \vdash u{:}\tau$ and $\tau \overset{*}{\underset{\Gamma}{\leftrightsquigarrow}} \tau'$. Then, by well-foundedness, $\tau = \Box$ or $\Gamma \vdash \tau{:}p \in \{\star, \Box\}$. The first case is not possible since $\Box$ is not expansible and not typable. If $\Gamma \vdash \tau : \Box$ then τ is a kind and, by Γ-stability of kinds, τ' must also be a kind. This is not possible hence τ is a type and the induction hypothesis applies. $\diamond$

Lemma 3.8 (Rule terms are objects) Given a rule term a, if $\Gamma \vdash a{:}b$ then $\Gamma \vdash b{:}\star$.

Proof. We prove the property by induction on the rule term structure of a. By well-foundedness, $b = \Box$ or $\Gamma \vdash b{:}p \in \{\star, \Box\}$. If $b = \Box$ then, by the kind structure lemma, $a = \star$ or a is a product and cannot be a rule term. Suppose that $\Gamma \vdash b{:}\Box$. Then a cannot be a variable of $Var^\star$, a function symbol or an object constructor headed term.

If $a = c\,d$ then, by stripping, $\Gamma \vdash c : \Pi x{:}e_1.e_2$, $\Gamma \vdash d : e_1$ and $b \overset{*}{\leftrightsquigarrow}_\Gamma e_2\{x \mapsto d\}$. By induction hypothesis, $\Gamma \vdash \Pi x{:}e_1.e_2 : \star$ thus $\Gamma, x{:}e_1 \vdash e_2 : \star$ and, by substitution, $\Gamma \vdash e_2\{x \mapsto d\} : \star$. By Γ-stability of kinds, $f\{x \mapsto d\}$ should be a kind, which is not possible by the classification lemma.

If $a = \lambda x{:}s.c$ then, by stripping, $\Gamma, x{:}s \vdash c : d$ and $b \overset{*}{\leftrightsquigarrow}_\Gamma \Pi x{:}s.d$. By induction hypothesis, $\Gamma \vdash d : \star$ thus $\Gamma \vdash \Pi x{:}s.d{:}\star$. By Γ-stability of kinds, $\Pi x{:}s.d$ should be a kind, which is not possible by the classification lemma again.

Therefore, the only possible solution is $\Gamma \vdash b{:}\star$. $\diamond$

4 Subject reduction and Γ-consistence of the extended algebraic types

In this section, we establish two important properties: subject reduction on the one hand, and Γ-consistence of extended algebraic types on the other hand, i.e. two Γ-equivalent extended algebraic types are syntactically equal. It is worth noting that these two properties are a consequence of a partial βh-normalization of the Γ-equivalence relation as explained below.

Definition 4.1 (Subject reduction) The subject reduction property expresses the fact that reduction preserves typing: if $\Gamma \vdash a{:}b$ and $a \rightsquigarrow^* a'$ then $\Gamma \vdash a'{:}b$.

If a reduction relation satisfies this property, no type error can occur during the execution of a program. And from a logical point of view, it ensures that if a is a proof of the proposition b then its reduct a' is also a proof of b.

The proof that our calculus enjoys the subject reduction property is non trivial. For the case of a rewrite step, it is done by induction on the structure of the type derivation of the rewrite term. The only non straightforward case is that of the application rule when there is a rewrite at the root. The following intermediate result will be needed:

$$\Pi x{:}a.b \overset{*}{\leftrightsquigarrow}_\Gamma \Pi x{:}a'.b' \;\Rightarrow\; a \overset{*}{\leftrightsquigarrow}_\Gamma a' \text{ and } b \overset{*}{\leftrightsquigarrow}_\Gamma b'.$$

In the calculus of constructions, the corresponding property (with $\overset{*}{\leftrightsquigarrow}$ instead of $\overset{*}{\leftrightsquigarrow}_\Gamma$) is obtained thanks to the Church-Rosser property which has not yet been shown for our calculus. We need to prove it directly. This explains why we adopted a different conversion rule where β-reductions and R-reductions are separated.

As $\Pi x{:}a.b \overset{*\,\Pi}{\leftrightsquigarrow}_\Gamma \Pi x{:}a'.b' \;\Rightarrow\; a \overset{*}{\leftrightsquigarrow}_\Gamma a'$ and $b \overset{*}{\leftrightsquigarrow}_\Gamma b'$, the idea is to show that

$$\Pi x{:}a.b \overset{*}{\leftrightsquigarrow}_\Gamma \Pi x{:}a'.b' \;\Rightarrow\; \Pi x{:}a.b \overset{*\,\Pi}{\leftrightsquigarrow}_\Gamma \Pi x{:}a'.b'.$$

We know that if $\Pi x : a.b \rightsquigarrow^* c$ then $c = \Pi x : a'.b'$. By Γ-stability of kinds, we also know that if $c \rightsquigarrow^* \Pi x{:}a.b$ and $\Pi x{:}a.b$ is a kind, then $c = \Pi x{:}a'.b'$. We are left with the case where $\Pi x{:}a.b$ is a type, and there are two sub-cases. If $c \rightsquigarrow^*_R \Pi x{:}a.b$ then $c = \Pi x{:}a'.b'$ by the R-head-unexpansivity of types (proved just below). We are left with the second sub-case, for which $c \rightsquigarrow^*_\beta \Pi x{:}a.b$. The idea is then to use a standard derivation (outside-in) for which $c \rightsquigarrow^*_{\beta h} \Pi x{:}a'.b' \rightsquigarrow^*_\beta \Pi x{:}a.b$. To complete the proof we need several technical lemmas.

Lemma 4.2 (Subject reduction for R-reduction) Assuming that the rewrite rules are admissible and $\Gamma \vdash a{:}b$, if $a \rightsquigarrow^*_R a'$ then $\Gamma \vdash a'{:}b$, and if $\Gamma \rightsquigarrow^*_R \Gamma'$ then $\Gamma' \vdash a{:}b$.

Proof. We prove it for one rewrite step, by induction on the structure of the derivation of $\Gamma \vdash a : b$. The only non straightforward case is that of the (fun) rule when the rewrite is at the root. Suppose that the applied rewrite rule is $f(\vec{c}) \rightsquigarrow e$. Then there exists a substitution $\{\vec{x} \mapsto \vec{d}\}$ such that $a = f(\vec{c}\{\vec{x} \mapsto \vec{d}\})$ and $a' = e\{\vec{x} \mapsto \vec{d}\}$. It is easy to see that there exists some terms $\vec{\tau}$ such that $\Gamma \vdash d_i{:}\tau_i$ $(1 \le i \le |\vec{d}|)$ and $\Gamma, \vec{x}{:}\vec{\tau} \vdash f(\vec{c}){:}b$. By stripping, $b \overset{*}{\leftrightsquigarrow}_\Gamma s$ where s is the output type of f. By conversion, $\Gamma, \vec{x}{:}\vec{\tau} \vdash f(\vec{c}) : s$. Since rewrite rules are admissible, $\Gamma, \vec{x}{:}\vec{\tau} \vdash e : s$. By conversion again, $\Gamma, \vec{x}{:}\vec{D} \vdash e : b$. And since the variables in $\vec{x}$ are not free in b nor in Γ, by substitution, $\Gamma \vdash e\{\vec{x} \mapsto \vec{d}\}{:}b$. $\diamond$

Lemma 4.3 (R-head-unexpansivity of types and kinds) Assume that a is a well-typed term in an environment Γ, $a \rightsquigarrow^*_R a'$ and $\Gamma \vdash a' : p \in \{\star, \Box\}$. Then, no rewrite from a to a' may occur at the root. Besides, if a' is a product, then a is also a product.

Proof. Let b be a type of a in Γ. By the subject reduction lemma for R-reduction, $\Gamma \vdash a' : b$ and, by type uniqueness, $b \overset{*}{\leftrightsquigarrow}_\Gamma p$. Therefore, by the Γ-unexpansivity of $\star$ and $\Box$, $b = p$ and a is a type or a kind. By the typed structure lemma, there cannot be rewrites at the root since types and kinds cannot be headed by function symbols. Hence, if a' is a product, then a must also be a product. $\diamond$

Lemma 4.4 (Subject reduction on types for βh-reduction) If $\Gamma \vdash a : \star$ and $a \rightsquigarrow^*_{\beta h} b$ then $\Gamma \vdash b : \star$.

Proof. We prove it for a βh-rewrite, that is, if $\Gamma \vdash (\lambda x{:}a.b)\, c : \star$ then $\Gamma \vdash b\{x \mapsto c\} : \star$. By stripping, we find that $\star \overset{*}{\leftrightsquigarrow}_\Gamma d'\{x \mapsto c\}$, $\Gamma \vdash \lambda x{:}a.b : \Pi x{:}a'.d'$, $\Gamma \vdash c : a'$, $\Pi x{:}a'.d' \overset{*}{\leftrightsquigarrow}_\Gamma \Pi x{:}a.d$, $\Gamma, x{:}a \vdash b : d$ and $\Gamma \vdash a : p \in \{\star, \Box\}$. If $d' = x$ then $c = \star$ and $a = \Box$ which is not possible since a must be typable. Thus $d' = \star$ and $\Pi x{:}a'.d'$ is a kind. By Γ-stability of kinds, $\Pi x{:}a.d$ is also a kind. Therefore $d = \star$ and $a \overset{*}{\leftrightsquigarrow}_\Gamma a'$. Then, by conversion, $\Gamma \vdash c : a$ and, by substitution, $\Gamma \vdash b\{x \mapsto c\} : \star$. $\diamond$

Lemma 4.5 (Commutativity between R-reduction and βh-reduction) [BFG97] If $a \rightsquigarrow^*_R b$ and $a \rightsquigarrow^*_{\beta h} c$, then there exists a term d such that $b \rightsquigarrow^*_{\beta h} d$ and $c \rightsquigarrow^*_R d$. Besides, if b is a type and c is a product, then d is a product type.

Proof. It suffices to prove the property for a βh-rewrite step, that is, if $a \rightsquigarrow^*_R b$ and $a \rightsquigarrow_{\beta h} c$, then there exists a term d such that $b \rightsquigarrow_{\beta h} d$ and $c \rightsquigarrow^*_R d$. If $a \rightsquigarrow_{\beta h} c$, then $a = (\lambda x : e_1.e_2)\, e_3$, $c = e_2\{x \mapsto e_3\}$ and $b = (\lambda x : e'_1.e'_2)\, e'_3$ for some terms $\vec{e}, \vec{e'}$ such that $e_1 \rightsquigarrow^*_R e'_1$, $e_2 \rightsquigarrow^*_R e'_2$ and $e_3 \rightsquigarrow^*_R e'_3$. So, it suffices to take $d = e'_2\{x \mapsto e'_3\}$ for which $b \rightsquigarrow_{\beta h} d$ and $c \rightsquigarrow^*_R d$. Hence, if c is a product then d must also be a product. And if b is a type then, by the subject reduction property of the βh-rewrite relation on types, d is a type. $\diamond$

Lemma 4.6 (Commutativity between β-reduction and βh-reduction) [BFG97] If $a \rightsquigarrow^*_\beta b$ and $a \rightsquigarrow^*_{\beta h} c$, then there exists a term d such that $b \rightsquigarrow^*_{\beta h} d$ and $c \rightsquigarrow^*_\beta d$. Besides, if b is a type and c is a product, then d is a product type.

Proof. It suffices to prove the property for a βh-rewrite step, that is, if $a \rightsquigarrow^*_\beta b$ and $a \rightsquigarrow_{\beta h} c$, then there exists a term d such that $c \rightsquigarrow^*_\beta d$ and $b = d$ or $b \rightsquigarrow_{\beta h} d$. Let us prove it by induction on the length of the rewrite sequence from a to b. If $b = a$ then it suffices to take $d = c$. Suppose now that $a \rightsquigarrow_\beta a' \rightsquigarrow^*_\beta b$. If $a \rightsquigarrow_{\beta h} a'$ then $a' = c$ and it suffices to take $d = b$. Otherwise, as $a \rightsquigarrow_{\beta h} c$, there exists some terms e, f, g such that $a = (\lambda x{:}e_1.e_2)\, e_3$ and $c = e_2\{x \mapsto e_3\}$. Since the β-rewrite from a to a' is not a βh-rewrite, there exists some terms e'_1, e'_2, e'_3 such that $a' = (\lambda x{:}e'_1.e'_2)\, e'_3$, $e_1 \rightsquigarrow_\beta e'_1$, $e_2 \rightsquigarrow_\beta e'_2$ or $e_3 \rightsquigarrow_\beta e'_3$. Hence $a' \rightsquigarrow_{\beta h} e'_2\{x \mapsto e'_3\}$ and, by induction hypothesis, there exists a term d such that $e'_2\{x \mapsto e'_3\} \rightsquigarrow^*_\beta d$ and $b = d$ or $b \rightsquigarrow_{\beta h} d$. As $c = e_2\{x \mapsto e_3\} \rightsquigarrow^*_\beta e'_2\{x \mapsto e'_3\}$, that d works.

Besides, if c is a product then d must also be a product. And if b is a type then, by the subject reduction property of the βh-rewrite relation on types, d is a type. $\diamond$

Lemma 4.7 (Postponement on types of R-reduction wrt βh-reduction) [BFG97] If *a is a type* and
$a \rightsquigarrow^*_R b \rightsquigarrow^*_{\beta h} c$, then there exists a term d such that $a \rightsquigarrow^*_{\beta h} d \rightsquigarrow^*_R c$. Besides, if c is a product, then d is a product type.

Proof. It suffices to prove the property for a βh-rewrite step, that is, if *a is a type*, $a \rightsquigarrow^*_R b$ and $b \rightsquigarrow_{\beta h} c$, then there exists a term d such that $a \rightsquigarrow_{\beta h} d$ and $d \rightsquigarrow^*_R c$. If $b \rightsquigarrow_{\beta h} c$ then $b = (\lambda x{:}e'_1.e'_2)\, e'_3$ and $c = e_2\{x \mapsto e_3\}$ for some terms e'_1, e'_2, e'_3. Furthermore, as a is a type, by the R-head-unexpansivity lemma of types and kinds, $a = (\lambda x : e_1.e_2)\, e_3$ for some terms $\vec{e}$ such that $e_1 \rightsquigarrow^*_R e'_1$, $e_2 \rightsquigarrow^*_R e'_2$ and $e_3 \rightsquigarrow^*_R e'_3$. So, it suffices to take $d = e_2\{x \mapsto e_3\}$ for which $a \rightsquigarrow_{\beta h} d$ and $d \rightsquigarrow^*_R c$. Furthermore, if c is a product then, by the R-head-unexpansivity lemma again, d is also a product. And, since a is a type, by the subject reduction property of the βh-rewrite relation on types, d is a product type. $\diamond$

Lemma 4.8 (Postponement of β-reduction wrt βh-reduction) [BFG97] If $a \rightsquigarrow^*_\beta b \rightsquigarrow^*_{\beta h} c$, then there exists a term d such that $a \rightsquigarrow^*_{\beta h} d \rightsquigarrow^*_\beta c$ with no βh-rewrite from d to c. Besides, if a is a type and c is a product, then d is a product type.

Proof. This results from the standardization theorem of the untyped λ-calculus (see for example [Bar84]). We can use it also in our calculus by applying it recursively inside the variable type of the abstractions. So, if a reduces to c then there exists an outside-in β-reduction from a to c. Hence it suffices to take the last βh-reduced term for d. Besides, if c is a product, then d is also a product since there is no βh-rewrites between d and c. And if a is a type, then, by the subject reduction property of the βh-rewrite relation on types, d is a product type. $\diamond$

Lemma 4.9 (*βh*-normalization of Γ-equivalence for product types) [BFG97] Assume that c is a type Γ-equivalent to a product type $\Pi x{:}a.b$ through a chain of reductions/expansions whose intermediate well-typed terms are types. Then, c has a βh-normal form Γ_Π-equivalent to $\Pi x{:}a.b$.

Proof. Suppose that $\Pi x{:}a.b \;{}^{*}_{\rho_1}\!\!\leftsquigarrow c_1 \rightsquigarrow^{*}_{\rho_2} c_2 \;{}^{*}_{\rho_3}\!\!\leftsquigarrow \ldots c_n \;{}^{*}_{\rho_{n+1}}\!\!\leftsquigarrow c$ ($\rho_i \in \{\beta, R\}$, $1 \le i \le n{+}1$, $n \ge 0$). As c_1 is a type and $\Pi x{:}a.b$ is a product in βh-normal form, by the postponement lemmas, there exists a well-typed product d_1 such that $c_1 \rightsquigarrow^{*}_{\beta h} d_1$ and $\Pi x{:}a.b \;{}^{*}_{\rho_1}\!\!\leftsquigarrow d_1$. Then, as c_2 is a type and d_1 is a product, by the commutativity lemmas, there exists a well-typed product d_2 such that $c_2 \rightsquigarrow^{*}_{\beta h} d_2$ and $d_1 \rightsquigarrow^{*}_{\rho_2} d_2$. By repeating this process until c is reached, we get a chain of reductions/expansions $\Pi x{:}a.b \;{}^{*}_{\rho_1}\!\!\leftsquigarrow d_1 \rightsquigarrow^{*}_{\rho_2} d_2 \ldots d_n \;{}^{*}_{\rho_{n+1}}\!\!\leftsquigarrow d_{n+1}$ such that the terms $\vec{d}$ are well-typed products and $c \rightsquigarrow^{*}_{\beta h} d_{n+1}$. Since d_{n+1} is a product, it is in βh-normal form. Hence c has a βh-normal form Γ_Π-equivalent to $\Pi x{:}a.b$. $\diamond$

Lemma 4.10 (Product decomposition) [BFG97] If $\Pi x{:}a.b$ and $\Pi x{:}a'.b'$ are two Γ-equivalent terms through a chain of reductions/expansions whose intermediate well-typed terms are types or kinds, then $a \overset{*}{\leftrightsquigarrow}_\Gamma a'$ and $b \overset{*}{\leftrightsquigarrow}_\Gamma b'$.

Proof. If one of the intermediate well-typed terms is a kind then, by Γ-stability of kinds, all of them are kinds and, by the kind structure lemma, all of them are products. Otherwise, all of them are types and, by βh-normalization, there exists another chain in which all intermediate well-typed terms are products. In both cases, $\Pi x{:}a.b \overset{*\,\Pi}{\leftrightsquigarrow}_\Gamma \Pi x{:}a'.b'$. Hence, $a \overset{*}{\leftrightsquigarrow}_\Gamma a'$ and $b \overset{*}{\leftrightsquigarrow}_\Gamma b'$. $\diamond$

Lemma 4.11 (Subject reduction for *β*-reduction) [BFG97] Assuming that $\Gamma \vdash a : b$, if $a \rightsquigarrow^{*}_{\beta} a'$ then $\Gamma \vdash a' : b$ and, if $\Gamma \rightsquigarrow_\beta \Gamma'$ then $\Gamma' \vdash a : b$.

Proof. By induction on the structure of the derivation of $\Gamma \vdash a : b$. The only non straightforward case is that of the application rule when there is a rewrite at the root:

$$\frac{\Gamma \vdash \lambda x{:}a.b : \Pi x{:}a'.d' \quad \Gamma \vdash c : a'}{\Gamma \vdash (\lambda x{:}a.b)\, c : d'\{x \mapsto c\}}$$

By stripping on $\Gamma \vdash \lambda x{:}a.b : \Pi x{:}a'.d'$, we find that $\Pi x{:}a'.d' \overset{*}{\leftrightsquigarrow}_\Gamma \Pi x{:}a.d$, $\Gamma, x{:}a \vdash b : d$ and $\Gamma \vdash a : p \in \{\star, \Box\}$. By definition of the conversion rule, the intermediate well-typed terms from $\Pi x{:}a.d$ to $\Pi x{:}a'.d'$ are types or kinds. By the product decomposition lemma, $a \overset{*}{\leftrightsquigarrow}_\Gamma a'$ and $d \overset{*}{\leftrightsquigarrow}_\Gamma d'$. By conversion, $\Gamma, x{:}a \vdash b : d'$ and $\Gamma \vdash c : a$. Hence, by substitution, $\Gamma \vdash b\{x \mapsto c\} : d'\{x \mapsto c\}$. $\diamond$

Theorem 4.12 (Subject reduction) [BFG97] Assuming that the rewrite rules are admissible and $\Gamma \vdash a : b$, if $a \rightsquigarrow^{*} a'$ then $\Gamma \vdash a' : b$ and, if $\Gamma \rightsquigarrow^{*} \Gamma'$ then $\Gamma' \vdash a : b$.

Proof. By Lemmas 4.2 on page 10 and 4.11. $\diamond$

Lemma 4.13 (Compatibility of Γ-equivalence with typing) Given two Γ-equivalent well-typed terms a and a', if $\Gamma \vdash a : b$ then $\Gamma \vdash a' : b$. Besides, given two Γ-equivalent environment Γ and Γ', if $\Gamma \vdash a : b$ then $\Gamma' \vdash a : b$.

Proof. By subject reduction and type uniqueness. $\diamond$

Lemma 4.14 (*R*-unexpansivity of sorts and type variables) Given a well-typed term a and a sort or type variable s, if $a \rightsquigarrow^{*}_{R} s$ then $a = s$.

Proof. If $a \ne s$ then a must be of the form $f(\vec{b})$ for some function symbol f and terms $\vec{b}$. Thus $\Gamma \vdash a : s'$ for some environment Γ and algebraic type s'. Then, by subject reduction, $\Gamma \vdash s : s'$ which is not possible. $\diamond$

Lemma 4.15 (βh-normalization of Γ-equivalence for sorts and type variables) Assume that c is a type Γ-equivalent to a sort or a type variable s through a chain of reductions/expansions whose intermediate well-typed terms are types. Then, s is the βh-normal form of c.

Proof. We use the same technique as for product types. Suppose that $s \;{}^{*}_{\rho_1}\!\!\leftsquigarrow c_1 \rightsquigarrow^{*}_{\rho_2} c_2 \;{}^{*}_{\rho_3}\!\!\leftsquigarrow \dots c_n \;{}^{*}_{\rho_{n+1}}\!\!\leftsquigarrow c$ ($\rho_i \in \{\beta, R\}$, $1 \leq i \leq n+1$, $n \geq 0$). As c_1 is a type and s is in βh-normal form, by the postponement lemmas, there exists a well-typed term d_1 such that $c_1 \rightsquigarrow^{*}_{\beta h} d_1$ and $s \;{}^{*}_{\rho_1}\!\!\leftsquigarrow d_1$. By R-unexpansivity of sorts, $\rho_1 = \beta$ and, as there is no βh-rewrite from s to d_1, $d_1 = s$. Then, by the commutativity lemmas, there exists a well-typed term d_2 such that $c_2 \rightsquigarrow^{*}_{\beta h} d_2$ and $s \rightsquigarrow^{*}_{\rho_2} d_2$. As s is irreducible, $d_2 = s$. By repeating this process until c is reached, we obtain that $c \rightsquigarrow^{*}_{\beta h} s$. $\diamond$

Lemma 4.16 (Γ-consistence of extended algebraic types) Given two extended algebraic types s and s', if they are Γ-equivalent through a chain of reductions/expansions whose intermediate well-typed terms are types, then $s = s'$.

Proof. By the product decomposition lemma, the problem is reduced to that of sorts and type variables. By βh-normalization, two Γ-equivalent sorts or type variables are necessarily equal since they are in βh-normal form. $\diamond$

From now on, we will use a strengthened version of the stripping lemma where $b = s$ whenever $b \overset{*}{\underset{\Gamma}{\leftrightsquigarrow}} s \in \mathcal{T}_{\mathcal{S}}$.

5 Strong normalization

In this section, we assume that the reduction relation for which we want to prove the strong normalization property enjoys the subject reduction property.

Definition 5.1 (Strong normalization) A term is *strongly normalizable* if any reduction starting from it terminates.

This property is very important since it implies, together with confluence, the logical soundness of the system (Section 6 on page 26) and the decidability of type-checking (Section 7 on page 27).

To prove the strong normalization property for well-typed terms, we use the interpretation technique of Geuvers [Geu95] which is an extension to the calculus of constructions of the well known "reducibility candidates" of Girard [GLT88] but where types are not interpreted by sets of well-typed terms. This technique is modular and gives a short and flexible proof of the strong normalization property.

For the rewriting aspects of that proof, we rely on the new "general schema" of Jouannaud and Okada for higher-order rules [JO97b].

A proof of $\Gamma \vdash a : b \Rightarrow a \in SN$ by induction on the structure of a does not go through because of the application case: if c and d are strongly normalizable terms, the application $c\,d$ is not strongly normalizable a priori (a well known example in the untyped λ-calculus is the application $\omega\,\omega$ where ω is the strongly normalizable term $\lambda x.xx$). So, the idea is to use a stronger induction hypothesis by proving $\Gamma \vdash a\!:\!b \Rightarrow a \in [\![b]\!]_\xi$ such that $[\![b]\!]_\xi \subseteq SN$ and $[\![\;]\!]$ is "stable by application".

Such an interpretation $[\![b]\!]_\xi$ relies on what is generally called "reducibility candidates". Introductory material can be found in the Chapter 3 of [Wer94]. There exists several definitions such as that of Girard [GLT88], the saturated sets of Tait [Tai67], or that of Parigot [Par93]. A survey is given in [Gal90]. In addition to what is generally required for an interpretation of types, we need the reducibility candidates to be stable by reduction. Besides, for a set of reducibility candidates to be admissible, we require that it contains some subsets of SN which will be used as interpretations of the sorts.

5.1 Interpretation of the types

By "interpretation of the types", we mean the interpretation of terms that can be the type of some other term. This includes $Type$, $Kind$ and $\Box$.

Definition 5.2 (Reducibility candidates) A set C of terms is a *reducibility candidate* if it satisfies the following conditions:

- $\cdot$ $C \subseteq SN$ (reducible elements are strongly normalizable)
- $\cdot$ $Var \subseteq C$ (variables are reducible)
- $\cdot$ $\forall\, a \in Term,\ \forall\, b, c \in SN,\ a\{x \mapsto b\} \in C\ \Rightarrow\ (\lambda x{:}c.a)\, b \in C$ (a β-redex is reducible if its reduct is reducible and the abstraction variable type is strongly normalizable)
- $\cdot$ $\forall\, a \in C,\ \forall\, b \in Term,\ a \rightsquigarrow b\ \Rightarrow\ b \in C$ (stability by reduction)

Lemma 5.3 SN is a reducibility candidate.

Definition 5.4 Given two sets of terms C and D, we define the set of terms $C \twoheadrightarrow D = \{a \in Term \mid \forall\, b \in C,\ a\, b \in D\}$.

Definition 5.5 (Algebraic type interpretation) We inductively define the *algebraic type interpretation* as follows:

- $\cdot$ $SN(\mathsf{s}) = \{a \in SN \mid$ if $a \rightsquigarrow^* C(b_1, \ldots, b_n)$ $(n \geq 1)$ and $\tau(C) = s_1 \rightarrow \ldots \rightarrow s_n \rightarrow \mathsf{s}$ then $b_i \in SN(s_i)$ $(1 \leq i \leq n)\}$,
- $\cdot$ $SN(s_1 \rightarrow s_2) = SN(s_1) \twoheadrightarrow SN(s_2)$.

Let us justify this definition. Suppose we have a family $(\mathsf{s}_i)_{i \in I}$ of sorts. Let $P = \{X \subseteq SN \mid X$ contains all strongly normalizable terms that do not reduce to a constructor headed term$\}$. It is easy to see that P is a complete lattice for set inclusion. Now, let $F : P^I \rightarrow P^I$, $(X_i)_{i \in I} \mapsto (X'_i)_{i \in I}$ where $X'_i = X_i \cup \{a \in SN \mid a \rightsquigarrow^* C(\vec{b}),\ \tau(C) = t_1 \rightarrow \ldots \rightarrow t_n \rightarrow \mathsf{s}_i$ and $b_k \in X(t_k)$ $(1 \leq k \leq n)\}$, where X is inductively defined as follows: $X(\mathsf{s}_i) = X_i$ and $X(s \rightarrow t) = X(s) \twoheadrightarrow X(t)$. F is clearly a monotone function. Hence, from the Tarski's theorem [Tar55], it has a least fixed point $(\bar{X}_i)_{i \in I}$.

Lemma 5.6 For any algebraic type t, $SN(t)$ is a reducibility candidate.

Definition 5.7 (Admissible sets of reducibility candidates) A set RC of reducibility candidates is *admissible* if it satisfies the following conditions:

- $\cdot$ $SN \in RC$
- $\cdot$ $\forall\, \mathsf{s} \in \mathcal{S},\ SN(\mathsf{s}) \in RC$
- $\cdot$ $\forall\, C, D \in RC,\ C \twoheadrightarrow D \in RC$ (stability by $\twoheadrightarrow$)
- $\cdot$ $\forall\, I \neq \emptyset,\ \forall\, (C_i)_{i \in I} \in RC^I,\ \bigcap_{i \in I} C_i \in RC$ (stability by non empty intersection)

As an example, let us prove that the saturated sets [Tai67] that are stable by reduction is an admissible set of reducibility candidates.

Definition 5.8 (Saturated sets) A set S of terms is *saturated* if:

- $\cdot$ $S \subseteq SN$,
- $\cdot$ for any variable x and strongly normalizable terms $\vec{d}$, $x\, \vec{d} \in S$,
- $\cdot$ for any variable x and strongly normalizable terms $a, b, c, \vec{d}$, if $a\{x \mapsto b\}\, \vec{d} \in S$ then $(\lambda x{:}c.a)\, b\, \vec{d} \in S$,
- $\cdot$ if $a \in S$ and a' is a reduct of a then $a' \in S$.

We denote by SAT the set of all saturated sets.

Lemma 5.9 (SAT admissibility) SAT is an admissible set of reducibility candidates.

Proof. It is easy to see that saturated sets are reducibility candidates. Let us prove that SAT is admissible.

- $\cdot$ It is easy to see that SN and $SN(\mathsf{s})$ $(\mathsf{s} \in \mathcal{S})$ are saturated sets. Indeed, any maximal rewrite sequence of $(\lambda x{:}c.a)\, b$ contains a βh-rewrite, hence conducts to a reduct of $a\{x \mapsto b\}$.
- $\cdot$ If S and T are two saturated sets then $S \twoheadrightarrow T$ is also saturated. First, $S \twoheadrightarrow T \in SN$ since $Var \in S$ and $T \subseteq SN$. Second, $S \twoheadrightarrow T$ contains any term of the form $x\, \vec{d}$ or $(\lambda x{:}c.a)\, b\, \vec{d}$ if $a\{x \mapsto b\}\, \vec{d}$ is itself in $S \twoheadrightarrow T$ since $S \in SN$ and T contains already these terms. Third, $S \twoheadrightarrow T$ is clearly stable by reduction since $Var \in S$ and T is itself stable by reduction.
- $\cdot$ Saturated sets are clearly stable by non empty intersection.

$\diamond$

In the following, RC will refer to any admissible set of reducibility candidates.

Definition 5.10 (Interpretation of the kinds) [Geu95] The interpretation of the kinds is inductively defined as follows, where $\mathcal{F}(X, Y)$ denotes the set theoretic function space from X to Y:

· $[\![\star]\!] = RC$
· $[\![\Pi\alpha{:}K.K']\!] = \mathcal{F}([\![K]\!], [\![K']\!])$
· $[\![\Pi x{:}\tau.K]\!] = [\![K]\!]$
We denote by RC^* the set of all the elements of the kind interpretations: $RC^* = \{x \in [\![K]\!] \mid K \in Kind\}$.

Lemma 5.11 (Substitution property of kind interpretation) [Geu95] If $\Gamma, x{:}a \vdash b : \Box$ and $\Gamma \vdash c : a$ then $[\![b\{x \mapsto c\}]\!] = [\![b]\!]$.

Proof. By induction on the structure of the derivation of $\Gamma, x{:}a \vdash b : \Box$. ◇

Lemma 5.12 (Compatibility of kind interpretation with Γ-equivalence) If a and a' are two Γ-equivalent kinds then $[\![a]\!] = [\![a']\!]$.

Proof. By induction on the kind structure of a. If $a = \star$ then, by the Γ-unexpansivity lemma of $\star$, $a' = \star$ and $[\![a]\!] = [\![a']\!]$. Otherwise, $a = \Pi x{:}b.c$ and $a' = \Pi x{:}b'.c'$ for some terms b, c, b', c' such that $b \overset{*}{\leftrightsquigarrow}_\Gamma b'$ and $c \overset{*}{\leftrightsquigarrow}_\Gamma c'$. Since a and a' are kinds, c and c' are kinds and, as $\overset{*}{\leftrightsquigarrow}_\Gamma$ preserves typing, b is a kind if and only if b' is a kind. So, by induction hypothesis, $[\![c]\!] = [\![c']\!]$ and $[\![b]\!] = [\![b']\!]$ if b is a kind. Therefore, $[\![a]\!] = [\![a']\!]$. ◇

Definition 5.13 (Γ-valuations) [Geu95] A *valuation* is a function from $Var^\Box$ to RC^*. Given an environment Γ, a valuation ξ is a Γ-*valuation*, or is *compatible with* Γ, if $\xi(x) \in [\![\Gamma(x)]\!]$ for each variable $x \in dom(\Gamma) \cap Var^\Box$.

Definition 5.14 (Interpretation of the types) [8] Given an environment Γ and a Γ-valuation ξ, the *interpretation of the types* is inductively defined as follows:
· $[\![\alpha]\!]_\xi = \xi(\alpha)$
· $[\![T\,T']\!]_\xi = [\![T]\!]_\xi([\![T']\!]_\xi)$
· $[\![T\,u]\!]_\xi = [\![T]\!]_\xi$
· $[\![\lambda\alpha{:}K.T]\!]_\xi$ = the function from $[\![K]\!]$ to $\{[\![T]\!]_{\xi,\alpha=r} \mid r \in [\![K]\!]\}$ which associates $[\![T]\!]_{\xi,\alpha=r}$ to each $r \in [\![K]\!]$
· $[\![\lambda x{:}\tau.T]\!]_\xi = [\![T]\!]_\xi$
· $[\![\Pi\alpha{:}K.a]\!]_\xi = [\![K]\!]_\xi \twoheadrightarrow \bigcap_{S \in [\![K]\!]} [\![a]\!]_{\xi,\alpha=S}$
· $[\![\Pi x{:}\tau.a]\!]_\xi = [\![\tau]\!]_\xi \twoheadrightarrow [\![a]\!]_\xi$
· $[\![\star]\!]_\xi = [\![\Box]\!]_\xi = SN$
· $[\![\mathtt{s}]\!]_\xi = SN(\mathtt{s})$

The correctness of that definition is ensured by the following lemma. Indeed, in the $T\,T'$ case, as T has for type a product operating on kinds, $[\![T]\!]_\xi$ must be a function which applies to $[\![T']\!]_\xi$.

Lemma 5.15 (Correctness of the type interpretation) [Geu95] Given an environment Γ and a Γ-valuation ξ,
· if $\Gamma \vdash a : b$ and $\Gamma \vdash b : \Box$ then $[\![a]\!]_\xi \in [\![b]\!]$,
· if $\Gamma \vdash a : \Box$ then $[\![a]\!]_\xi \in RC$.

Proof. The first statement is proved by induction on the structure of the derivation of $\Gamma \vdash a : b$. The second, by induction on the structure of the kind a. ◇

Lemma 5.16 (Reducibility of constructor headed terms) Given an environment Γ, a Γ-valuation ξ and a constructor C whose type is $s_1 \to \ldots \to s_n \to \mathtt{s}$ $(n \geq 0)$, a term $C(a_1, \ldots, a_n) \in [\![\mathtt{s}]\!]_\xi$ if and only if $a_i \in [\![s_i]\!]_\xi$ $(1 \leq i \leq n)$.

Proof. By definition of $[\![\mathtt{s}]\!]_\xi$. ◇

Lemma 5.17 (Substitution property of type interpretation) [Geu95] Given an environment Γ, a Γ-valuation ξ and a constructor or a kind b such that $\Gamma, x{:}a \vdash b : c$ and $\Gamma \vdash d : a$. Then, $[\![b\{x \mapsto d\}]\!]_\xi = [\![b]\!]_{\xi, x=[\![d]\!]_\xi}$ if $x \in Var^\Box$, $[\![b]\!]_\xi$ otherwise.

Proof. By induction on the structure of the derivation of $\Gamma, x{:}a \vdash b : c$. ◇

[8]except for the sorts, it is due to [Geu95]

Lemma 5.18 (Compatibility of type interpretation with Γ-equivalence) If a and a' are two Γ-equivalent type constructors or two Γ-equivalent kinds, then $[\![a]\!]_\xi = [\![a']\!]_\xi$.

Proof. Since $\overset{*}{\leftrightsquigarrow}_\Gamma$ preserves typing, it suffices to prove it for one rewrite, that is, if a and a' are two type constructors or two kinds such that $a \rightsquigarrow a'$, then $[\![a]\!]_\xi = [\![a']\!]_\xi$. By induction on the type derivation of a, the only non straightforward case is that of the application rule when there is a β-rewrite at the root. Then, the previous lemma applies. ◇

Definition 5.19 (Γ, ξ-substitutions) [Geu95] Given an environment Γ and a Γ-valuation ξ, a substitution θ is a *Γ, ξ-substitution*, or is *compatible with* Γ *and* ξ, if $dom(\theta) \subseteq dom(\Gamma)$ and, for each variable $x \in dom(\Gamma)$, $x\theta \in [\![\Gamma(x)]\!]_\xi$.

5.2 Strong normalization proof

Lemma 5.20 (Main lemma for strong normalization) [Geu95] [9] Assume that Γ is an environment, ξ a valuation compatible with Γ and θ a substitution compatible with Γ and ξ. Assume also that, if f is a function symbol of arity $n \geq 0$ and type $s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s$, and $a_1 \ldots a_n$ are terms such that $\Gamma \vdash a_i : s_i$ and $a_i \in [\![s_i]\!]_\xi$ $(1 \leq i \leq n)$, then $f(\vec{a}\theta) \in [\![s]\!]_\xi$. Then, $\Gamma \vdash a : b$ implies that $a\theta \in [\![b]\!]_\xi$.

Proof. By induction on the structure of the derivation of $\Gamma \vdash a : b$.

(ax) $\vdash \star : \Box$

$\star\theta = \star \in [\![\Box]\!]_\xi = SN$ since $\star$ is in normal form.

(sort) $\vdash \mathsf{s} : \star \quad (\mathsf{s} \in \mathcal{S})$

$s\theta = s \in [\![\star]\!]_\xi = SN$ since s is in normal form.

(var) $\dfrac{\Gamma \vdash c : p}{\Gamma, x{:}c \vdash x : c} \quad (x \in Var^p \setminus dom(\Gamma),\ p \in \{\star, \Box\})$

$x\theta \in [\![c]\!]_\xi$ since θ is compatible with Γ and ξ.

(weak) $\dfrac{\Gamma \vdash a : b \quad \Gamma \vdash c : p}{\Gamma, x{:}c \vdash a : b} \quad (x \in Var^p \setminus dom(\Gamma),\ p \in \{\star, \Box\})$

$a\theta \in [\![b]\!]_\xi$ by induction hypothesis since if ξ is compatible with $\Gamma, x{:}c$ then ξ is compatible with Γ, and if θ is compatible with $\Gamma, x{:}c$ and ξ then θ is also compatible with Γ and ξ.

(cons) $\dfrac{\Gamma \vdash a_1 : s_1 \quad \ldots \quad \Gamma \vdash a_n : s_n}{\Gamma \vdash C(a_1, \ldots, a_n) : s} \quad (C \in \mathcal{C}^n,\ \tau(C) = s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s,\ n \geq 0)$

Trivial.

(fun) $\dfrac{\Gamma \vdash a_1 : s_1 \quad \ldots \quad \Gamma \vdash a_n : s_n}{\Gamma \vdash f(a_1, \ldots, a_n) : s} \quad (f \in \mathcal{F}^n_t,\ t = s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s,\ n \geq 0)$

By hypothesis.

(abs) $\dfrac{\Gamma, x{:}a \vdash b : c \quad \Gamma \vdash \Pi x{:}a.c : q}{\Gamma \vdash \lambda x{:}a.b : \Pi x{:}a.c} \quad (x \notin dom(\Gamma),\ q \in \{\star, \Box\})$

$(\lambda x{:}a.b)\theta = \lambda x{:}a\theta.b\theta$. By stripping, $\Gamma \vdash a : p \in \{\star, \Box\}$ and $\Gamma, x{:}a \vdash c : q$. By induction hypothesis, $a\theta \in [\![p]\!]_\xi = SN$.

Suppose that a is a kind. Then, $[\![\Pi x{:}a.c]\!]_\xi = [\![a]\!]_\xi \twoheadrightarrow \bigcap_{S \in [\![a]\!]} [\![c]\!]_{\xi \cup \{x \mapsto S\}}$. Let $S \in [\![a]\!]$. Then, $\xi \cup \{x \mapsto S\}$ is a valuation compatible with $\Gamma, x{:}a$. Let $d \in [\![a]\!]_\xi$. $[\![a]\!]_{\xi \cup \{x \mapsto S\}} = [\![a]\!]_\xi$ since $x \notin FV(a)$. Hence, $\theta \cup \{x \mapsto d\}$ is a substitution compatible with $\Gamma, x : a$ and $\xi \cup \{x \mapsto S\}$. By induction hypothesis, $b\theta\{x \mapsto d\} \in [\![c]\!]_{\xi \cup \{x \mapsto S\}}$. If $q = \star$ then $[\![c]\!]_{\xi \cup \{x \mapsto S\}} \in [\![\star]\!] = RC$. If $q = \Box$ then $[\![c]\!]_{\xi \cup \{x \mapsto S\}} \in RC$ also. Hence, by definition of reducibility candidates, $(\lambda x{:}a\theta.b\theta)\, d \in [\![c]\!]_{\xi \cup \{x \mapsto S\}}$ and $\lambda x{:}a\theta.b\theta \in [\![\Pi x{:}a.c]\!]_\xi$.

Suppose now that a is a type. Then, $[\![\Pi x{:}a.c]\!]_\xi = [\![a]\!]_\xi \twoheadrightarrow [\![c]\!]_\xi$. Let $d \in [\![a]\!]_\xi$. Then, $\theta \cup \{x \mapsto d\}$ is a substitution compatible with $\Gamma, x{:}a$ and ξ. By induction hypothesis, $b\theta\{x \mapsto d\} \in [\![c]\!]_\xi$. If $q = \star$ then $[\![c]\!]_\xi \in [\![\star]\!] = RC$ If $q = \Box$ then $[\![c]\!]_\xi \in RC$ also. Hence, by definition of reducibility candidates, $(\lambda x{:}a\theta.b\theta)\, d \in [\![c]\!]_\xi$ and $\lambda x{:}a\theta.b\theta \in [\![\Pi x{:}a.c]\!]_\xi$.

(app) $\dfrac{\Gamma \vdash a : \Pi x{:}b.c \quad \Gamma \vdash d : b}{\Gamma \vdash a\, d : c\{x \mapsto d\}}$

$(a\, d)\theta = (a\theta)\,(d\theta)$ and, by induction hypothesis, $d\theta \in [\![b]\!]_\xi$.

If b is a kind then $[\![c\{x \mapsto d\}]\!]_\xi = [\![c]\!]_{\xi \cup \{x \mapsto [\![d]\!]_\xi\}}$ and $[\![\Pi x : b.c]\!]_\xi = [\![b]\!]_\xi \twoheadrightarrow \bigcap_{S \in [\![b]\!]} [\![c]\!]_{\xi \cup \{x \mapsto S\}}$. Thus $(a\theta)\,(d\theta) \in \bigcap_{S \in [\![b]\!]} [\![c]\!]_{\xi \cup \{x \mapsto S\}} \subseteq [\![c]\!]_{\xi \cup \{x \mapsto [\![d]\!]_\xi\}}$ since $[\![d]\!]_\xi \in [\![b]\!]$.

If b is a type then $[\![c\{x \mapsto d\}]\!]_\xi = [\![c]\!]_\xi$ and $[\![\Pi x{:}b.c]\!]_\xi = [\![b]\!]_\xi \twoheadrightarrow [\![c]\!]_\xi$. Thus $(a\theta)\,(d\theta) \in [\![c]\!]_\xi$.

[9] in [Geu95], there is no function symbol

(conv) $\dfrac{\Gamma \vdash a:b \quad \Gamma \vdash b':p}{\Gamma \vdash a:b'}$ $(p \in \{\star, \Box\},\ b \rightsquigarrow^*_\beta b'$ or $b' \rightsquigarrow^*_\beta b$ or $b \rightsquigarrow^*_R b'$ or $b' \rightsquigarrow^*_R b)$

By induction hypothesis, $a\theta \in [\![b]\!]_\xi$ and $[\![b]\!]_\xi = [\![b']\!]_\xi$.

(prod) $\dfrac{\Gamma \vdash a:p \quad \Gamma, x{:}a \vdash b:q}{\Gamma \vdash \Pi x{:}a.b:q}$ $(x \notin dom(\Gamma),\ p,q \in \{\star, \Box\})$

$(\Pi x{:}a.b)\theta = \Pi x{:}a\theta.b\theta$ and $[\![q]\!]_\xi = SN$.

By induction hypothesis, $a\theta \in [\![p]\!]_\xi = SN$. Let us prove now that $b\theta \in SN$. By correctness, if $p = \star$ then $[\![a]\!]_\xi \in [\![\star]\!] = RC$, and if $p = \Box$ then $[\![a]\!]_\xi \in RC$.

Suppose that a is a kind and let $S \in [\![a]\!]$. $[\![a]\!]_{\xi \cup \{x \mapsto S\}} = [\![a]\!]_\xi$ since $x \notin FV(a)$. Hence, $\xi \cup \{x \mapsto S\}$ is a valuation compatible with $\Gamma, x{:}a$ and θ is a substitution compatible with $\Gamma, x{:}a$ and $\xi \cup \{x \mapsto S\}$. By induction hypothesis, $b\theta \in [\![q]\!]_{\xi \cup \{x \mapsto S\}} = SN$.

Suppose now that a is a type. Since θ is compatible with Γ and ξ, $dom(\theta) \subseteq dom(\Gamma)$ and $x\theta = x$. Hence, ξ is a valuation compatible with $\Gamma, x{:}a$ and θ is a substitution compatible with $\Gamma, x{:}a$ and ξ. By induction hypothesis, $b\theta \in [\![q]\!]_\xi = SN$.

In both cases, $a\theta \in SN$ and $b\theta \in SN$, hence $\Pi x{:}a\theta.b\theta \in SN$.

◇

Theorem 5.21 (Strong normalization) [Geu95] [10] If the conditions of Lemma 5.20 on the preceding page are satisfied then any well-typed term is strongly normalizable.

Proof. Let a be a term of type b in an environment Γ. Let ϕ be the function inductively defined on kinds as follows:

· $\phi(\star) = SN$,
· $\phi(\Pi x{:}K.K') =$ the constant function from $[\![K]\!]$ to $\{\phi(K')\}$ which associates $\phi(K')$ to each $r \in [\![K]\!]$,
· $\phi(\Pi x{:}\tau.K) = \phi(K)$.

Now, let ξ be a valuation defined as follows: if $\alpha{:}K \in \Gamma$ then $\xi(\alpha) = \phi(K)$. Then it is easy to see that ξ is a valuation compatible with Γ: $\xi(\alpha) \in [\![K]\!]$. Furthermore, the identity substitution is compatible with Γ and ξ: if $x{:}c \in \Gamma$ then $\Gamma \vdash c:p \in \{\star, \Box\}$ and $[\![c]\!]_\xi \in RC$ by correctness of the type interpretation, hence $x \in [\![c]\!]_\xi$. Then, by the main lemma for strong normalization, $a \in [\![b]\!]_\xi$. By correctness of the type interpretation, either $b = \Box$ and $[\![b]\!]_\xi = SN$, or $\Gamma \vdash b:p \in \{\star, \Box\}$ and $[\![b]\!]_\xi \in RC$. Hence, $[\![b]\!]_\xi \subseteq SN$ and a is strongly normalizable. ◇

Hence, we can already set forth all the fundamental properties for the β-reduction alone.

Lemma 5.22 (β-reduction properties) The β-reduction relation enjoys the subject reduction, termination, and confluence properties. Hence any term a has a unique β-normal form (resp. βh-normal form) denoted by $\beta nf(a)$ (resp. $\beta hnf(a)$).

Proof. Subject reduction is proved in Lemma 4.11 on page 12. Strong normalization is easy to see. As we restrict our attention to β-rewrites only, function symbols are trivially reducible since no rewrite can occur at the root of a function headed term. Hence Theorem 5.21 applies. Finally, since the β-reduction is strongly normalizable and locally confluent, it is confluent. ◇

5.3 Reducibility of first-order function symbols

As we are going to use well-founded inductions to prove the reducibility of the function symbols, we recall hereafter the necessary background about well-founded orderings. The interested reader may find more details in [DJ90].

Definition 5.23 (Orderings)

· An order is *well-founded* if there is no strictly decreasing infinite chain.
· An order $>$ on terms is *monotonic* if it is stable by substitution and context: $a > b \Rightarrow c[a\theta]_m > c[b\theta]_m$.
· If $>_1$ and $>_2$ are two orders then $> = (>_1, >_2)_{lex}$ is their associated strict lexicographic ordering: $(a_1, a_2) > (b_1, b_2)$ if $a_1 >_1 b_1$ or $a_1 =_1 b_1$ and $a_2 >_2 b_2$. If $>_1 = >_2 = >$ then it is denoted $>_{lex}$. This definition is naturally extended to n-uples.

[10] in [Geu95], there is no function symbol

· If $>$ is an order then $>_{mul}$ is its associated strict multi-set ordering. It is the transitive closure of the following relation: $M \cup \{a\} >_{mul} M \cup \{b_1, \ldots, b_n\}$ $(n \geq 0)$ if $a > b_i$ $(1 \leq i \leq n)$.

Lemma 5.24 (Well-founded orderings properties)
· If $>_1$ and $>_2$ are two well-founded orderings then so is $(>_1, >_2)_{lex}$.
· If $>$ is a well-founded ordering then so is $>_{mul}$.
· If $>$ is a well-founded monotonic ordering on terms then so is $(> \cup \rhd)$ where $\lhd$ is the strict subterm ordering.

Definition 5.25 (Cap and aliens) [JO97a] Given an injection ψ from $Term$ to $Var^\star$, the *cap of a term* a, denoted $cap(a)$, is the first-order algebraic term inductively defined as follows:
· $cap(C(a_1, \ldots, a_n)) = C(cap(a_1), \ldots, cap(a_n))$
· $cap(f(a_1, \ldots, a_n)) = f(cap(a_1), \ldots, cap(a_n))$ if $f \in \mathcal{F}_1$,
· $cap(a) = \psi(a)$ otherwise.
So, $a = cap(a)\{\vec{x} \mapsto \vec{b}\}$ where $x_i = \psi(b_i)$ $(1 \leq i \leq |\vec{x}|)$. The terms in $\vec{b}$ are called the *aliens* of a and their multi-set is denoted by $aliens(a)$.

Lemma 5.26 (Reducibility of first-order function symbols) [JO97a] [11] Assume that:
· the rules of R_1 are conservative,
· $\rightsquigarrow_{R_1}$ terminates on well-typed first-order algebraic terms.
Then, for any environment Γ, valuation ξ compatible with Γ, first-order function symbol $f \in \mathcal{F}_t^n$ such that $t = \mathsf{s_1} \to \ldots \to \mathsf{s_n} \to \mathsf{s}$ and terms a_i $(1 \leq i \leq n)$, if $\Gamma \vdash a_i : \mathsf{s}_i$ and $a_i \in [\![\mathsf{s}_i]\!]_\xi$ $(1 \leq i \leq n)$ then $f(a_1, \ldots, a_n) \in [\![\mathsf{s}]\!]_\xi$.

Proof. As f is a first-order function symbol, $\mathsf{s} \in \mathcal{S}$ and $[\![\mathsf{s}]\!]_\xi = SN(\mathsf{s})$. Let $b = f(a_1, \ldots, a_n)$. Then $b = cap(b)\{\vec{x} \mapsto \vec{d}\}$ where $\{\vec{d}\} = aliens(b)$.

Let us prove that any term b of type s such that the terms of $aliens(b)$ are strongly normalizable is in $SN(\mathsf{s})$ by induction on $(aliens(b), cap(b))$ with $((\rightsquigarrow^* \cup \unrhd)_{mul}, \rightsquigarrow_{R_1}^*)_{lex}$ standing for our order. This order is well-founded since the aliens are strongly normalizable and $\rightsquigarrow_{R_1}$ terminates on well-typed first-order algebraic terms.

It suffices to prove that any reduct of b is in $SN(\mathsf{s})$. If b is reducible then either a term in $aliens(b)$ is reducible or $cap(b)$ is R_1-reducible. Indeed, there cannot be β-rewrites in a first-order algebraic term and, by definition of first-order algebraic terms, there cannot be rewrites whose redex covers both $cap(b)$ and $aliens(b)$. Let b' be a reduct of b at position $p \in Pos(b)$.

If $p \in Pos(cap(b))$ then $b' = c'\{\vec{x} \mapsto \vec{d}\}$ such that $cap(b') = c'$ and $cap(b) \overset{p}{\rightsquigarrow}_{R_1} c'$. As the first-order rules are conservative, $aliens(b') \subseteq aliens(b)$ and the induction hypothesis applies.

If $p \notin Pos(cap(b))$ then there exists i $(1 \leq i \leq n)$ and a term e such that $d_i \rightsquigarrow e$ and $b' = c\{\vec{x} \mapsto \vec{d'}\}$ where $d'_i = e$ and $d'_j = d_j$ $(1 \leq j \leq n, j \neq i)$. Then $cap(b') = cap(b)\{x_i \mapsto cap(e)\}$ and $aliens(b') = (aliens(b) \setminus \{d_i\}) \bigcup aliens(e)$. But $\{d_i\} (\rightsquigarrow^+ \cup \rhd)_{mul}\, aliens(e)$ thus the induction hypothesis applies. $\diamond$

If there is no higher-order rewrite rule then the conservativity hypothesis can be dropped. In this case, the proof must be done by induction on the pair made of the cap of the β-normal form of a, called its *extended cap*, (the strong normalization and confluence of the β-reduction alone can be proved beforehand; see Lemma 5.22 on the previous page), and its aliens in βR-normal forms (the aliens are supposed strongly normalizable) [JO97a]. Indeed, higher-order rules can introduce new redexes which may increase the cap via their higher-order variables. Therefore, this induction argument does not hold when there are such variables, hence the need for conservativity in this case.

5.4 Reducibility of higher-order function symbols

The purpose of this section is to define a class of higher-order rewrite rules for which we are able to prove their well-foundedness. It will have to include the recursors for strictly positive sorts.

Given a lefthand side rule term, the idea is to define a set of admissible righthand side rule terms built from the reducible subterms of the lefthand side.

First, we begin by distinguishing a subclass of terms that will play a key role in the definition of the higher-order rewrite rules schema. This subclass is intimately related to the structure of positive sorts.

[11] called "Principal Case" in [JO97a]

Definition 5.27 (Γ,s-terms) Given an algebraic type s and an environment Γ, a term a is a Γ,*s-term* if it is typable in Γ by an algebraic type in which s occurs positively.

Definition 5.28 (Γ,s-subterm relation) Given an algebraic type s, an environment Γ and two terms a and b, b is a Γ,*s-subterm* of a, $a \trianglerighteq_{\Gamma,s} b$, if and only if $a \trianglerighteq b$ and each superterm of b in a is a Γ,s-term.

Given a reducible term, not all its subterms are reducible. Hence, we define hereafter a relation that catches some reducible subterms. Together with the variables of basic sort, these terms will be the reducible subterms from which admissible righthand sides will be built.

Definition 5.29 (Term accessibility) [JO97b] Given two rule terms a and b, a is *accessible* in b if:

- $a = b$,
- $b = C(\vec{d})$ and a is accessible in one of the terms of $\vec{d}$,
- $b = \lambda x{:}t.d$ and a is accessible in d ($x \notin FV(a)$).

Lemma 5.30 (Compatibility of accessibility with reducibility) Given two rule terms a, b, an extended algebraic type s, an extended algebraic environment Γ, a valuation ξ compatible with Γ and a substitution θ, if $\Gamma \vdash b : s$, $b\theta \in [\![s]\!]_\xi$ and a is accessible in b then there is an extended algebraic type t such that $\Gamma \vdash a : t$ and $a\theta \in [\![t]\!]_\xi$.

Proof. By induction on the structure of the proof that a is accessible in b. If $b = a$ then this is trivial. Suppose that $b = C(\vec{d})$, $\tau(C) = s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s$ and a is accessible in d_i. Then $b\theta = C(\vec{d}\theta)$ and, since $b\theta \in [\![s]\!]_\xi$, by definition of $[\![s]\!]_\xi$, $d_i\theta \in [\![s_i]\!]_\xi$. Hence, the induction hypothesis applies. Suppose now that $b = \lambda x{:}t.d$ and a is accessible in d. Then $s = t \rightarrow t'$, $\Gamma, x{:}t \vdash d : t'$ and $b\theta = \lambda x{:}t.d\theta$. As $x \in [\![t]\!]_\xi$, $(b\theta)\, x \in [\![t']\!]_\xi$ and, since reducibility candidates are stable by reduction, $d\theta \in [\![t']\!]_\xi$. Thus, the induction hypothesis applies. $\diamond$

Lemma 5.31 (Basic inductive sorts) [JO97b] For any environment Γ, Γ-valuation ξ and basic sort $\mathtt{s}$, $[\![\mathtt{s}]\!]_\xi = SN$.

Proof. By definition, $[\![\mathtt{s}]\!]_\xi \subseteq SN$. Let us prove by induction on SN with $(\rightsquigarrow^* \cup \trianglerighteq)$ as order that $SN \subseteq [\![\mathtt{s}]\!]_\xi$. Let $a \in SN$ and suppose that $a \rightsquigarrow^* C(\vec{b})$ where $C \in \mathcal{C}(\mathtt{s})$. Since $\mathtt{s}$ is a basic inductive sort, $\tau(C) = \mathtt{s}_1 \rightarrow \ldots \rightarrow \mathtt{s}_n \rightarrow \mathtt{s}$ where every $\mathtt{s}_i$ is also a basic inductive sort. As a is strongly normalizable, every b_i is also strongly normalizable. Hence, by induction hypothesis, every $b_i \in [\![\mathtt{s}_i]\!]_\xi$ and $a \in [\![\mathtt{s}]\!]_\xi$. $\diamond$

Now, let us have a look at the third rule of the recursor for `ord`, which is a typical example of strictly positive sort:

$$\mathtt{rec}^t_{\mathtt{ord}}(\mathtt{lim}_{\mathtt{ord}}(f),u,v,w) \;\rightsquigarrow\; w\ f\ \lambda n{:}\mathtt{nat}.\mathtt{rec}^t_{\mathtt{ord}}(f\,n,u,v,w)$$

To prove the strict decreasingness of the arguments of the recursive call, we would like to say that $\mathtt{lim}_{\mathtt{ord}}(f)$ "$\triangleright$" $f\,n$. The idea is then to neglect the term n in comparison to f: f is "critical" in a precise sense defined below. Hence, in a recursive call, the actual arguments of f will not be compared with the arguments of the recursive call, but instead with the critical subterms of the arguments of its recursive call.

Definition 5.32 (Application decomposition) A term a can always be written $a_1 \ldots a_n$ ($n \geq 1$) where a_1 is not an application. $a_1 \ldots a_n$ is called the *application decomposition* of a.

Definition 5.33 (Γ,s-critical subterm) [12] Assume that s is an algebraic type, Γ an environment and a a Γ,s-term whose application decomposition is $a_1 \ldots a_n$ ($n \geq 1$). Then, the Γ,*s-critical subterm* of a, $\chi^s_\Gamma(a)$, is its smallest subterm $a_1 \ldots a_k$ ($1 \leq k \leq n$) such that $a \trianglerighteq_{\Gamma,s} a_1 \ldots a_k$.

Such a smallest subterm always exists since the definition works at least for $k = n$.

In order to manage efficiently the fact that our function symbols have several arguments, we define hereafter a class of well-founded orderings on sequences of terms that mix lexicographic and multi-set orderings. Its purpose is to enable users to do complex comparisons when trying to prove the decreasingness of the arguments of a recursive call.

[12] notion first introduced by [JO97b] with a different, more involved formulation

Definition 5.34 (Status orderings) [13] A *status* of arity n ($n \geq 0$) is a term of the form $lex(t_1, \ldots, t_p)$ ($p \geq 0$) where t_i ($1 \leq i \leq p$) is either x_j ($1 \leq j \leq n$) or a term of the form $mul(x_{k_1}, \ldots, x_{k_q})$ ($1 \leq k_l \leq n$, $1 \leq l \leq q$) such that each variable x_i ($1 \leq i \leq n$) occurs at most once. A position $i \in \{1 \ldots n\}$ is *lexicographic* if there exists $j \in \{1 \ldots p\}$ such that $t_j = x_i$. A *status term* is a status whose variables are substituted by terms of CAIC.

Let *stat* be a status of arity $n \geq 0$, $I = \{i_1, \ldots, i_k\}$ a subset of the lexicographic positions of *stat*, called *inductive positions*, $S = (>^{i_1}, \ldots, >^{i_k})$ a sequence of orders on terms indexed by I, and $>$ an order on terms. Then, we define the corresponding *status ordering*, $>^S_{stat}$ on sequences of terms as follows:

- $(a_1, \ldots, a_n) >^S_{stat} (b_1, \ldots, b_n)$ iff $stat\{\vec{x} \mapsto \vec{a}\} >^S_{stat} stat\{\vec{x} \mapsto \vec{b}\}$,
- $lex(c_1, \ldots, c_p) >^S_{lex(t_1,\ldots,t_p)} lex(d_1, \ldots, d_p)$ iff $(c_1, \ldots, c_p)\, (>^S_{t_1}, \ldots, >^S_{t_p})_{lex}\, (d_1, \ldots, d_p)$,
- $>^S_{x_i} = >^i$ if $i \in I$, $>$ otherwise,
- $mul(c_1, \ldots, c_q) >_{mul(x_{k_1},\ldots,x_{k_q})} mul(d_1, \ldots, d_q)$ iff $\{c_1, \ldots, c_q\} >_{mul} \{d_1, \ldots, d_q\}$.

Note that it boils down to the usual lexicographic ordering if $stat = lex(x_1, \ldots, x_n)$ or to the multi-set ordering if $stat = lex(mul(x_1, \ldots, x_n))$. It is easy to see that if $>, >^{i_1}, \ldots, >^{i_k}$ are well-founded then so is $>^S_{stat}$.

Definition 5.35 (Function status and inductive positions) [JO97b] From now on, we will assume that every function symbol $f \in \mathcal{F}^n_t$ has an associated status $stat_f$ of arity n and an associated set of *inductive positions* $Ind(f) \subseteq \{1 \ldots n\}$ that are lexicographic positions in $stat_f$.

Inductive positions are used in case of recursive definitions and correspond to the arguments on which the induction is done.

Definition 5.36 (Critical interpretation) [JO97b] Given an environment Γ and a function symbol f of arity $n \geq 0$ and type $s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s$, we define the *critical interpretation function* $\phi_{f,\Gamma}$ as follows:

- $\phi_{f,\Gamma}(a_1, \ldots, a_n) = (\phi^1_{f,\Gamma}(a_1), \ldots, \phi^n_{f,\Gamma}(a_n))$,
- $\phi^i_{f,\Gamma}(a_i) = a_i$ if $i \notin Ind(f)$,
- $\phi^i_{f,\Gamma}(a_i) = \chi^{s_i}_\Gamma(a_i)$ if $i \in Ind(f)$.

Definition 5.37 (Critical ordering) Let $f(\vec{c}) \rightsquigarrow e$ be an admissible rewrite rule where f is a function symbol of arity $n \geq 0$ and type $s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s$, and $S = (\rhd_{\Gamma_{f(\vec{c})}, s_i})_{i \in Ind(f)}$. Then, we define the *critical ordering* of f, $>_f$, as being $\rhd^S_{stat_f}$.

Besides, in the case where $Ind(f) = \{1\}$ and $stat_f = lex(x_1)$, given a rule term u and a position m in u, we inductively define the *extended critical ordering* of f, $>^{u,m}_f$, as follows: $c_1 >^{u,m}_f c'_1$ iff

- $c_1 \rhd_{\Gamma, s_1} \chi^{s_1}_{\Gamma_{f(\vec{c})}}(c'_1)$,
- or $\chi^{s_1}_{\Gamma_{f(\vec{c})}}(c'_1) = y_i \notin FV(c_1)$, $u = K_1[\lambda x{:}t.K_2[a\,\lambda\vec{y}{:}\vec{t}.x\,\vec{b}]_{m_2}]_{m_1}$, $b_k = \lambda\vec{z}{:}\vec{t'}.u_{|m}$, a has no variable bound in K_2, and $c_1 >^{u,m'}_f a$, where m' is the position of a in u.

Definition 5.38 (General schema) [14] An admissible rewrite rule $f(\vec{c}) \rightsquigarrow e$ where $\tau(f) = s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s$ satisfies the *general schema* if and only if:

- $\forall\, i \in Ind(f)$, $\chi^{s_i}_\Gamma(c_i) = c_i$ and $c_i \notin Var$,
- $e \in RHS_f(\vec{c})$

where $RHS_f(\vec{c})$ is the least set of rule terms containing:

- every variable of $Var^\star \setminus FV(\vec{c})$,
- every free variable of $\vec{c}$ whose type in $\Gamma_{f(\vec{c})}$ is a basic inductive sort,
- every term a accessible in $\vec{c}$ such that if a is reducible then every subterm of a is also reducible,

and closed under the following operations, assuming that the terms on which they are applied are typable in some compatible extensions of $\Gamma_{f(\vec{c})}$:

- **construction:** given a constructor $C \in \mathcal{C}^n$ ($n \geq 0$) such that $\tau(C) = t_1 \rightarrow \ldots \rightarrow t_p \rightarrow t$, if $e_i \in RHS_f(\vec{c})$ ($1 \leq i \leq p$) is of type t_i, then $C(e_1, \ldots, e_p) \in RHS_f(\vec{c})$ and is of type t,
- **defined application:** given a function symbol $g <_\mathcal{F} f$ of type $t_1 \rightarrow \ldots \rightarrow t_p \rightarrow t$, if $e_i \in RHS_f(\vec{c})$ ($1 \leq i \leq p$) is of type t_i, then $g(e_1, \ldots, e_p) \in RHS_f(\vec{c})$ and is of type t,

[13] inspired from [JO97a]

[14] inspired from the new "General schema" of Jouannaud and Okada [JO97b]

· **application:** if $u \in RHS_f(\vec{c})$ is of type $t \to t'$ and $v \in RHS_f(\vec{c})$ is of type t, then $u\,v \in RHS_f(\vec{c})$ and is of type t',
· **abstraction:** if $u \in RHS_f(\vec{c})$ is of type t' then $\lambda x{:}t.u \in RHS_f(\vec{c})$ and is of type $t \to t'$,
· **reduction:** if $u \in RHS_f(\vec{c})$ is of type t and $u \rightsquigarrow u'$ then $u' \in RHS_f(\vec{c})$ and is of type t,
· **admissible recursive call:**
  · **direct case:** if $\vec{c'}$ are n terms of $RHS_f(\vec{c})$ of respective type $s_1, \ldots, s_n$, and $\vec{c} >_f \phi_{f,\Gamma}(\vec{c'})$, then $f(\vec{c'}) \in RHS_f(\vec{c})$ and is of type t.
  · **indirect case:** if $Ind(f) = \{1\}$, $stat_f = lex(x_1)$, $u \in RHS_f(\vec{c})$ is of type t, $m \in Pos(u)$, $m_{|m}$ is of type s, $\vec{c'}$ are n terms of $RHS_f(\vec{c})$ of respective types $s_1, \ldots, s_n$, and $c_1 >_f^{u,m} \chi^{s_1}_{\Gamma_{f(\vec{c})}}(c'_1)$, then $u[f(\vec{c'})]_m \in RHS_f(\vec{c})$ and is of type t.

Lemma 5.39 (Strictly positive recursors) The recursor rules for strictly positive inductive sorts (Definition 2.20 on page 7) satisfy the general schema.

Proof. Let s be a strictly positive inductive sort and t an algebraic type. The rules for rec^{s}_t are clearly well-typed, algebraic and type-preserving (see Section 8 on page 28), hence they are admissible. Besides, $\chi^{\mathsf{s}}_\Gamma(C_i(\vec{a})) = C_i(\vec{a})$ and it easy to see that the righthand side $b_i\,\vec{a}\,\vec{b'}$ belong to $RHS_{rec^{\mathsf{s}}_t}(C_i(\vec{a}), \vec{b})$.

Let us take $Ind(rec^{\mathsf{s}}_t) = \{1\}$ and $stat_{rec^{\mathsf{s}}_t} = lex(x_1)$. The first argument of a recursive call is of the form $(a_j\,\vec{x})$ where $\vec{x}$ are bound variables. Hence, $\chi^{\mathsf{s}}_\Gamma(a_j\,\vec{x}) = a_j$ and $C_i(\vec{a}) \rhd_{\Gamma,\mathsf{s}} a_j$ since $C_i(\vec{a})$ has type s and a_j has type $s^i_1 \to \ldots \to s^i_{n_i} \to \mathsf{s}$ where s^i_k are built from strictly positive inductive sorts strictly smaller than s in $\geq_{\mathcal{S}}$. $\diamond$

Lemma 5.40 (Positive recursors) The recursor rules for positive inductive sorts (Definition 2.20 on page 7) satisfy the general schema.

Proof. $\diamond$

Lemma 5.41 (Well-typedness of righthand sides) Given an admissible rewrite rule $f(\vec{c}) \rightsquigarrow e$ following the general schema, there exists an algebraic type s such that $\Gamma_{f(\vec{c})} \vdash e{:}s$.

Proof. By induction on the structure of $e \in RHS_f(\vec{c})$. e cannot be a variable of $Var^\star \setminus FV(\vec{c})$ since $FV(e) \subseteq FV(\vec{c})$. Variables of $Var^\star \setminus FV(\vec{c})$ are introduced through abstractions in which their type is an algebraic one. The case where e is accessible in c_i has already been treated in Lemma 5.30 on page 19. Besides, all the other constructions are well-typed. $\diamond$

Finally, we give a proof of reducibility for higher-order function symbols that satisfy the general schema. The proof relies on the definition of an interpretation of recursive calls whose abstract properties are defined below. We will then be left with the problem of finding such an interpretation.

Definition 5.42 (Admissible recursive call interpretation) Given a function symbol f of arity $n \geq 0$, a *recursive call interpretation* is given by:
· a status ordering $\geq^S_{stat_f}$ where $\geq$ is an order on terms and $S = (\geq^i)_{i \in Ind(f)}$ is a sequence of orders on terms indexed by $Ind(f)$,
· an interpretation function $\Phi_{f,\Gamma} = (\Phi^1_{f,\Gamma}, \ldots, \Phi^n_{f,\Gamma})$ for each environment Γ.

A recursive call interpretation is *admissible* if it satisfies the following properties:

(Well-foundedness) Assume that $f(\vec{c}) \rightsquigarrow e$ is an admissible rewrite rule following the general schema, $f(\vec{c'})$ is a subterm of e such that $\vec{c} >_f \phi_{f,\Gamma}(\vec{c'})$, and θ is a $\Gamma_{f(\vec{c})}$-substitution. Then, $\Phi_{f,\Gamma}(\vec{c}\theta) >^S_{stat_f} \Phi_{f,\Gamma}(\vec{c'}\theta)$.

(Compatibility) Assume that s is the output type of f, $\vec{a}$ and $\vec{a'}$ are two strongly normalizable sequences of terms such that $\Gamma \vdash f(\vec{a}){:}s$ and $\vec{a} \rightsquigarrow^* \cup \trianglerighteq \vec{a'}$. Then, $\Phi_{f,\Gamma}(\vec{a}) \geq^S_{stat_f} \Phi_{f,\Gamma}(\vec{a'})$.

Lemma 5.43 (Reducibility of higher-order function symbols) Assume that:
· the rewrite rules are admissible (Definition 2.17 on page 5),
· the function ordering $>_{\mathcal{F}}$ is well-founded on $\mathcal{F}_\omega$ (excluding mutually recursive definitions for higher-order function symbols),

- the higher-order rules satisfy the general schema (Definition 5.38 on page 20),
- the first-order function symbols are reducible,
- for each higher-order function symbol f, there exists an admissible recursive call interpretation $(\geq^S_{stat_f}, \Phi_{f,\Gamma})$.

Then, for any environment Γ, Γ-valuation ξ, higher-order function symbol $f \in \mathcal{F}^n_t$ of type $t = s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s$, and sequence of n terms $\vec{a}$, if $\Gamma \vdash a_i : s_i$ and $a_i \in [\![s_i]\!]_\xi$ $(1 \leq i \leq n)$ then $f(\vec{a}) \in [\![s]\!]_\xi$.

Proof. The proof is done with three levels of induction: on the function symbols, on the sequence of terms to which f is applied and on the righthand side structure of the defining rules of f.

As the definition order is well-founded, we can reason by induction on it. So, we suppose that any other symbol appearing in the defining rules of f have the desired property.

There exists $\mathtt{s}' \in \mathcal{S}$ such that $s = s_{n+1} \rightarrow \ldots \rightarrow s_{n+k} \rightarrow \mathtt{s}'$ $(k \geq 0)$. Hence $[\![s]\!]_\xi = [\![s_{n+1}]\!]_\xi \twoheadrightarrow \ldots \twoheadrightarrow [\![s_{n+k}]\!]_\xi \twoheadrightarrow [\![\mathtt{s}']\!]_\xi$ and $f(\vec{a}) \in [\![s]\!]_\xi$ if and only if, for any terms $d_1, \ldots, d_k$ such that $d_i \in [\![s_{n+i}]\!]_\xi$ $(1 \leq i \leq k)$, $f(\vec{a})\,\vec{d} \in SN(\mathtt{s}')$. Let us prove this property by induction on $(\vec{d}, \Phi_{f,\Gamma}(\vec{a}), \vec{a})$ with $(\,(\rightsquigarrow^*)_{lex}\,, \geq^S_{stat_f}, (\rightsquigarrow^*)_{lex}\,)_{lex}$ as well-founded order.

Let $b = f(\vec{a})\,\vec{d}$. It suffices to prove that every reduct b' of b is in $SN(\mathtt{s}')$.

If $f(\vec{a})$ is not reduced at its root then either one a_i or one d_i is reduced. Thus $b' = f(\vec{a'})\,\vec{d'}$ such that $\vec{a} \rightsquigarrow \vec{a'}$ or $\vec{d} \rightsquigarrow \vec{d'}$. As reducibility candidates are stable by reduction, $a'_i \in [\![s_i]\!]_\xi$ $(1 \leq i \leq n)$ and $d'_i \in [\![s_{n+i}]\!]_\xi$ $(1 \leq i \leq k)$. If one a_i is reduced, by compatibility of the interpretation, $\Phi_{f,\Gamma}(\vec{a}) \geq^S_{stat_f} \Phi_{f,\Gamma}(\vec{a'})$. Hence, in both cases, the induction hypothesis applies.

If $f(\vec{a})$ is reduced at its root then $\vec{a} = \vec{c}\theta$ and $b' = e\theta\,\vec{d}$ for some terms $\vec{c}, e$ and substitution θ such that $f(\vec{c}) \rightsquigarrow e$ is the applied rule. Since rewrite rules are admissible, there exists a unique algebraic environment $\Gamma_{f(\vec{c})} \overset{*}{\leftrightsquigarrow}_\Gamma \Gamma|_{FV(\vec{c})}$. Hence ξ is a $\Gamma_{f(\vec{c})}$-valuation.

Let $x \in FV(e)$. If x is a free variable of c_i whose type $\Gamma_{f(\vec{c})}(x)$ is a basic sort then $x\theta \in [\![\Gamma_{f(\vec{c})}(x)]\!]_\xi$ since $[\![\Gamma_{f(\vec{c})}(x)]\!]_\xi = SN$ and $x\theta$ is a subterm of a_i which is strongly normalizable. Otherwise, x is accessible in c_i. Since $\Gamma_{f(\vec{c})} \vdash c_i : s_i$ and $c_i\theta = a_i \in [\![s_i]\!]_\xi$, by compatibility of accessibility with reducibility, $x\theta \in [\![\Gamma_{f(\vec{c})}(x)]\!]_\xi$. Hence θ is compatible with $\Gamma_{f(\vec{c})}$ and ξ. By compatibility of type interpretation with Γ-equivalence, θ is compatible with Γ and ξ.

Then, let us show by induction on the structure of $e \in RHS_f(\vec{c})$ that, if $\Gamma \vdash e : t$ then, for any Γ, ξ-substitution θ, $e\theta \in [\![t]\!]_\xi$.

- $e = x \in Var^\star \setminus FV(\vec{c})$: By compatibility of θ with Γ and ξ, $e\theta \in [\![t]\!]_\xi$.
- e is a free variable of c_i whose type is a basic sort: By compatibility of θ with Γ and ξ, $e\theta \in [\![t]\!]_\xi$.
- e is accessible in c_i: By compatibility of accessibility with reducibility, $e\theta \in [\![t]\!]_\xi$.
- construction: $e = C(e_1, \ldots, e_p)$ and $\tau(C) = t_1 \rightarrow \ldots \rightarrow t_p \rightarrow \mathtt{t}$. By induction hypothesis on e_i, $e_i\theta \in [\![t_i]\!]_\xi$. Hence $e\theta \in [\![\mathtt{t}]\!]_\xi$.
- defined application: $e = g(e_1, \ldots, e_p)$ where $\tau(g) = t_1 \rightarrow \ldots \rightarrow t_p \rightarrow t$ and $g <_\mathcal{F} f$. By induction hypothesis, $e_i\theta \in [\![t_i]\!]_\xi$. Hence, $e\theta \in [\![t]\!]_\xi$, by hypothesis on first-order function symbols, or by outer induction hypothesis on higher-order function symbols since $g <_\mathcal{F} f$.
- application: $e = u\,v$. By induction hypothesis on u and v, $u\theta \in [\![t' \rightarrow t]\!]_\xi$ and $v\theta \in [\![t']\!]_\xi$. Hence $e\theta \in [\![t]\!]_\xi$.
- abstraction: $e = \lambda x{:}t_1.u$ and $t = t_1 \rightarrow t_2$ such that $\Gamma, x{:}t_1 \vdash u : t_2$. Let $v \in [\![t_1]\!]_\xi$. By induction hypothesis on u, $u\theta\{x \mapsto v\} \in [\![t_2]\!]_\xi$. Hence $(\lambda x{:}t_1.u\theta)\,v \in [\![t_2]\!]_\xi$ and $e\theta \in [\![t]\!]_\xi$.
- reduction: e is a reduct of a term $u \in RHS_f(\vec{c})$. As terms of $RHS_f(\vec{c})$ are typable by algebraic types in $\Gamma_{f(\vec{c})} \overset{*}{\leftrightsquigarrow}_\Gamma \Gamma|_{FV(\vec{c})}$, and algebraic types are Γ-consistent, $\Gamma \vdash u : t$ and, by induction hypothesis on u, $u\theta \in [\![t]\!]_\xi$. Since reducibility candidates are stable by reduction, $e\theta \in [\![t]\!]_\xi$.
- admissible recursive call:
  - direct case: $e = f(\vec{c'})$ and $\vec{c} >_f \phi_{f,\Gamma}(\vec{c'})$. By well-foundedness of the recursive call interpretation, $\Phi_{f,\Gamma}(\vec{c}\theta) >^S_{stat_f} \Phi_{f,\Gamma}(\vec{c'}\theta)$. Hence, the outer induction hypothesis on $\vec{c'}\theta$ applies.
  - indirect case: $Ind(f) = \{1\}$, $stat_f = lex(x_1)$, $e = u[f(\vec{c'})]_m$ and $c_1 >^{u,m}_f \chi^{s_1}_\Gamma(c'_1)$. Let us prove the result by induction on the proof that $c_1 >^{u,m}_f \chi^{s_1}_\Gamma(c'_1)$
    - $c_1 >_f \chi^{s_1}_\Gamma(c'_1)$: Then, $\vec{c} >_f \phi_{f,\Gamma}(\vec{c'})$ and, by well-foundedness of the recursive call interpretation, $\Phi_{f,\Gamma}(\vec{c}\theta) >^S_{stat_f} \Phi_{f,\Gamma}(\vec{c'}\theta)$. Hence, the outer induction hypothesis on $\vec{c'}\theta$ applies and $f(\vec{c'}\theta) \in [\![s]\!]_\xi$. Besides, it is easy to see that $e\theta = u\theta[f(\vec{c'}\theta)]_m \in [\![t]\!]_\xi$, since, by induction hypothesis on u, every subterm of u is reducible. Hence, replacing a subterm of $u\theta$ by a reducible term with the good type at such a position gives a reducible term.
    - $\chi^{s_1}_\Gamma(c'_1) = y_i \notin FV(c_1)$, $u = K_1[\lambda x{:}t.K_2[a\,\lambda\vec{y}{:}\vec{t}.x\,\vec{b}]_{m_2}]_{m_1}$, $b_k = \lambda\vec{z}{:}\vec{t'}.u_{|m}$, a has no variable bound in K_2, and $c_1 >^{u,m'}_f a$, where m' is the position of a in u.

      Let $v = \lambda x{:}t.K_2[a\,\lambda\vec{y}{:}\vec{t}.x\,\vec{b}]_{m_2}$ and R be the set of reducts w' of $v\theta[f(\vec{c'}\theta)]_p$ $(m = m_1 \cdot p')$ such that if w' is reduced then one y_i is instanciated. Such reducts can be obtained by replacing subterms of reducts of $v\theta$ by reducts of $f(\vec{c'}\theta)$. Indeed, since $f(\vec{c'}\theta)$ is an argument of x which cannot be

reduced or instanciated, there cannot be interactions between the reductions of $v\theta$ and those of $f(\vec{c'}\theta)$.
By induction hypothesis on u and $\vec{c'}$, $u\theta$ and $\vec{c'}\theta$ are strongly normalizable. Hence, $v\theta$ is strongly normalizable too. Besides, since, by definition of the general schema, $c_1 \notin Var$, a reduct of $f(\vec{c'}\theta)$ is of the form $f(\vec{d})$ such that $\vec{c'}\theta \leadsto^* \vec{d}$. Hence, $f(\vec{c'}\theta)$ is strongly normalizable.
Therefore, R is a finite set. A reduct of an element of R is either in normal form or else of the form $v'[x\,\vec{b'}]_{p'}$ where $b'_k = \lambda\vec{z}{:}\vec{t'}.f(c''_1, \vec{d})$ and c''_1 is a subterm of a reduct of $a\theta$, since a is supposed to have no variable bound in K_2.
In the first case, if it is a constructor headed term then it must be a reduct of $u\theta$, hence it is reducible. In the other case, by compatibility of the recursive call interpretation and by induction hypothesis on $c_1 >_f^{u,m'} a$, we obtain that $v'[x\,\vec{b'}]_{p'}$ is reducible. Hence, $K_1\theta[v'[x\,\vec{b'}]_{p'}]_{m_1}$ is reducible.

◇

5.5 Recursive call interpretation

In this section, we define a recursive call interpretation and prove that it is admissible, hence proving the reducibility of higher-order function symbols satisfying the general schema.

Since the interpretation must be compatible with the reduction relation, the idea is to interpret a term by its reducts. Besides, since the comparisons are done with critical subterms, we need to erase all subterms that are not Γ,s-terms. For these two ideas to be compatible, we need to take into account only those reducts obtained via rewrites at "positive positions".

Definition 5.44 (Γ,s-reduction relation) Given an algebraic type s and an environment Γ, we define the Γ,s-*rewrite relation* on Γ,s-*terms* as follows:

- $a \overset{\epsilon}{\leadsto}_{\Gamma,s} a'$ if and only if $a \overset{\epsilon}{\leadsto} a'$.
- $C(\vec{a}) \overset{i\cdot m}{\leadsto}_{\Gamma,s} C(\vec{a'})$ if and only if $a_i \overset{m}{\leadsto}_{\Gamma,s} a'_i$.
- $f(\vec{a}) \overset{i\cdot m}{\leadsto}_{\Gamma,s} f(\vec{a'})$ if and only if $a_i \overset{m}{\leadsto}_{\Gamma,s} a'_i$.
- $\lambda x{:}a.b \overset{2\cdot m}{\leadsto}_{\Gamma,s} \lambda x{:}a.b'$ if and only if $b \overset{m}{\leadsto}_{\Gamma',s} b'$ where $\Gamma' = \Gamma, x{:}a$.
- $a\,b \overset{1\cdot m}{\leadsto}_{\Gamma,s} a'\,b$ if and only if $a \overset{m}{\leadsto}_{\Gamma,s} a'$.
- $a\,b \overset{2\cdot m}{\leadsto}_{\Gamma,s} a\,b'$ if and only if $b \overset{m}{\leadsto}_{\Gamma,s} b'$.

A term a Γ,s-*rewrites* to a term a', $a \leadsto_{\Gamma,s} a'$, if there exists $m \in Pos(a)$ such that $a \overset{m}{\leadsto}_{\Gamma,s} a'$. The Γ,$s$-*reduction relation* is the reflexive and transitive closure of the Γ,s-rewrite relation. A term is in Γ,s-*normal form* if it cannot be Γ,s-reduced.

Given a strongly normalizable term a, we denote by $R^s_\Gamma(a)$ the least set containing a and stable by Γ,s-reduction.

We define the *anti*-Γ,s-*rewrite relation* by: $a \overset{m}{\leadsto}_{\Gamma\!\!/\,s} a'$ if and only if $a \overset{m}{\leadsto} a'$ but a is not Γ,s-reducible at position m.

Definition 5.45 (Erasing function) [JO97b] Given an algebraic type s, an environment Γ and a new constant $\bot_b$ for each Γ-type b, we define the following *erasing function* Ψ^s_Γ on every Γ-term a of type b,

- $\Psi^{s_i}_\Gamma(a) = \bot_b$ if a is not a Γ,s-term.

Otherwise, $\Psi^{s_i}_\Gamma(a)$ is defined by case on a:

- $\Psi^{s_i}_\Gamma(x) = x$,
- $\Psi^{s_i}_\Gamma(C(a_1, \ldots, a_n)) = C(\Psi^{s_i}_\Gamma(a_1), \ldots, \Psi^{s_i}_\Gamma(a_n))$,
- $\Psi^{s_i}_\Gamma(f(a_1, \ldots, a_n)) = f(\Psi^{s_i}_\Gamma(a_1), \ldots, \Psi^{s_i}_\Gamma(a_n))$,
- $\Psi^{s_i}_\Gamma(\lambda x{:}a.b) = \lambda x{:}a.\Psi^{s_i}_{\Gamma'}(b)$ where $\Gamma' = \Gamma, x{:}a$,
- $\Psi^{s_i}_\Gamma(a\,b) = \Psi^{s_i}_\Gamma(a)\,\Psi^{s_i}_\Gamma(b)$.

Definition 5.46 (Recursive call interpretation) [15] Given an environment Γ, and a function symbol f of arity $n \geq 0$ and type $s_1 \to \ldots \to s_n \to s$, we define its *recursive call interpretation function* $\Phi_{f,\Gamma}$ on strongly normalizable Γ-terms as follows:

- $\Phi_{f,\Gamma}(a_1, \ldots, a_n) = (\Phi^1_{f,\Gamma}(a_1), \ldots, \Phi^n_{f,\Gamma}(a_n))$

[15]inspired from the interpretation of [JO97b]

· $\Phi^i_{f,\Gamma}(a_i) = \{a_i\}$ if $i \notin \mathit{Ind}(f)$
· $\Phi^i_{f,\Gamma}(a_i) = \widehat{\Psi^{s_i}_\Gamma}(R^{s_i}_\Gamma(\chi^{s_i}_\Gamma(a_i)))$ if $i \in \mathit{Ind}(f)$

where $\widehat{\Psi^{s_i}_\Gamma}$ is the multi-set extension of $\Psi^{s_i}_\Gamma$.

Its associated *recursive call interpretation order* is $(\,(\rightsquigarrow^* \cup \trianglerighteq)_{mul}\,)_{stat_f}$.

Lemma 5.47 (R^s_Γ properties) Assume that s is an algebraic type, Γ is an environment and, a and b are two strongly normalizable terms.

1) If $a \rhd_{\Gamma,s} b$ then $R^s_\Gamma(a)\,(\rhd_{\Gamma,s})_{mul}\,R^s_\Gamma(b)$.
2) If $a \underset{\Gamma,s}{\rightsquigarrow} b$ then $R^s_\Gamma(a) \supseteq R^s_\Gamma(b)$.
3) If $a \underset{\Gamma\not,s}{\rightsquigarrow} b$ then $R^s_\Gamma(a)\,((\underset{\Gamma\not,s}{\rightsquigarrow})^*)_{mul}\,R^s_\Gamma(b)$.

Proof.

1) Suppose that $b = a_{|m}$. Let $b' \in R^s_\Gamma(b)$. Since $a \rhd_{\Gamma,s} b$, each superterm of b in a is a Γ,s-term. Hence, $a[b']_m \in R^s_\Gamma(a)$ and $a[b']_m \rhd_{\Gamma,s} b'$.
2) Immediate.
3) Let $b' \in R^s_\Gamma(b)$. Suppose that $a \overset{m}{\underset{\Gamma\not,s}{\rightsquigarrow}} b$ ($m \in \mathit{Pos}(a)$). If a is not a Γ,s-term then, by subject reduction, b is not a Γ,s-term either. Hence, $R^s_\Gamma(a) = \{a\}\,((\underset{\Gamma\not,s}{\rightsquigarrow})^*)_{mul}\,R^s_\Gamma(b) = \{b\}$. Otherwise, there exists a superterm $c = a_{|m_1}$ ($m = m_1 \cdot m_2$) of $a_{|m}$ which is not a Γ,s-term but whose superterms are Γ,s-terms. Then, $c \overset{m_2}{\underset{\Gamma\not,s}{\rightsquigarrow}} c' = b_{|m_1}$, $b = a[c']_{m_1}$, and a Γ,s-reduction path from b to b' is also a Γ,s-reduction path from a to a'. Hence, it suffices to replace in b' all the occurrences of c' by c to obtain a term a' such that $a'(\underset{\Gamma\not,s}{\rightsquigarrow})^* b'$.

$\diamond$

Lemma 5.48 (Ψ^s_Γ properties) Assume that s is an algebraic type, Γ is an environment and, a and b are two terms.

1) If $\Gamma \vdash a\!:\!b$ then $\Gamma \vdash \Psi^s_\Gamma(a)\!:\!b$.
2) If $a \rhd_{\Gamma,s} b$ then $\Psi^s_\Gamma(a) \rhd_{\Gamma,s} \Psi^s_\Gamma(b)$.
3) If $a \underset{\Gamma\not,s}{\rightsquigarrow} b$ then $\Psi^s_\Gamma(a) = \Psi^s_\Gamma(b)$.

Proof.

1) Trivial.
2) Suppose that $b = a_{|m}$. Since each superterm of b in a is typable by an algebraic type in which s occurs positively, $\Psi^s_\Gamma(a) = \Psi^s_\Gamma(a)[\Psi^s_\Gamma(b)]_m$. Since Ψ^s_Γ preserves typing, $\Psi^s_\Gamma(a) \rhd_{\Gamma,s} \Psi^s_\Gamma(b)$.
3) Suppose that $a \overset{m}{\underset{\Gamma\not,s}{\rightsquigarrow}} b$. If a is not a Γ,s-term then $\Psi^s_\Gamma(a) = \Psi^s_\Gamma(b) = \perp_d$ where d is a Γ-type of a. Otherwise, b has a superterm $c = a_{|m_1}$ ($m = m_1 \cdot m_2$) in a which is not a Γ,s-term but whose superterms are Γ,s-terms. Hence, $\Psi^s_\Gamma(a) = \Psi^s_\Gamma(a)[\perp_d]_{m_1}$ where d is a Γ-type of c. Since $c \overset{m_2}{\underset{\Gamma\not,s}{\rightsquigarrow}} c' = b_{|m_1}$, $b = a[c']_{m_1}$ and c' is not a Γ,s-term, $\Psi^s_\Gamma(b) = \Psi^s_\Gamma(a)$.

$\diamond$

Lemma 5.49 ($\widehat{\Psi^s_\Gamma}$ properties) Assume that s is an algebraic type, Γ is an environment and, a and b are two strongly normalizable terms.

1) If $a \rhd_{\Gamma,s} b$ then $\widehat{\Psi^s_\Gamma}(R^s_\Gamma(a))\,(\rhd_{\Gamma,s})_{mul}\,\widehat{\Psi^s_\Gamma}(R^s_\Gamma(b))$.
2) If $a \rightsquigarrow b$ then $\widehat{\Psi^s_\Gamma}(R^s_\Gamma(a))\,((\underset{\Gamma\not,s}{\rightsquigarrow})^*)_{mul}\,\widehat{\Psi^s_\Gamma}(R^s_\Gamma(b))$.

Proof.

1) By property of R^s_Γ and Ψ^s_Γ.
2) Let $b' \in \widehat{\Psi^s_\Gamma}(R^s_\Gamma(b))$. Then, $b' = \Psi^s_\Gamma(b'')$ where $b'' \in R^s_\Gamma(b)$. If $a \underset{\Gamma,s}{\rightsquigarrow} b$ then, by property of R^s_Γ, $R^s_\Gamma(a) \supseteq R^s_\Gamma(b)$, hence the result. Otherwise, we get the result by property of R^s_Γ and Ψ^s_Γ.

$\diamond$

Lemma 5.50 (Well-foundedness of the interpretation) Assume that $f(\vec{c}) \rightsquigarrow e$ is an admissible rewrite rule following the general schema, $f(\vec{c'})$ is a subterm of e such that $\vec{c} \rhd'_{stat_f} \phi_{f,\Gamma}(\vec{c'})$, and θ is a $\Gamma_{f(\vec{c})}$-substitution. Then, $\Phi_{f,\Gamma}(\vec{c}\theta)\,(\,(\rhd)_{mul}\,)_{stat_f}\,\Phi_{f,\Gamma}(\vec{c'}\theta)$.

Proof. Let us assume that $\vec{c} \rhd_{stat_f} \phi_{f,\Gamma}(\vec{c'})$ results from the comparison between c_i and $\phi^j_{f,\Gamma}(c'_j)$. There are two cases:

- If $i \in Ind(f)$, then $j = i$ since inductive positions are lexicographic. Hence, $\phi_f^i(c_i') = \chi_\Gamma^{s_i}(c_i)$, $\Phi_{f,\Gamma}^i(c_i\theta) = \widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(c_i\theta)))$ and $\Phi_{f,\Gamma}^i(c_i'\theta) = \widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(c_i'\theta)))$. By definition of the general schema, $\chi_\Gamma^{s_i}(c_i) = c_i$ and $c_i \rhd_{\Gamma,s_i} \chi_\Gamma^{s_i}(c_i')$. Hence, c_i cannot be a variable and $\chi_\Gamma^{s_i}(c_i\theta) = \chi_\Gamma^{s_i}(c_i)\theta = c_i\theta$. Besides, $c_i\theta \rhd_{\Gamma,s_i} \chi_\Gamma^{s_i}(c_i')\theta \rhd_{\Gamma,s_i} \chi_\Gamma^{s_i}(c_i'\theta)$. By property of $\widehat{\Psi_\Gamma^{s_i}}$, $\widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(c_i\theta)))\ (\rhd_{\Gamma,s_i})_{mul}\ \widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(c_i'\theta)))$, hence the result.
- If $i \notin Ind(f)$, then $j \notin Ind(f)$. Hence, $\phi_f^j(c_j') = c_j'$, $\Phi_{f,\Gamma}^i(c_i\theta) = \{c_i\theta\}$ and $\Phi_{f,\Gamma}^i(c_j\theta) = \{c_j\theta\}$. Since $c_i\theta \rhd c_j'\theta$, $\Phi_{f,\Gamma}^i(c_i\theta)\,(\rhd)_{mul}\,\Phi_{f,\Gamma}^j(c_j'\theta)$, hence the result.

◇

Lemma 5.51 (Compatibility of the interpretation) Assume that f is a function symbol of arity $n \geq 0$ and type $s_1 \to \ldots \to s_n \to s$, $\vec{a}$ and $\vec{a'}$ are two strongly normalizable sequences of terms such that $\Gamma \vdash f(\vec{a}) : s$ and $\vec{a} \rightsquigarrow^* \cup \trianglerighteq \vec{a'}$. Then, $\Phi_{f,\Gamma}(\vec{a})\,((\rightsquigarrow^* \cup \trianglerighteq)_{mul})_{stat_f}\,\Phi_{f,\Gamma}(\vec{a'})$.

Proof. The properties of $\widehat{\Psi_\Gamma^{s_i}}$ insures the compatibility of $\Phi_{f,\Gamma}$ with $\trianglerighteq$. For $\rightsquigarrow^*$, it suffices to prove it for one rewrite. Suppose that $a_i \rightsquigarrow a_i'$ $(1 \leq i \leq n)$. We distinguish two cases:

- If $i \notin Ind(f)$, then $\Phi_{f,\Gamma}^i(a_i) = \{a_i\}$ and $\Phi_{f,\Gamma}^i(a_i') = \{a_i'\}$. Since $a_i \rightsquigarrow a_i'$, $\Phi_{f,\Gamma}^i(a_i)\,(\rightsquigarrow^*)_{mul}\,\Phi_{f,\Gamma}^i(a_i')$, hence the result.
- If $i \in Ind(f)$, then $\Phi_{f,\Gamma}^i(a_i) = \Psi_\Gamma^{s_i}(R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(a_i)))$ and $\Phi_{f,\Gamma}^i(a_i') = \Psi_\Gamma^{s_i}(R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(a_i')))$. Let $b_1 \ldots b_{n'}$ $(n' \geq 1)$ be the application decomposition of a_i and suppose that $\chi_\Gamma^s(a) = a_1 \ldots a_k$ $(1 \leq k \leq n')$. Again, we distinguish four cases:
  - If a_i is reduced in b_j $(k+1 \leq j \leq n)$ then, by subject reduction, $\chi_\Gamma^{s_i}(a_i') = \chi_\Gamma^{s_i}(a_i)$, hence the result.
  - If a_i is reduced in b_j $(1 \leq j \leq k, k \geq 2)$ then $\chi_\Gamma^{s_i}(a_i) \rightsquigarrow \chi_\Gamma^{s_i}(a_i')$ and, by property of $\widehat{\Psi_\Gamma^{s_i}}$, $\widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(a_i)))$ $((\rightsquigarrow_{\not\Gamma,s})^*)_{mul}$ $\widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(a_i')))$, hence the result.
  - If a_i is reduced in b_1 and $k = 1$ then $\chi_\Gamma^{s_i}(a_i) \rightsquigarrow b_1'$ and $\chi_\Gamma^{s_i}(a_i') = \chi_\Gamma^{s_i}(b_1')$. By property of $\widehat{\Psi_\Gamma^{s_i}}$, $\widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(a_i)))$ $((\rightsquigarrow_{\not\Gamma,s})^*)_{mul}$ $\widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s_i}(b_1'))$ $(\trianglerighteq_{\Gamma,s_i})_{mul}$ $\widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(b_1')))$, hence the result.
  - Suppose now that $n' \geq 2$, $b_1 = \lambda x{:}b.c$ and $a_i' = c\{x \mapsto b_2\}\, b_3 \ldots b_n$. There are three cases:
    - If $k \geq 3$ then $\chi_\Gamma^{s_i}(a_i) \rightsquigarrow \chi_\Gamma^{s_i}(a_i')$. By property of $\widehat{\Psi_\Gamma^{s_i}}$, $\widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(a_i)))\,((\rightsquigarrow_{\not\Gamma,s})^*)_{mul}\,\widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s}(\chi_\Gamma^{s_i}(a_i')))$, hence the result.
    - If $k = 2$ then $\chi_\Gamma^{s_i}(a_i') = \chi_\Gamma^{s_i}(c\{x \mapsto b_2\})$. By property of $\widehat{\Psi_\Gamma^{s_i}}$, $\widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(a_i)))$ $((\rightsquigarrow_{\not\Gamma,s})^*)_{mul}$ $\widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s_i}(c\{x \mapsto b_2\}))$ $(\trianglerighteq_{\Gamma,s_i})_{mul}$ $R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(a_i'))$, hence the result.
    - If $k = 1$ then, by definition of a Γ,s-critical subterm, there exists some algebraic types t and t' such that $\Gamma \vdash \lambda x{:}b.c : t$, $\Gamma \vdash (\lambda x{:}b.c)\, b_2 : t'$ and s_i occurs positively in t and t'. Then, $t = t_2 \to t'$ where t_2 is a type for b_2 and s does not occur or occurs negatively in t_2. Hence, $c \neq x$, $\chi_\Gamma^{s_i}(a_i') = \chi_\Gamma^{s_i}(c)\{x \mapsto b_2\}$, $R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(a_i')) = \{d\{x \mapsto b_2\} \mid d \in R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(c))\}$ and $\widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(a_i'))) = \widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(c)))$. Besides, $\widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(a_i))) = \{\lambda x{:}b.c' \mid c' \in \widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s_i}(c))\}\,(\rhd)_{mul}\,\widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s_i}(c))\,(\trianglerighteq_{\Gamma,s_i})_{mul}\,\widehat{\Psi_\Gamma^{s_i}}(R_\Gamma^{s_i}(\chi_\Gamma^{s_i}(c)))$, hence the result.

◇

5.6 Strong normalization of CAIC

Now, all the conditions are met to set forth the strong normalization property of our calculus.

Theorem 5.52 (Strong normalization of CAIC) Assume that:

- the rewrite rules are admissible (Definition 2.17 on page 5),
- the rules of R_1 are conservative (if $R_\omega \neq \emptyset$),
- $\rightsquigarrow_{R_1}$ terminates on well-typed first-order algebraic terms,
- the function ordering $>_{\mathcal{F}}$ is well-founded on $\mathcal{F}_\omega$ (excluding mutually recursive definitions for higher-order function symbols),
- the rules of R_ω satisfy the general schema (Definition 5.38 on page 20).

Then, every well-typed term is strongly normalizable.

Proof. By Lemmas 5.26 on page 18, 5.43 on page 21, 5.50 on the preceding page, 5.51 and Theorem 5.21 on page 17. ◇

6 Confluence

We recall some definitions and results about confluence. The interested reader may find more details in [DJ90].

Definition 6.1 (Confluence) $\rightsquigarrow$ is *confluent* if, for any terms a, b, c such that $a \rightsquigarrow^* b$ and $a \rightsquigarrow^* c$, there exists a term d such that $b \rightsquigarrow^* d$ and $c \rightsquigarrow^* d$.

$\rightsquigarrow$ is *locally confluent* if, for any terms a, b, c such that $a \rightsquigarrow b$ and $a \rightsquigarrow c$, there exists a term d such that $b \rightsquigarrow^* d$ and $c \rightsquigarrow^* d$.

Lemma 6.2 (Newman) [New42] If a relation is strongly normalizable and locally confluent then it is confluent.

Definition 6.3 (Critical pair) Given two rewrite rules $l_1 \rightsquigarrow r_1$ and $l_2 \rightsquigarrow r_2$ with no variable in common, the *critical pair* of $l_2 \rightsquigarrow r_2$ on $l_1 \rightsquigarrow r_1$ at a non variable position $p \in Pos(l_1)$, if it exists, is the pair of terms $(r_1\theta, l_1\theta[r_2\theta]_m)$ where θ is the most general unifier of $l_{1|m}$ and l_2 (assuming that the abstractions are function symbols, and bound variables are constants).

It is said *trivial* if $l_2 \rightsquigarrow r_2$ is a renamed version version of $l_1 \rightsquigarrow r_1$ and $m = \epsilon$.

A critical pair is *confluent* if its elements have a common reduct.

Lemma 6.4 (Critical pairs) [Hue80] Given a set R_1 of first-order rewrite rules, if its (non trivial) critical pairs are confluent then $\rightsquigarrow_{R_1}$ is locally confluent.

Theorem 6.5 (Confluence) [BFG97] Assume that:
· the strong normalization conditions are satisfied (Theorem 5.52 on the previous page),
· R_1 is locally confluent on well-typed first-order algebraic terms,
· R_ω do not introduce critical pairs with R_1, R_ω and β,
then every well-typed term is confluent.

Proof. The β-reduction is locally confluent and the critical pairs between $\rightsquigarrow_{R_1}$ and $\rightsquigarrow_\beta$ are confluent. ◇

Theorem 6.6 (Full conversion admissibility) If the conditions for the strong normalization and for the confluence are satisfied (Theorem 6.5) then the following conversion rule is deducible:

$$\text{(conv")} \quad \frac{\Gamma \vdash a : b \quad \Gamma \vdash b' : p}{\Gamma \vdash a : b'} \quad (p \in \{\star, \Box\},\ b \overset{*}{\leftrightsquigarrow} b')$$

Theorem 6.7 (Logical soundness) If the conditions for the strong normalization and for the confluence are satisfied (Theorem 6.5) then there cannot be proofs of $false = \Pi x{:}\star.x$.

Proof. In [Bar93], Proposition 5.2.31 says that in any PTS extending the system F [16], if $\vdash a : false$ then a has no normal form. So, if every well-typed term is strongly normalizable, there cannot exist proofs of $false$. We reproduce that proof here.

Suppose a has a normal form a'. Then, by subject reduction, $\vdash a' : false$. As $\vdash false : \star$, a' is an object. So, by the typed structure lemma, it can only be a function symbol, a variable, an application or an abstraction. If a' is a function symbol then, by stripping, $false \overset{*}{\leftrightsquigarrow}_\Gamma s \in \mathcal{T}_S$ which is not possible. a' cannot be a variable since it is typed in an empty environment. If a' is an application then it can be decomposed as $b_1 \dots b_n$ $(n \geq 2)$ such that b_1 is not an application. But then, $\vdash b_1 : \Pi\vec{y}{:}\vec{c}.false$ for some terms $\vec{c}$. b_1 cannot be a function symbol nor a variable, so it must be an abstraction which is not possible since a' is in normal form. Therefore a' is an abstraction $\lambda x{:}\star.b$ and, by stripping, $x{:}\star \vdash b : x$. b cannot be a function symbol nor an abstraction. If it is an application then it can be decomposed as $c_1 \dots c_n$ $(n \geq 2)$ such c_1 is not an application. As c_1 cannot be a function symbol nor an abstraction, it must be a variable. So $c_1 = x$ and, by stripping, $x{:}\star \vdash x : \Pi\vec{y}{:}\vec{d}.x$ for some terms $\vec{d}$. Hence, $\star \overset{*}{\leftrightsquigarrow} \Pi\vec{y}{:}\vec{d}.x$ which is not possible. ◇

[16] simply typed λ-calculus extended with polymorphism

7 Type-checking decidability

In this section, we assume that the conditions for the strong normalization and for the confluence are satisfied (Theorem 6.5 on the preceding page). The equality used in the algorithms is the α-equivalence. In practice, this is avoided by using De Bruijn's terms [dB72].

Algorithms are written in a Caml-Light-like syntax [LM92].

`nf` denotes a function that takes a term as argument and returns its βR-normal form.

`type` denotes a function that takes a function symbol or a constructor as argument and returns its algebraic type.

Definition 7.1 (Type-deduction algorithm) We define hereafter a recursive function called `deduce` which takes an environment Γ and a term a of CAIC as arguments. Its goal is to compute a type of a in the environment Γ, if such a type exists, or to raise an exception called `error`.

```
letrec deduce $\Gamma$ $a$ =
   match a with
      $\Box$ -> raise error
      | $\star$ -> $\Box$
      | $s$ -> $\star$
      | $x$ -> if $x \in \Gamma$ then (nf $\Gamma(x)$) else raise error
      | $\Pi x{:}b.c$ -> if (deduce $\Gamma$ $b$)$\notin \{\star, \Box\}$ then raise error
         else let $q$=(deduce $(\Gamma, x{:}b)$ c) in
            if $q \in \{\star, \Box\}$ then $q$ else raise error
      | $\lambda x{:}b.c$ -> if (deduce $\Gamma$ $b$)$\notin \{\star, \Box\}$ then raise error
         else let $b'$=(nf $b$) in let $e$=(deduce $(\Gamma, x{:}b')$ $c$) in
            if $e = \Box$ then raise error else $\Pi x{:}d.e$
      | $b\,c$ -> let $d$=(deduce $\Gamma$ $c$) in match (deduce $\Gamma$ $b$) with
         $\Pi x{:}d.f$ -> (nf $f\{x \mapsto c\}$)
         | _ -> raise error
      | $C(a_1, \ldots, a_n)$ -> let $s_1$=(deduce $\Gamma$ $a_1$) in ... let $s_n$=(deduce $\Gamma$ $a_n$) in
         if (type $C$)=$s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s$ then $s$ else raise error
      | $f(a_1, \ldots, a_n)$ -> let $s_1$=(deduce $\Gamma$ $a_1$) in ... let $s_n$=(deduce $\Gamma$ $a_n$) in
         if (type $f$)=$s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s$ then $s$ else raise error
```

Lemma 7.2 (`deduce` properties)

(Termination) `deduce` terminates.

(Normality) If `deduce` does not raise an error then its result term is in normal form.

(Correction) Given a valid environment Γ, if (`deduce` Γ a) does not raise an error but returns a term b then $\Gamma \vdash a : b$.

(Completeness) If $\Gamma \vdash a : b$ then (`deduce` Γ a) does not raise an error but returns the normal form of b.

Proof.

· Termination: the recursive calls are done with a strict subterm of a as second argument.
· Normality: by induction on the structure of a.
· Correction: idem.
· Completeness: by induction on the structure of the derivation of $\Gamma \vdash a : b$.

◇

Definition 7.3 (Environment validity algorithm) We define hereafter a recursive function called `is_valid` which takes an environment as argument and returns a boolean value. Its goal is to check whether the environment is valid or not.

```
letrec is_valid $\Gamma$ =
   try match $\Gamma$ with
      $nil$ -> true
      | $\Gamma', x{:}a$ -> (is_valid $\Gamma'$) and (deduce $\Gamma'$ $a$) $\in \{\star, \Box\}$
   with error -> false
```

Lemma 7.4 (`is_valid` properties)
(Termination) `is_valid` terminates.
(Correctness) If (`is_valid` Γ) returns `true` then Γ is a valid environment, otherwise it is not.

Proof.
· Termination: the recursive call is done on a strictly smaller environment.
· Correctness: by induction on the environment length.
$\diamond$

Definition 7.5 (Type-checking algorithm) We define hereafter a function called `type_check` which takes an environment Γ and two terms a, b as arguments and returns a boolean value. Its goal is to check whether $\Gamma \vdash a\!:\!b$ is a valid judgement.

```
let type_check Γ a b =
   try (is_valid Γ) and (deduce Γ a) = (nf b)
   with error -> false
```

Lemma 7.6 (`type_check` correctness) Given an environment Γ and two terms a, b, if (`type_check` Γ a b) returns `true` then $\Gamma \vdash a\!:\!b$ is a valid judgement, otherwise it is not.

Proof. If (`is_valid` Γ) returns `false` then, by correctness, Γ is not a valid environment and $\Gamma \vdash a\!:\!b$ is not a valid judgement. If (`is_valid` Γ) returns `true` then, by correctness, Γ is a valid environment. If (`deduce` Γ a) raise an error then, by completeness, a is not typable in Γ and $\Gamma \vdash a:b$ is not a valid judgement. If (`deduce` Γ a) does not raise an error but returns a term b' in normal form then, by correctness, $\Gamma \vdash a:b'$. If $\Gamma \vdash a:b$ is a valid judgement then, by type uniqueness, $b \overset{*}{\leftrightsquigarrow}_{\Gamma} b'$. Then, by confluence, the normal form of b is b'. Hence, if the normal form of b is b' then, by conversion, $\Gamma \vdash a\!:\!b$ is a valid judgement, otherwise it is not. $\diamond$

Theorem 7.7 (Type-checking decidability) If the conditions for the strong normalization and for the confluence are satisfied (Theorem 6.5 on page 26) then type-checking is decidable.

Proof. Termination and correctness of `type_check`. $\diamond$

8 Rule admissibility

The aim of this section is to study the decidability of the admissibility conditions for rewrite rules (Definition 2.17 on page 5). The equality used in the algorithms is the α-equivalence. In practice, this is avoided by using De Bruijn's terms [dB72].

Algorithms are written in a Caml-Light-like syntax [LM92].

`nf` denotes a function that takes a term as argument and returns its βR-normal form.

`type` denotes a function that takes a function symbol or a constructor as argument and returns its algebraic type.

`result_type` denotes a function that takes a function symbol or a constructor as argument and returns its result type.

`mgu` denotes a function that takes a list of environments as argument and computes, if it exists, their most general unifier by computing that of the types of the variables shared by the environments, otherwise it raises an error.

`gen_var` denotes a function that returns a variable not already used.

8.1 Admissible rule terms

It is well known that terms of the simply typed λ-calculus have extended algebraic *principal types and environments* [Bar93] (see Definition 8.4 on the next page) that can be automatically infered. Since rule terms are simply-typed λ-terms, we will use this property to verify the admissibility conditions.

To prove the well-typedness condition, we need the following *algebraic stripping property* to be valid: given an algebraic type s, if $\Gamma \vdash a\,b : s$ then there exists an algebraic type t such that $\Gamma \vdash a : t \rightarrow s$ and $\Gamma \vdash b\!:\!t$. Hence, we define hereafter a class of rule terms that enjoys this property.

Definition 8.1 (Admissible rule terms) A rule term is *admissible* if it is in β-normal form and contains no subterm of the form $x\,\vec{d}$ where one of the terms $\vec{d}$ is of the form $\lambda\vec{z}{:}\vec{s}.y\,\vec{e}$ and x and y are free variables.

Note that any subterm of an admissible rule term is also admissible. Hence, we can reason by induction on its (β-normal form) structure.

Lemma 8.2 (Algebraic stripping) If $a\,b$ is an admissible rule term then there exists an algebraic type t such that, for any environment Γ and algebraic type s, $\Gamma \vdash a\,b{:}s$ implies $\Gamma \vdash b{:}t$ and $\Gamma \vdash a{:}t \rightarrow s$.

Proof. By stripping, $\Gamma \vdash a{:}\Pi x{:}d.e$, $\Gamma \vdash b{:}d$ and $s \overset{*}{\leftrightsquigarrow}_{\Gamma} e\{x \mapsto b\}$. Let us prove the property by induction on the rule term structure of a. The case $a = C(\vec{u})$ is not possible since constructors have sorts as result types. The case $a = \lambda x{:}t.u$ is not possible either since $a\,b$ is in β-normal form. If $a = f(\vec{u})$ then, by stripping, $\Pi x{:}d.e \overset{*}{\leftrightsquigarrow}_{\Gamma} s'$ where s' is the output type of f. Hence, $s' = t \rightarrow t'$ and, by product decomposition, $d \overset{*}{\leftrightsquigarrow}_{\Gamma} t$. If $a = x$ or $a = u\,v$ then $a\,b = w\,\vec{b}$ where w is not an application. If $w = x$ then none of the terms $\vec{b}$ is of the form $\lambda\vec{z}{:}\vec{s}.y\,\vec{e}$. Hence, they have a type Γ-equivalent to an algebraic type t_i independent of Γ. If $w = f(\vec{u})$ then b has an algebraic type independent of Γ. $\diamond$

Lemma 8.3 (Decidability of the admissibility of a rule term) It is decidable whether a rule term is admissible or not.

Proof. Trivial. $\diamond$

8.2 Decidability of well-typedness

Definition 8.4 (Principal type and environment) Given a rule term a, a pair (Γ, s) made of an (extended) algebraic environment and an (extended) algebraic type is a *principal (extended) algebraic type and environment* of a if and only if $\Gamma \vdash a{:}s$ and, for any other (extended) algebraic pair (Γ', s') such that $\Gamma' \vdash a{:}s'$, there exists a substitution θ such that $s' = s\theta$ and $\Gamma' = \Gamma\theta$.

Definition 8.5 (Type inference algorithm) We define hereafter a recursive function called `infer` which takes a rule term a as argument. Its goal is to compute an extended algebraic environment Γ and an extended algebraic term b, if they exist, such that $\Gamma^{\Box}, \Gamma \vdash a : b$ is a valid judgement, where $dom(\Gamma^{\Box}) = FV(\Gamma) \cap Var^{\Box}$ and $\Gamma^{\Box}(\alpha) = \star$ for any $\alpha \in dom(\Gamma^{\Box})$.

```
letrec infer $a$ =
   match $a$ with
      $x$ -> let $\alpha$=(gen_var) in ($x{:}\alpha$,$\alpha$)
      | $C(a_1,\dots,a_n)$ -> let ($\Gamma_1$,$t_1$)=(infer $a_1$) in ... let ($\Gamma_n$,$t_n$)=(infer $a_n$) in
         let $x$=(gen_var) in let $\theta$=(mgu $\Gamma_1$ ... $\Gamma_n$ $x$:(type $C$) $x{:}t_1 \rightarrow \dots \rightarrow t_n \rightarrow$(result_type $C$)) in
            ($\Gamma_1\theta \cup \dots \cup \Gamma_n\theta$, (result_type $C$))
      | $f(a_1,\dots,a_n)$ -> let ($\Gamma_1$,$t_1$)=(infer $a_1$) in ... let ($\Gamma_n$,$t_n$)=(infer $a_n$) in
         let $x$=(gen_var) in let $\theta$=(mgu $\Gamma_1$ ... $\Gamma_n$ $x$:(type $f$) $x{:}t_1 \rightarrow \dots \rightarrow t_n \rightarrow$(result_type $f$)) in
            ($\Gamma_1\theta \cup \dots \cup \Gamma_n\theta$, (result_type $f$))
      | $\lambda x{:}s.b$ -> let ($\Gamma$,$t$)=(infer $b$) in
         let $\theta$=(mgu $x{:}s$ $\Gamma$) in ($\Gamma\theta \setminus x{:}s$,$s \rightarrow t\theta$)
      | $b\,c$ -> let ($\Gamma_1$,$t_1$)=(infer $b$) in let ($\Gamma_2$,$t_2$)=(infer $c$) in
         let $\theta$=(mgu $\Gamma_1$ $\Gamma_2$) in let $\alpha$=(gen_var) in let $\theta'$=(mgu $x{:}t_1\theta$ $x{:}t_2\theta \rightarrow \alpha$) in
            ($\Gamma_1\theta\theta' \cup \Gamma_2\theta\theta'$, $\alpha\theta'$)
```

Lemma 8.6 (`infer` properties)
(Termination) `infer` terminates.
(Minimality) If (`infer` a) does not raise an error but returns a pair (Γ, s) then $dom(\Gamma) = FV(a)$.
(Algebraicity) If (`infer` a) does not raise an error but returns a pair (Γ, s) then Γ and s are extended algebraic.
(Correctness) If (`infer` a) does not raise an error but returns a pair (Γ, s) then $\Gamma^{\Box}, \Gamma \vdash a{:}s$.

Proof.

- Termination: The recursive calls are done on strict subterms of a.
- Minimality: By induction on the rule term structure of a.
- Algebraicity: By induction on the structure of a. If Γ_1 and Γ_2 are extended algebraic environments and $\theta =$`(mgu` Γ_1 Γ_2`)` then θ is also extended algebraic.
- Correctness: By induction on the structure of a. The cases x, $C(\vec{b})$ and $f(\vec{b})$ are trivial.
  - $a = \lambda x{:}s.b$: By induction hypothesis, $\Gamma^\square, \Gamma \vdash b{:}t$. Let θ be the most general unifier of Γ and $x{:}s$. Then $\Gamma\theta^\square, \Gamma\theta \vdash b{:}t\theta$. Let $\Gamma_2 = \Gamma\theta \setminus x{:}s$. Then $\Gamma_2^\square, \Gamma_2 \vdash a : s \rightarrow t\theta$.
  - $a = b\,c$: By induction hypothesis, $\Gamma_1^\square, \Gamma_1 \vdash b : t_1$ and $\Gamma_2^\square, \Gamma_2 \vdash c : t_2$. Let θ be the most general unifier of Γ_1 and Γ_2 and $\Gamma_3 = \Gamma_1\theta \cup \Gamma_2\theta$. Then, by weakening and substitution, $\Gamma_3^\square, \Gamma_3 \vdash b : t_1\theta$ and $\Gamma_3^\square, \Gamma_3 \vdash c{:}t_2\theta$. If $t_1\theta = t_2\theta \rightarrow s$ then $\Gamma_3^\square, \Gamma_3 \vdash a : s$. If $t_1\theta = \alpha$ then let $\Gamma_4 = \Gamma_3\{\alpha \mapsto (t_2\theta \rightarrow \beta)\}$. Then $\Gamma_4^\square, \Gamma_4 \vdash a : \beta$.

◇

Theorem 8.7 (`infer` Completeness) Given an admissible rule term a, an extended algebraic environment Γ and an extended algebraic type s, if $\Gamma \vdash a : s$ is a valid judgement then `(infer` a`)` does not raise an error but returns a pair (Γ', s') such that there exists a substitution σ verifying $s'\sigma = s$ and $\Gamma'\sigma = \Gamma|_{FV(a)}$.

Proof. By induction on the structure of a.

- $a = x$:
  By stripping, $s \overset{*}{\leftrightsquigarrow}_\Gamma t$ such that $x{:}t \in \Gamma$. By Γ-consistence of extended algebraic types, $s = t$. `(infer` a`)` returns $(x{:}\alpha, \alpha)$. Hence it suffices to take $\sigma = \{\alpha \mapsto s\}$: $\alpha\sigma = s$ and $(x{:}\alpha)\sigma = x{:}s = \Gamma|_{FV(a)}$.
- $a = C(a_1, \ldots, a_n)$, $\tau(C) = s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s'$:
  By stripping, $s \overset{*}{\leftrightsquigarrow}_\Gamma s'$ and $\Gamma \vdash a_i : s_i \; (1 \leq i \leq n)$. By Γ-consistence of extended algebraic types, $s = s'$. By induction hypothesis, `(infer` a_i`)` returns a pair (Γ_i, t_i) such that there exists a substitution σ_i verifying $t_i\sigma_i = s_i$ and $\Gamma_i\sigma_i = \Gamma|_{FV(a_i)}$. By definition of `gen_var`, the substitutions σ_i have disjoint domains. So let θ be the substitution such that $dom(\theta) = \biguplus_{i\in\{1\ldots n\}} dom(\sigma_i)$ and $\theta|_{dom(\sigma_i)} = \sigma_i$. If x is a variable shared by Γ_1 and Γ_2 then $\Gamma_1(x)\theta = \Gamma(x) = \Gamma_2(x)\theta$. Furthermore $t_i\theta = s_i$. Hence the environments $\Gamma_1, \ldots, \Gamma_n$, $x{:}$ $\tau(C)$, $x{:}t_1 \rightarrow \ldots \rightarrow t_n \rightarrow s$ are unifiable. Since $dom(\theta) = FV(t_1 \rightarrow \ldots \rightarrow t_n \rightarrow s)$, $(t_1 \rightarrow \ldots \rightarrow t_n \rightarrow s)\theta = \tau(C)$ and $FV(\tau(C)) = \emptyset$, their most general unifier is θ itself. Hence `(infer` a`)` returns $(\Gamma|_{FV(a)}, s)$ and it suffices to take the identity for σ.
- $a = f(a_1, \ldots, a_n)$, $f \in \mathcal{F}_t^n$, $t = s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s'$:
  Idem.
- $a = \lambda x{:}t.b$:
  By stripping, $s \overset{*}{\leftrightsquigarrow}_\Gamma \Pi x{:}t_1.t_2$ and $\Gamma, x{:}t_1 \vdash b : t_2$. Hence $s = s_1 \rightarrow s_2$ such that $s_1 \overset{*}{\leftrightsquigarrow}_\Gamma t_1$ and $s_2 \overset{*}{\leftrightsquigarrow}_\Gamma t_2$. By Γ-consistence of extended algebraic types, $s_1 = t_1$. By conversion, $\Gamma, x{:}t_1 \vdash b : s_2$ which is extended algebraic. By induction hypothesis, `(infer` b`)` returns a pair (Γ', s_2') such that there exists a substitution σ' verifying $s_2'\sigma' = s_2$ and $\Gamma'\sigma' = (\Gamma, x{:}s_1)|_{FV(b)}$. If $x \in FV(b)$ then $\Gamma'(x)\sigma' = s_1$. Hence Γ' and $x{:}s_1$ are unifiable. Let θ be their most general unifier. Then there exist a substitution σ such that $\sigma' = \theta\sigma$. `(infer` a`)` returns $(\Gamma'\theta \setminus x{:}s_1, s_1 \rightarrow s_2'\theta)$. We verify that σ satisfies the desired properties: $(s_1 \rightarrow s_2'\theta)\sigma = s_1 \rightarrow s_2 = s$ and $(\Gamma'\theta \setminus x{:}s_1)\sigma = (\Gamma, x{:}s_1)|_{FV(b)} \setminus x{:}s_1 = \Gamma|_{FV(a)}$.
- $a = b\,c$:
  By algebraic stripping, there exists an extended algebraic type t such that $\Gamma \vdash b : t \rightarrow s$ and $\Gamma \vdash c : t$. By induction hypothesis, `(infer` b`)` returns a pair (Γ_1, t_1) such that there exists a substitution σ_1 verifying $dom(\sigma_1) = FV(t_1)$, $t_1\sigma_1 = t \rightarrow s$ and $\Gamma_1\sigma_1 = \Gamma|_{FV(b)}$. `(infer` c`)` returns a pair (Γ_2, t_2) such that there exists a substitution σ_2 verifying $dom(\sigma_2) = FV(t_2)$, $t_2\sigma_2 = t$ and $\Gamma_2\sigma_2 = \Gamma|_{FV(c)}$. Let σ' be the substitution such that $dom(\sigma') = dom(\sigma_1) \uplus dom(\sigma_2)$ and $\sigma'|_{dom(\sigma_i)} = \sigma_i \; (i \in \{1, 2\})$. If $x \in dom(\Gamma_1) \cap dom(\Gamma_2)$ then $\Gamma_1(x)\sigma' = \Gamma(x) = \Gamma_2(x)\sigma'$. Hence Γ_1 and Γ_2 are unifiable. Let θ be their most general unifier. Then there exists a substitution σ'' such that $\sigma' = \theta\sigma''$. $t_1\theta\sigma'' = t \rightarrow s$ and $(t2\theta \rightarrow \alpha)\sigma'' = t \rightarrow \alpha$. Hence $t_1\theta$ and $t2\theta \rightarrow \alpha$ are unifiable. Let θ' be their most general unifier. Then there exists a substitution σ'' such that $\sigma''\{\alpha \mapsto s\} = \theta'\sigma$. We verify that σ satisfies the desired properties: $\alpha\theta'\sigma = s$ and $(\Gamma_1\theta\theta' \cup \Gamma_2\theta\theta')\sigma = \Gamma|_{FV(a)}$.

◇

Theorem 8.8 (Decidability of well-typedness) If a is a rule term then it is decidable whether it is typable or not.

Proof. Termination, correctness and completeness of `infer`. ◇

8.3 Decidability of algebraicity

Lemma 8.9 (Environment algebraicity) Given an admissible rule term a, an environment Γ and an (extended) algebraic type s, if $\Gamma \vdash a\!:\!s$ then $\beta hnf(\Gamma|_{FV(a)})$ is (extended) algebraic.

Proof. By induction on the rule term structure of a. If $a = x$ then $s \overset{*}{\underset{\Gamma}{\leftrightsquigarrow}} b$ such that $x{:}b \in \Gamma$. By property of (extended) algebraic types, $\beta hnf(b) = s$. If $a = \lambda x{:}t.b$ then $s = t \rightarrow t'$ and $\Gamma, x{:}t \vdash b\!:\!t'$. By induction hypothesis, $\beta whnf((\Gamma, x{:}t)|_{FV(a)\cup\{x\}})$ is (extended) algebraic. Hence $\beta hnf(\Gamma|_{FV(a)})$ is (extended) algebraic. If $a = C(\vec{b})$ or $a = f(\vec{b})$ then, by induction hypothesis, $\beta hnf(\Gamma|_{FV(b_i)})$ $(1 \le i \le |\vec{b}|)$ is (extended) algebraic. Hence $\beta hnf(\Gamma|_{FV(a)})$ is (extended) algebraic. If $a = b\,c$ then, since a is admissible, by algebraic stripping, $\Gamma \vdash b : t \rightarrow s$ and $\Gamma \vdash c : t$ for some algebraic type t. By induction hypothesis, $\beta hnf(\Gamma|_{FV(b)})$ and $\beta whnf(\Gamma|_{FV(c)})$ are (extended) algebraic. Hence $\beta whnf(\Gamma|_{FV(a)})$ is (extended) algebraic. $\diamond$

In other words, the free variables of an admissible rule term whose type is (extended) algebraic are constrained to have an (extended) algebraic type.

Theorem 8.10 (Decidability of algebraicity) Any admissible rule term headed by a function symbol satisfies the algebraicity condition.

Proof. As it is headed by a function symbol, it has an algebraic type. By the algebraicity lemma, any environment in which it has that type is equivalent to an algebraic environment. By completeness of the inference, this can only be its infered environment. $\diamond$

8.4 Decidability of type-preservation

Definition 8.11 (Type-deduction algorithm) We define hereafter a recursive function called `deduce_rt` which takes an extended algebraic environment Γ and a rule term a as arguments. Its goal is to compute the extended algebraic type of a in the environment Γ, if such a type exists, or to raise an exception called `error`.

```
letrec deduce_rt $\Gamma$ $a$ =
   match a with
      | $x$ -> if $x \in \Gamma$ then $\Gamma(x)$ else raise error
      | $\lambda x{:}t.b$ -> let $t'$=(deduce_rt ($\Gamma, x{:}t$) $b$) in $t \rightarrow t'$
      | $b\,c$ -> let $t$=(deduce_rt $\Gamma$ $c$) in match (deduce_rt $\Gamma$ $b$) with
         $t \rightarrow t'$ -> $t'$
         | _ -> raise error
      | $C(a_1, \ldots, a_n)$ -> let $s_1$=(deduce_rt $\Gamma$ $a_1$) in ... let $s_n$=(deduce_rt $\Gamma$ $a_n$) in
         if (type $C$)=$s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s$ then $s$ else raise error
      | $f(a_1, \ldots, a_n)$ -> let $s_1$=(deduce_rt $\Gamma$ $a_1$) in ... let $s_n$=(deduce_rt $\Gamma$ $a_n$) in
         if (type $f$)=$s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s$ then $s$ else raise error
```

Lemma 8.12 (`deduce_rt` properties)
(Termination) `deduce_rt` terminates.
(Algebraicity) If (`deduce_rt` Γ) does not raise an error but returns a term s then s is an extended algebraic type.
(Correctness) If (`deduce_rt` Γ a) does not raise an error but returns a term s then $\Gamma \vdash a\!:\!s$.
(Completeness) Given an extended algebraic type s, if $\Gamma \vdash a : s$ then (`deduce_rt` Γ a) does not raise an error but returns s.

Proof.
· Termination: the recursive calls are done with a strict subterm of a as second argument.
· Algebraicity: by induction on the structure of a.
· Correctness: idem.
· Completeness: idem.
$\diamond$

Definition 8.13 (Type-preservation algorithm) We define hereafter a function called `preserv` which takes two rule terms l and r as arguments and returns a boolean value. Its goal is to check whether the rule $l \rightsquigarrow r$ satisfies the type-preservation condition.

```
let preserv $l$ $r$ =
   try let ($\Gamma$,$s$)=(infer $l$) in
      try (deduce_rt $\Gamma$ $r$)=$s$
      with error -> false
   with error -> true
```

Lemma 8.14 (`preserv` correctness) Given an admissible rule term l headed by a function symbol and a rule term r, if (`preserv` l r) returns `true` then $l \rightsquigarrow r$ satisfies the type-preservation condition, otherwise it does not.

Proof. Let s be the result type of the function symbol by which l is headed. If (`infer` l) raises an error then, by completeness, l is not typable in any extended algebraic environment. By the environment algebraicity lemma, l is not typable in any environment. Hence, $l \rightsquigarrow r$ trivially satisfies the type-preservation condition since there is no environment Γ such that $\Gamma \vdash l:s$.

Suppose that (`infer` l) does not raise an error but returns a pair (Γ, s). Let Γ' be an environment such that $\Gamma' \vdash l:s$. Since l is admissible, by the environment algebraicity lemma, $\Gamma'|_{FV(l)}$ is reducible to an algebraic environment Γ''. Hence, by completeness of `infer`, Γ is algebraic and $\Gamma'' = \Gamma$.

If (`deduce_rt` Γ r) does not raise an error but returns an algebraic type t then, by correctness, $\Gamma \vdash r:t$.

By weakening and subject reduction, $\Gamma' \vdash r:t$. If $\Gamma' \vdash r:s$ then, by type uniqueness and Γ-consistence of algebraic types, $t = s$. If $t = s$ then $\Gamma' \vdash r:s$ is a valid judgement and $l \rightsquigarrow r$ satisfies the type-preservation condition. If $t \neq s$ then $\Gamma' \not\vdash r:s$ and $l \rightsquigarrow r$ does not satisfy the type-preservation condition.

If (`deduce_rt` Γ r) raises an error then r is not typable in Γ. Hence, r is not typable in Γ' and $l \rightsquigarrow r$ does not satisfy the type-preservation condition. $\diamond$

Theorem 8.15 (Decidability of type-preservation) If $l \rightsquigarrow r$ is a rule made of an admissible rule term l then it is decidable whether it satisfies the type-preservation condition.

Proof. Termination and correctness of `preserv`. $\diamond$

9 Conclusion

We have extended the calculus of constructions with inductive sorts, first-order and higher-order rewrite rules. We have proved that the combined calculus enjoys the subject reduction property, is strongly normalizable, confluent, logically sound and that type-checking is decidable.

This work has been based on the work of Barbanera, Fernández and Geuvers [BFG94], and on the recent work of Jouannaud and Okada on positive inductive sorts [JO97b].

Compared to [BFG94], we provide inductive types, a more general schema for higher-order definitions, abstractions and applied variables in both sides of the rewrite rules. We have clarified several notions introduced in this paper, namely: some aspects of the subject reduction proof; the proof of unicity for algebraic types; the notion of "cube-embeddability", called rule admissibility in our paper; its decidability proof which applies to a much larger class of definitions; a much simpler proof of strong normalization inspired from [Geu95].

Compared to [JO97b], we have again clarified several key-notions: the notion of critical subterm and of an admissible recursive call interpretation; the expression of the general schema; and the reducibility proof of the higher-order function symbols. Besides, all these notions have been formulated in the context of a much richer type theory, the calculus of constructions, instead of a simply-typed λ-calculus with inductive types, and our strong normalization proof is highly modular.

We believe that it should not be too hard to add polymorphic rewriting, to extend the general schema to catch recursors for positive inductive types, although there are few practical examples where this is really needed. This would provide with more inductive types than in the Calculus of Inductive Constructions [Wer94].

On the other hand, we have not yet considered the case of dependent inductive sorts, strong elimination, subsorts and quotient types, conditional rewriting, universes and η-reductions. This is left for future work.

References

[Bar84] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics (2nd ed.)*. North-Holland, 1984.

[Bar93] H. Barendregt. *Handbook of Logic in Computer Science*, volume 2, chapter 2. Oxford University Press, 1993.

[BBC$^+$98] B. Barras, S. Boutin, C. Cornes, J. Courant, D. Delahaye, D. de Rauglaudre, J.-C. Filliâtre, E. Gimnez, H. Herbelin, G. Huet, P. Loiseleur, C. Muñoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saïbi, and B. Werner. *The Coq Proof Assistant Reference Manual Version 6.2.* INRIA-Rocquencourt-CNRS-Universit Paris Sud- ENS Lyon, May 1998.

[BFG94] F. Barbanera, M. Fernández, and H. Geuvers. Modularity of strong normalization and confluence in the algebraic-λ-cube. In *Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 406–415, Paris, France, 4–7 July 1994. IEEE Computer Society Press.

[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, November 1997.

[CH88] T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:96–120, 1988.

[CPM90] Th. Coquand and C. Paulin-Mohring. Inductively defined types. In P. Martin-Löf and G. Mints, editors, *Proceedings of Colog'88*, LNCS 417. Springer-Verlag, 1990.

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

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–309. North-Holland, 1990.

[Gal90] J. Gallier. On Girard's "Candidats de Reductibilité". In P.-G. Odifreddi, editor, *Logic and Computer Science*. North Holland, 1990.

[Geu95] H. Geuvers. A short and flexible proof of strong normalization for the calculus of constructions. In P. Dybjer, B. Nordström, and J. Smith, editors, *Selected Papers 2nd Intl. Workshop on Types for Proofs and Programs, TYPES'94, Båstad, Sweden, 6–10 June 1994*, volume 996 of *Lecture Notes in Computer Science*, pages 14–38. Springer-Verlag, Berlin, 1995.

[GLT88] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types.* Cambridge Tracts in Theoretical Computer Science 7. Cambridge University Press, 1988.

[GN91] J. H. Geuvers and M.-J. Nederhof. A modular proof of strong normalization for the calculus of constructions. *Journal of Functional Programming*, 1(2):155–189, 1991.

[How80] W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, Inc., New York, N.Y., 1980.

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

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

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

[LM92] X. Leroy and M. Mauny. *The Caml-Light Reference Manual.* INRIA, 1992.

[New42] M. H. A. Newman. On theories with A combinatorial definition of equivalence. In *Annals of Math*, volume 43, pages 223–243, 1942.

[Par93] M. Parigot. Strong normalization for second order classical natural deduction. In *Proceedings 8th Annual IEEE Symp. on Logic in Computer Science, LICS'93, Montreal, Canada, 19–23 June 1993*, pages 39–46. IEEE Computer Society Press, 1993.

[PM93] C. Paulin-Mohring. Inductive definitions in the system coq - rules and properties. In M. Bezem and J. F. Groote, editors, *LNCS 664*. Springer-Verlag, 1993.

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

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its application. *Pacific J.Math.*, 5:285–309, 1955.

[Wer94] B. Werner. *Une Théorie des Constructions Inductives*. PhD thesis, Université Paris VII, 1994.