Modular properties of algebraic pure type systems

Gilles Barthe [1,2*] gillesb@cs.kun.nl
Herman Geuvers [1,3] herman@win.tue.nl

[1] Faculty of Mathematics and Informatics, University of Nijmegen, The Netherlands
[2] Department of Computer Science, University of Manchester, United Kingdom
[3] Faculty of Math. and Informatics, Technological Univ. of Eindhoven, The Netherlands

Abstract. *We introduce the framework of algebraic pure type systems, a generalisation of pure type systems with higher order rewriting* à la *Jouannaud-Okada, and initiate a generic study of the modular properties of these systems. We give a general criterion for a system of this framework to be strongly normalising. As an application of our criterion, we recover all previous strong normalisation results for algebraic pure type systems.*

1 Introduction

Algebraico-functional languages, introduced by Jouannaud and Okada in [18], are based on a very powerful paradigm combining type theory and higher-order rewriting systems. These languages embed in typed λ-calculi higher-order rewriting and hence allow the definition of abstract data types as it is done in equational languages such as OBJ. Examples of such languages which have been studied in the literature include the algebraic simply typed λ-calculus ([18]), algebraic type assignments systems ([1]) and the algebraic calculus of constructions ([2]). In this paper, we introduce a very general framework to study the combination of type theories with higher-order rewriting systems. The combination is based on pure type systems ([3]); the result is a very general framework of *algebraic pure type systems* which covers in particular the systems of the algebraic λ-cube, a generalisation of Barendregt's cube studied in [2, 18]. A particular interest of the framework is that it offers the possibility to initiate a generic study of the meta-theory of these systems. First, basic meta-theoretic results, such as the substitution lemma or the generation lemma ([3, 13]) can be proved for arbitrary algebraic pure type systems. Second, one can address modularity results in a very abstract way, as it has been successfully done in term-rewriting (some striking examples can be found in [20, 25]). The main contribution of this paper is to give a general criterion for an algebraic pure type system to be strongly normalising. We show that if a pure type system satisifies a certain abstract condition (slightly stronger than being strongly normalising) and a finite

* Address from October 1995: Department of Software Technology, CWI, The Netherlands

list of higher-order algebraic rewriting systems satisfy Jouannaud and Okada's scheme, then the combined system is strongly normalising for the combined reduction if it satisfies the subject reduction property. As a corollary, we obtain a new proof of strong normalisation for the algebraic calculus of constructions ([2] and [1, 7, 8, 18] for subsystems) and to our knowledge the first proof of strong normalisation for algebraic higher-order logic (the algebraic extension of λHOL [13]) and the algebraic calculus of constructions with universes (with left-linear rewriting systems). In our view, the distinctive features of our approach are its generality (all the known results on modularity of termination for algebraic pure type systems can be obtained as a corollary of our result), its simplicity (the complexity of the proof is similar to the corresponding strong normalisation argument for pure type systems) and its flexibility (it is easy to adapt the proof to variants of pure type systems).

The paper is organised as follows: in the next section, we introduce algebraic pure type systems. In section 3, we give an alternative syntax in which variables come labelled with a potential type and show the 'equivalence' between the two formulations. Besides we formulate a general criterion for an algebraic pure type system to be strongly normalising. In section 4, we prove strong normalisation for those systems satisfying the virterion by a general model construction. Section 5 focuses on the applications of the result to existing systems. The last section contains some final remarks about the work as well as directions for future research.

We assume the reader to be reasonably familiar with pure type systems and their basic meta-theory, as presented for example in [3] or [13].

2 Combining higher-order rewriting systems and pure type systems

2.1 Higher-order rewriting systems

In this section, we introduce higher-order rewriting systems. The presentation is deliberately non-conventional in some respects but has been chosen to give a clear presentation of the general schema of [18]. For examples and applications of the general schema, the reader is refered to [10, 18].

Let Λ be a set. Elements of Λ are called base data[2]. The set of data is defined inductively as follows:

- every base datum is a datum;
- if $\sigma_1, \ldots, \sigma_n$ are data and τ is a base datum, then $(\sigma_1, \ldots, \sigma_n \rightarrow \tau)$ is a datum.

By convention, brackets associate to the right and will be omitted when the convention applies. A datum of the form $(\tau_1, \ldots, \tau_m, \sigma_1, \ldots, \sigma_n \rightarrow \sigma)$ where the τ_i's are higher-order data (i.e of arrow type) and the σ_i's are base data is called

[2] Usually elements of Λ are called sorts. We prefer to keep this name for the sorts of the pure type system.

a *saturated datum*. The set of *first-order data* is the subset of saturated data for which $m = 0$, i.e. a first-order datum is one of the form $(\sigma_1, \dots, \sigma_n \rightarrow \sigma)$ where the σ_i's are base data. (Note that σ is a base datum by the definition of data.) The set of saturated data and first-order data are respectively denoted by Λ^* and Λ^1.

Definition 1 *A* higher-order signature Σ *over* Λ *consists of an indexed family of (pairwise disjoint) sets* $(\mathcal{F}_w)_{w \in \Lambda^*}$*.*

Elements of the $\mathcal{F}_w$'s are called function symbols. A function symbol is first-order if it belongs to $\mathcal{F}_w$ for some first-order datum w and higher-order otherwise. For every datum τ, the set $T_{(\Sigma,\tau)}$ of terms of datum τ is defined inductively. As usual, we start from a countably infinite set of variables V_τ for each datum τ. The rules are:

- elements of V_τ are terms of datum τ;
- if $x \in V_{(\sigma_1,\dots,\sigma_n \rightarrow \tau)}$ and t_i has datum σ_i for $i = 1, \dots, n$, then $x(t_1, \dots, t_n)$ has datum τ
- if $f \in \mathcal{F}_{(\sigma_1,\dots,\sigma_n \rightarrow \tau)}$ and t_i has datum σ_i for $i = 1, \dots, n$, then $f(t_1, \dots, t_n)$ has datum τ.

A term is *first-order* if all variables occurring in it are of base datum and all function symbols occurring in it are of first-order datum. A term is *higher-order* otherwise. Note that all terms are fully applied in the sense that only variables can be of higher-order datum. First-order terms are of the form $f(t_1, \dots, t_n)$ where f is a first-order function symbol and the t_i's are first-order terms. Higher-order terms are of the form $F(X_1, \dots, X_m, t_1, \dots, t_n)$ where the X_i's are higher-order variables and the t_i's are terms of base datum. The set var of variables of a term, occurences and substitution are defined as usual.

Definition 2 *A* rewrite rule *is a pair (s, t) (written $s \rightarrow t$) of terms of the same datum such that* $\mathsf{var}(t) \subseteq \mathsf{var}(s)$ *and s is not a variable.*

A rewrite rule is *first-order* if the terms are and *higher-order* otherwise. Recall that a rewrite rule $s \rightarrow t$ is *non-duplicating* if the number of occurences of each variable x in t is lesser or equal to the number of occurences of x in s.

Definition 3 ([2, 18]) *A higher-order rewrite rule $F(X_1, \dots, X_m, t_1, \dots, t_n) \rightarrow v$ satisfies* the general schema *if*

1. *F is a higher-order function symbol;*
2. *F does not occur in any of the t_i's;*
3. *the higher-order variables occuring in the t_i's belong to $(X_1, \dots, X_m)$;*
4. *for every subterm of v of the form $F(X'_1, \dots, X'_m, r_1, \dots, r_n)$, one has $\mathbf{t} \rhd_{mul} \mathbf{r}$ where $\rhd_{mul}$ is the multiset extension of the strict subterm ordering.*

Condition 2 is not essential but ensures that $F(X_1, \dots, X_m, t_1, \dots, t_n)$ is rewritable in the sense of [10]. Note that as a consequence of the definition, F does not occur in any subterm of v of the form $F(X'_1, \dots, X'_m, r_1, \dots, r_n)$ except in head position. Higher-order rewrite rules are a mild generalisation of the rules of primitive recursion.

Definition 4 *A* higher-order rewriting system *is a set of rewrite rules such that:*

- *first-order rules are non-duplicating;*
- *higher-order rules satisfy the general schema;*
- *there are no mutually recursive definitions of higher-order function symbols.*

The last requirement is not essential but has been added to simplify proofs. In the sequel, we let $\rightarrow_R$ denote the algebraic reduction relation. As usual, we distinguish between first-order reduction $\rightarrow_{for}$ and higher-order reduction $\rightarrow_{hor}$.

2.2 Algebraic pure type systems

In this paragraph, we extend the framework of pure type systems with higher-order rewriting *à la* Jouannaud-Okada. The resulting framework of algebraic pure type systems covers a large class of algebraico-functional languages and provides a suitable basis to study modular properties of these languages.

Definition 5 *An* algebraic pure type system *(or apts for short) is specified by a quintuple* $\lambda\mathcal{S} = (\mathcal{R}, S, \mathsf{sortax}, \mathsf{rules}, \mathsf{datax})$ *where*

- $\mathcal{R}$ *is a finite list of higher-order rewriting systems* $\mathcal{R}_i = (\Lambda_i, \Sigma_i, R_i)$ *(i.e.* Λ_i *is a set of (base) data,* Σ_i *is a higher-order signature over* Λ_i *and* $\mathcal{R}_i$ *is a higher-order rewriting system over* Σ_i*) for* $i = 1, \ldots, n$*;*
- S *is a set of sorts;*
- $\mathsf{sortax} : S \rightharpoonup S$*,* $\mathsf{rules} : S \times S \rightharpoonup S$ *and* $\mathsf{datax} : \{\Lambda_1, \ldots, \Lambda_n\} \rightharpoonup S$ *are partial functions.*

Note that the definition implicitely requires the algebraic pure type system to be functional in the sense of [13] (such systems are called singly-sorted in [3]). This is not a real restriction as one can hardly imagine a non-functional pure type system of interest.

Definition 6 *Let* V *be an arbitrary infinite set. The set of pseudo-terms* Pseudo *of an algebraic pure type system* $\lambda\mathcal{S} = (\mathcal{R}, S, \mathsf{sortax}, \mathsf{rules}, \mathsf{datax})$ *is defined as follows:*

- *variables, sorts and data are pseudo-terms;*
- *if* A, B *are pseudo-terms and* $x \in V$*, then* $A\ B$*,* $\lambda x : A.B$ *and* $\Pi x : A.B$ *are pseudo-terms;*
- *if* f *is a function symbol of some signature* Σ_i *of datum* $(\tau_1, \ldots, \tau_n \rightarrow \tau)$ *and* $t_1, \ldots, t_n$ *are pseudo-terms, then* $f(t_1, \ldots, t_n)$ *is a pseudo-term.*

In [2], function symbols are treated as constants whereas we chose to treat them as constructors. Our choice was dicted by matters of convenience but there is no real difference between the two systems. In particular, our result applies to algebraic pure type systems with either definition of pseudo-terms.

There are two notions of reduction on pseudo-terms: algebraic reduction $\rightarrow_R$ inherited from the term-rewriting systems and β-reduction. The combined reduction is denoted by $\rightarrow_{mix}$. The rules for derivation for $\lambda\mathcal{S}$ are:

Axiom	$\dfrac{}{\vdash c : s}$	if $\mathsf{datax}\ A = s$ and $c \in A$ or $\mathsf{sortax}\ c = s$
Function	$\dfrac{\Gamma \vdash t_i : \sigma_i \quad \text{for } i = 1, \ldots, n}{\Gamma \vdash f(t_1, \ldots, t_n) : \tau}$	if f is a function symbol of datum $\sigma_1, \ldots, \sigma_n \rightarrow \tau$
Start	$\dfrac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}$	if $x \notin \Gamma$
Weakening	$\dfrac{\Gamma \vdash t : B \quad \Gamma \vdash B : s}{\Gamma, x : B \vdash t : B}$	if $x \notin \Gamma$
Product	$\dfrac{\Gamma, \Delta \vdash A : s_1 \quad \Gamma, x : A, \Delta \vdash B : s_2}{\Gamma, \Delta \vdash \Pi x : A.B : s_3}$	if $\mathsf{rules}(s_1, s_2) = s_3$ and $x \notin \mathsf{FV}(\Delta)$
Application	$\dfrac{\Gamma \vdash t : \Pi x : A.B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B[u/x]}$	
Abstraction	$\dfrac{\Gamma, x : A, \Delta \vdash t : B \quad \Gamma \vdash \Pi x : A.B : s}{\Gamma, \Delta \vdash \lambda x : A.t : \Pi x : A.B}$	if $x \notin \mathsf{FV}(\Delta)$
Exp/Red	$\dfrac{\Gamma \vdash u : A \quad \Gamma \vdash B : s}{\Gamma \vdash u : B}$	if $A \rightarrow_{\beta R} B$ or $B \rightarrow_{\beta R} A$

Note that the abstraction and product rules have a slightly more general presentation than usual (see [3] for example). For pure type systems, the two presentations can be shown to be equivalent; in fact, this is a simple consequence of the permutation lemma and strengthening ([17]). In an algebraic pure type system, the reduction relation is not confluent on the set of pseudo-terms; as a result, the usual proofs of subject reduction and of other results relying on subject reduction, such as strenghtening cannot be extended. This motivates the following definition.

Definition 7 *An (algebraic) pure type system* $\lambda\mathcal{S} = (\mathcal{R}, S, \mathsf{sortax}, \mathsf{rules}, \mathsf{datax})$ *has the* subject reduction property *if for all pseudo-terms* M, N, A *with* $M \rightarrow_\beta N$ *and pseudo-context* Γ*,*

$$\Gamma \vdash M : A \quad \Rightarrow \quad \Gamma \vdash N : A$$

As subject reduction for R-reduction holds in an arbitrary algebraic pure type system, it is easy to conclude that in an algebraic pure type system with the subject reduction property,

$$\Gamma \vdash M : A \quad \Rightarrow \quad \Gamma \vdash N : A$$

for every pseudo-context Γ and all pseudo-terms M, N, A with $M \rightarrow_{mix} N$.

The fact that one cannot prove subject reduction for algebraic pure type systems might appear as a serious drawback of the system. Fortunately, for most systems of interest, including the systems of the algebraic λ-cube ([2, 10]) or algebraic higher-order logic, subject reduction holds ([2, 10]). Subject reduction can also be ensured by imposing some conditions on the rewriting systems: if the rewriting systems are left-linear, then the reduction relation is confluent on the set of pseudo-terms and subject reduction can be proved as usual. Finally, note that we know that conversion paths in derivations go through legal terms even if we do not know subject reduction: this is enforced by the expansion/reduction rule. This restrictive rule ensures that the very basic property of soundness, as defined in [15], holds.

In order to have a standard presentation of the results in this paper, we introduce the following terminology.

Definition 8 *An algebraic pure type system* $\lambda\mathcal{S} = (\boldsymbol{\mathcal{R}}, S, \mathsf{sortax}, \mathsf{rules}, \mathsf{datax})$ *is* $\mathcal{R}$-confluent (resp. $\mathcal{R}$-terminating, resp. $\mathcal{R}$-canonical) *if all its rewriting systems are confluent (resp. terminating, resp. canonical).*

3 A criterion for strong normalisation

In [24], Terlouw gives a general criterion for a type system to be strongly normalising. We adapt his criterion to pure type systems and give an equivalent criterion in terms of algebraic pure type systems with labelled variables. The advantage of the second characterisation is that it eliminates the need to reason on contexts.

3.1 Stratified algebraic pure type systems

Definition 9 *A term M is a* prototype *in context Γ if there exist a sort s and pseudo-terms $P_1, \ldots, P_n$ such that $\Gamma \vdash M\ P_1\ \ldots\ P_n : s$.*

For every family of contexts $\boldsymbol{\Gamma} = (\Gamma_i)_{i\in\mathbb{N}}$, we can define a relation $\prec_{\Gamma}$ on pseudo-terms as the smallest relation such that for every pseudo-terms M, N, if $M\ N$ is a prototype in context Γ_i for some $i \in \mathbb{N}$, then $M\ N \prec_{\Gamma} M$ and $N \prec_{\Gamma} M$. Furthermore, we say that a family of contexts $(\Gamma_i)_{i\in\mathbb{N}}$ is *compatible* if for every $i \in \mathbb{N}$, the context Γ_i is an initial part of the context Γ_{i+1}.

Definition 10 *An (algebraic) pure type system is* stratified *if for every compatible family of contexts $\boldsymbol{\Gamma} = (\Gamma_i)_{i\in\mathbb{N}}$, the relation $\prec_{\Gamma}$ is well-founded.*

The main result of the paper is the following general strong normalisation criterion.

Theorem 11 *Every stratified $\mathcal{R}$-terminating (algebraic) pure type system with the subject reduction property is strongly normalising.*

The combined reduction is weakly Church-Rosser on legal terms, so we can advocate Newton's Lemma to lift Theorem 11 to $\mathcal{R}$-canonical algebraic pure type system.

Proposition 12 *Every stratified $\mathcal{R}$-canonical (algebraic) pure type system with the subject reduction property is strongly normalising and confluent.*

As a corollary, we recover the standard results on strong normalisation of algebraic pure type systems as well as some new results. As for the known results, we feel our proof improves on previous work by being direct and of the same complexity as the strong normalisation proof for the (pure) λ-cube. In contrast, the authors of [2] have to consider a reduction-preserving mapping of the algebraic calculus of constructions into an algebraic type assignment system and to show that the target system is strongly normalising.

Corollary 13 - *Systems of the algebraic λ-cube are strongly normalising provided R-reduction is strongly normalising on algebraic terms ([2, 18]).*

- *Algebraic higher-order logic is strongly normalising provided R-reduction is strongly normalising on algebraic terms.*
- *The algebraic calculus of constructions with universes is strongly normalising provided all rewrite systems are left-linear and R-reduction is strongly normalising on algebraic terms.*

Similar results exist for R-canonical algebraic pure type systems.

3.2 Labelled variables

In this section, we introduce a technical variant of (algebraic) pure type systems in which variables are "typed". This is reminiscent of some presentations of simply typed λ-calculus in which each type τ comes equipped with a set of variables of type τ. In (algebraic) pure type systems, terms and types are defined simultaneously so the naive approach taken for simply typed λ-calculus cannot be used any longer. Our solution is to assign to every variable a pseudo-term, which will be its unique type if the variable is well-typed. In the sequel, we consider a fixed pure type system $\lambda\mathcal{S} = (S, A, R)$; as usual, its set of pseudo-terms is denoted by T.

Definition 14 *A* variable labelling *is a map $\epsilon : V \rightarrow T$ is such that the set $\{x \in V | \epsilon x = t\}$ is infinite for every $t \in T$.*

Of course, such maps always exist if V is sufficiently large (the cardinal of V is determined by the cardinal of S). One nice aspect of variable labelling is that it eliminates the need to manipulate contexts. In the sequel, we assume we are given a fixed labelling ϵ. We can define a notion of derivation w.r.t. ϵ; the rules are

Axiom	$\dfrac{}{\vdash_\epsilon c : s}$	if $\mathsf{datax}\ A = s$ and $c \in A$ or $\mathsf{sortax}\ c = s$
Function	$\dfrac{\vdash t_i : \sigma_i \quad \text{for } i = 1, \ldots, n}{\vdash f(t_1, \ldots, t_n) : \tau}$	if f is a function symbol of datum $\sigma_1, \ldots, \sigma_n \to \tau$
Start	$\dfrac{\vdash_\epsilon A : s}{\vdash_\epsilon x : A}$	if $\epsilon x \equiv A$ and x is fresh in A
Product	$\dfrac{\vdash_\epsilon A : s_1 \quad \vdash_\epsilon B : s_2}{\vdash_\epsilon \Pi x : A.B : s_3}$	if $\epsilon x \equiv A$ and $(s_1, s_2, s_3) \in R$
Application	$\dfrac{\vdash_\epsilon t : \Pi x : A.B \quad \vdash_\epsilon u : A}{\vdash_\epsilon tu : B[u/x]}$	
Abstraction	$\dfrac{\vdash_\epsilon t : B \quad \vdash_\epsilon \Pi x : A.B}{\vdash_\epsilon \lambda x : A.t : \Pi x : A.B}$	
Conversion	$\dfrac{\vdash_\epsilon u : A \quad \vdash_\epsilon B : s}{\vdash_\epsilon u : B}$	if $A \twoheadrightarrow_{\beta R} B$ or $B \twoheadrightarrow_{\beta R} A$

It is not difficult to check that (algebraic) pure type systems with variable labelling are essentially equivalent to (algebraic) pure type systems.

Proposition 15 – *If $\vdash_\epsilon M : A$, then $\Gamma \vdash M : A$ for some context Γ.*
– *If $\Gamma \vdash M : A$, then $\vdash_\epsilon \rho M : \rho A$ for some variable renaming ρ.*

It follows that strong normalisation and subject reduction of the system with labelled variables (or labelled system for short) is equivalent to strong normalisation and subject reduction of the original system. Besides, one can reformulate the criterion for systems with labelled variables.

Definition 16 *Let $\lambda\mathcal{S}$ be an algebraic pure type system with a variable labelling ϵ. A* prototype *is a pseudo-term M for which there exist $N_1, \ldots, N_p \in$ Pseudo and $s \in S$ such that*

$$\vdash_\epsilon M\ N_1\ \ldots\ N_p : s$$

The set of prototypes is denoted by Proto. As before, we consider the relation $\prec$ defined as the smallest relation such that

$$\forall M, N \in \mathsf{Pseudo}.(M\ N) \in \mathsf{Proto} \Rightarrow N \prec M\ \wedge\ (M\ N) \prec M$$

Definition 17 *$\lambda\mathcal{S}$ is* stratified *if the relation $\prec$ is well-founded.*

Theorem 11 can now be rephrased as:

Theorem 18 *Every $\mathcal{R}$-terminating stratified labelled pure type system with the subject reduction property is strongly normalising.*

Theorem 11 follows easily from Theorem 18.

4 The proof of Theorem 18

In this section, we prove Theorem 18. The proof is divided in two parts: in the first part, we prove that algebraic reduction is strongly normalising on legal terms. In the second part, we give a model-construction for stratified algebraic pure type systems. Strong normalisation is derived easily from the model construction.

4.1 Strong normalisation of algebraic reduction

Strong normalisation of algebraic reduction on legal terms can be established in a straightforward fashion by advocating modularity results from [11] for example. The technique is inspired from [4] and consists of viewing λ-calculus as an algebraic signature. In this way, we define for every $\mathcal{R}$-algebraic pure type system $\lambda\mathcal{S} = (\mathcal{R}, S, \mathsf{sortax}, \mathsf{rules}, \mathsf{datax})$ an algebraic signature $\Sigma_{\lambda\mathcal{S}}$ extending the signatures of the rewrite systems and upon which algebraic reduction is terminating. Then we show that all legal terms can be obtained from the terms of $\Sigma_{\lambda\mathcal{S}}$ by an erasure map $|.|$ which reflects reduction. Strong normalisation of algebraic reduction on legal terms follows easily. In the sequel, we consider a finite sequence of terminating higher-order rewriting systems $\mathcal{R}_i = (\Lambda_i, \Sigma_i, R_i)$ for $i = 1, \ldots, n$. Let $\Lambda = \bigcup_{i=1,\ldots,n} \Lambda_i$ and let $\Sigma_{\lambda\mathcal{S}} = (\bigcup_{i=1,\ldots,n} \Sigma_i) \cup \Sigma_0$ where Σ_0 is the signature with function symbols:

- $\overline{s}_\tau : \tau$ for $s \in \{*, \Box\}$ and $\tau \in \Xi$,
- $\overline{\Pi}_{x,\tau_1,\tau_2,\tau_3}$, $\overline{\lambda}_{x,\tau_1,\tau_2,\tau_3} : \tau_1 \times \tau_2 \rightarrow \tau_3$ for every variable x and $\tau_1, \tau_2, \tau_3 \in \Xi$,
- $\mathsf{Appl}_{\tau_1,\tau_2,\tau_3} : \tau_1 \times \tau_2 \rightarrow \tau_3$ for every $\tau_1, \tau_2, \tau_3 \in \Xi$.

The union R_0 of the R_i's can be seen as a higher-order rewriting system over $\Sigma_{\lambda\mathcal{S}}$. Moreover R_0 is terminating.

Proposition 19 *$\rightarrow_R$ is strongly normalising on legal terms.*

Proof: we define a map from the terms of $\Sigma_{\lambda\mathcal{S}}$ to pseudo-terms. For the sake of simplicity, we assume that the set of variables for every sort τ is $\{x^\tau \mid x \in V\}$. The map $\lceil . \rceil$ is defined as follows:

$$\begin{aligned}
\lceil x^\tau \rceil &= x \\
\lceil f(t_1, \ldots, t_n) \rceil &= f(\lceil t_1 \rceil, \ldots, \lceil t_n \rceil) \\
\lceil \overline{\Pi}_{x,\tau_1,\tau_2,\tau_3}(t_1, t_2) \rceil &= \Pi x : \lceil t_1 \rceil . \lceil t_2 \rceil \\
\lceil \overline{\lambda}_{x,\tau_1,\tau_2,\tau_3}(t_1, t_2) \rceil &= \lambda x : \lceil t_1 \rceil . \lceil t_2 \rceil \\
\lceil \mathsf{Appl}_{\tau_1,\tau_2,\tau_3}(t_1, t_2) \rceil &= \lceil t_1 \rceil \ \lceil t_2 \rceil
\end{aligned}$$

The map is surjective on the set of legal terms. Moreover, every infinite R-reduction sequence on pseudo-terms can be lifted to an infinite R_0-reduction sequence on the terms of $\Sigma_{\lambda\mathcal{S}}$.$\Box$

4.2 The model construction

In this section, we present a model construction for stratified aptss with the subject reduction property. The construction is based on saturated sets and is a generalisation of strong normalisation proofs for pure type systems, such as the polymorphic λ-calculus ([16, 23, 12]) or the calculus of constructions ([14, 24]). The model is heavily inspired by [24]. Before giving a proof of Theorem 18, we need some preliminaries on saturated sets.

Saturated sets Traditionally, saturated sets are defined as sets of β-strongly normalisable untyped λ-terms. Here we consider a slightly different notion of saturated sets, more adapted to our framework: we define saturated sets as sets of pseudo-terms rather than sets of λ-terms. This is not really important but makes the proof slightly more elegant. Moreover, we consider typed saturated sets as in [19, 24] rather than untyped saturated sets. This means that the notion of saturated sets is defined relative to a set of pseudo-terms. This is not important for pure type systems but turns out to be crucial for algebraic pure type systems (otherwise, we cannot use the results of the principal case).

Recall that a pseudo-term M is *strongly normalising* if all reduction sequences starting from M are finite. The set of strongly normalising terms is denoted by SN. Saturated sets will be defined as subsets of SN with certain closure properties.

Definition 20 *A* base term *is a term of the form $x\ P_1\ \ldots\ P_n$ where $x \in V$ and $P_1, \ldots, P_n \in \mathsf{SN}$.*

The set of base terms is denoted by Base. Note that all base terms are strongly normalising.

Definition 21 Key-reduction $\rightarrow_k$ *is the smallest relation on pseudo-terms such that for every pseudo-terms $M, N, O, P_1, \ldots, P_n$*

$$(\lambda x : M.N)\ O\ P_1\ \ldots\ P_n \rightarrow_k N[O/x]\ P_1\ \ldots\ P_n$$

Note that a term has at most one key-redex. The term obtained from M by contracting its key redex is denoted by $\mathbf{kred}(M)$.

Definition 22 *Let $U \subseteq \mathsf{Pseudo}$. A set X of pseudoterms is* saturated *in U if :*

(i) $X \subseteq \mathsf{SN} \cap U$;
(ii) $\mathsf{Base} \cap U \subseteq X$;
(iii) If $\mathbf{kred}(M) \in X$ *and* $M \in \mathsf{SN} \cap U$*, then* $M \in X$.

The collection of all saturated sets in U is denoted by $SAT(U)$. In the sequel, we will use $SAT(M)$ for $M \in \mathsf{Pseudo}$ to denote the set of saturated sets in $\{N \in \mathsf{Pseudo} \mid\ \vdash N : M\}$. If $X \in \mathsf{SAT}(M)$, we will say X is a *M-saturated set.* We list some closure properties of saturated sets.

Fact 23 *Let $U, U' \subseteq$ Pseudo.*

- $\mathsf{SN}(U) = \mathsf{SN} \cap U$ *is a saturated set in U.*
- *The set of saturated sets in U is closed under arbitrary non-empty intersections.*
- *If X is saturated in U and Y is saturated in U', then $X \to Y$ defined by*

$$X \to Y = \{M \in W | \forall N \in X.M\ N \in Y\}$$

is saturated in W provided that $\mathsf{Base} \cap W \subset X \to Y$ *(i.e. for every $w \in$* $\mathsf{Base} \cap W$ *and $x \in X$, $wx \in Y$).*
-
- *If X is saturated in U and Y_x is saturated in U'_x for $x \in X$, then $\Pi x \in X.Y_x$ defined by*

$$\Pi x \in X.Y_x = \{M \in W | \forall N \in X.M\ N \in Y_N\}$$

is saturated in W provided $\mathsf{Base} \cap W \subset \Pi x \in X.Y_x$ *(i.e. for every $w \in$* $\mathsf{Base} \cap W$ *and $x \in X$, $wx \in Y_x$).*

If $M \in$ Pseudo, then $\mathsf{SN}(M)$ is the saturated set of strongly normalising terms of type M.

The principal case The key fact in the model construction for algebraic pure type systems is that the sets of strongly normalising terms of base datum enjoy suitable closure properties.

Proposition 24 *Let f be a function symbol of datum $(\sigma_1, \ldots, \sigma_n \to \tau)$. Then for all pseudo-terms $t_1, \ldots, t_n$,*

$$t_i \in \mathsf{SN}(\sigma_i) \quad \text{for } i = 1, \ldots, n \quad \Rightarrow \quad f(t_1, \ldots, t_n) \in \mathsf{SN}(\tau)$$

The proof is an adaptation of [18, 1]. This key fact ensures that the model construction for algebraic pure type systems can be carried out in exactly the same way as for pure type systems.

Intuition behind the proof The idea of the proof is to give a model construction in which types are interpreted as (saturated) sets and legal terms as pseudo-terms such that the following soundness condition is satisfied:

$$\vdash_\epsilon M : A \quad \Rightarrow \quad ([M]) \in \langle\langle A \rangle\rangle$$

where $\langle\langle A \rangle\rangle$ is the saturated set interpretation of A and $([M])$ is the pseudo-term interpretation of M. For simple systems, such as the (algebraic) simply typed λ-calculus $\lambda_\to$, the definition of $\langle\langle A \rangle\rangle$ can be given inductively on the structure of the terms and the soundness condition can be proved inductively. For the polymorphic λ-calculus $\lambda 2$, one is forced to parameterise interpretations by valuations. One then has to prove that if a valuation ρ satisfies certain properties, then

$$\vdash_\epsilon M : A \quad \Rightarrow \quad ([M])_\rho \in \langle\langle A \rangle\rangle_\rho$$

In a system with dependent types such as λP or $\lambda P\omega$, terms might occur in types so one cannot any longer define $\langle\!\langle A \rangle\!\rangle$ inductively. The standard solution is to define $\langle\!\langle A \rangle\!\rangle$ as a partial interpretation and show that it is well-defined on legal types. This requires the introduction of a new interpretation $a(M)$ which assigns to a term its possible values. The idea is that $a(M)$ should be defined for every type and be a set of saturated sets such that under suitable conditions

$$\vdash_\epsilon M : A \quad \Rightarrow \quad ([M])_\rho \in \langle\!\langle A \rangle\!\rangle_{\rho,\zeta}$$

Note that in this context valuations are of the form (ρ, ζ) where ρ assigns to every variable (in some domain) a pseudo-term and ζ assigns to every variable (in some domain) a saturated set. Note that dependent types introduce a new difficulty: we have indexed families of types, i.e. terms of type $B \to *$[3]. These terms, which we have defined earlier as prototypes, will also need to be intepreted. To be able to interpret them as families of types, we must use induction on their structure: if M is of type $B \to C \to *$, we want to define $a(M)$ as the set of families of maps $f_b : a(b) \to a(M\ b)$ for $b \in B$. This requires $a(b)$ and $a(M\ b)$ to be already defined. This requirement matches exactly the definition of $\prec$: the assumption that $\prec$ is well-founded enables us to define the interpretation $a(M)$ by $\prec$-induction. The other two interpretations will be defined as usual by induction on the structure of the terms.

Convention From now on, we will drop the subscript in $\vdash_\epsilon$.

The construction The set Data of data is defined as the union of the set of sorts of the rewriting systems. The set Type of types is defined by

$$\mathsf{Type} = \{M \in \mathsf{Pseudo} \mid\ \vdash M : s \text{ for some } s \in S\}$$

The map $a : \mathsf{Pseudo} \to \mathit{Set}$ is defined by case distinction:

- if $M \in \mathsf{Type} \setminus \mathsf{Data}$, $a(M) = \mathsf{SAT}(M)$;
- if $M \in \mathsf{Proto}$, $a(M) = \{(f_B)_{B \in \mathsf{cone}(M)} | f_B : a(B) \to a(M\ B)\}$;
- if $M \in \mathsf{Data}$, $a(M) = \{\mathsf{SN}(M)\}$;
- otherwise, $a(M) = \{\{\emptyset\}\}$;

where $\mathsf{cone}(M) = \{B \in \mathsf{Pseudo} | (M\ B) \in \mathsf{Proto}\}$. Define $\mathbb{A} = \bigcup_{M \in \mathsf{Pseudo}} a(M)$.

Definition 25 *A* valuation *is a pair (ρ, ζ) such that $\rho : V \to \mathsf{Pseudo}$ and $\zeta : V \to \mathbb{A}$.*

The extension $([.])_\rho : \mathsf{Pseudo} \to \mathsf{Pseudo}$ of ρ is defined as the unique capture-avoiding substitution extending ρ. We can extend ζ to terms by defining a map $\langle\!\langle . \rangle\!\rangle_{\rho\zeta} : \mathsf{Pseudo} \to \mathbb{A}$ as follows:

[3] This is not only true for dependent types but also for higher-order polymorphism as it occurs in $\lambda\omega$.

$$\begin{array}{rll}
\langle\!\langle x\rangle\!\rangle_{\rho\zeta} &= \zeta(x) & \text{if } x \in V \text{ and } \rho(x) \in \mathsf{Proto}\\
\langle\!\langle \Pi x : A.B\rangle\!\rangle_{\rho\zeta} &= \{P \in \mathsf{Pseudo} | \forall (N,Q) \in \mathcal{E}_{\rho\zeta}(A). & \\
& \qquad\qquad PN \in \langle\!\langle B\rangle\!\rangle_{\rho(x:=N),\zeta(x:=Q)})\} & \text{if } (\![\Pi x : A.B]\!)_\rho \in \mathsf{Type}\\
\langle\!\langle M\ N\rangle\!\rangle_{\rho\zeta} &= (\langle\!\langle M\rangle\!\rangle_{\rho\zeta})_{(\![N]\!)_\rho}\ \langle\!\langle N\rangle\!\rangle_{\rho\zeta} & \text{if } (\![MN]\!)_\rho \in \mathsf{Proto}\\
\langle\!\langle \lambda x : A.b\rangle\!\rangle_{\rho\zeta} &= (\boldsymbol{\lambda} c \in a(B).\langle\!\langle b\rangle\!\rangle_{\rho(x:=B),\zeta(x:=c)})_{B\in\mathsf{cone}((\![\lambda x:A.b]\!)_\rho)} & \text{if } (\![\lambda x : A.b]\!)_\rho \in \mathsf{Proto}\\
\langle\!\langle M\rangle\!\rangle_{\rho\zeta} &= \mathsf{SN}(\mathrm{M}) & \text{if } M \in \mathsf{Data}\\
\langle\!\langle M\rangle\!\rangle_{\rho\zeta} &= \{\emptyset\} & \text{otherwise}
\end{array}$$

where for every $M \in \mathsf{Pseudo}$,

$$\mathcal{E}_{\rho\zeta}(M) = \{(N,Q) \in \mathsf{Pseudo} \times \mathbb{A} \mid\ \vdash N : (\![M]\!)_\rho,\ N \in \langle\!\langle M\rangle\!\rangle_{\rho\zeta},\ Q \in a(N)\}$$

The following lemma is easily established by induction on the structure of M.

Lemma 26 *Let $M, N \in \mathsf{Pseudo}$. Let (ρ,ζ) and (ρ',ζ') be two valuations.*

- *If $\rho x = \rho' x$ and $\zeta x = \zeta' x$ for every $x \in \mathsf{FV}(M)$, then $\langle\!\langle M\rangle\!\rangle_{\rho\zeta} = \langle\!\langle M\rangle\!\rangle_{\rho'\zeta'}$.*
- *$\langle\!\langle M[N/x]\rangle\!\rangle_{\rho\zeta} = \langle\!\langle M\rangle\!\rangle_{\rho(x:=(\![N]\!)_\rho),\zeta(x:=\langle\!\langle N\rangle\!\rangle_{\rho\zeta})}$*

As a consequence of Lemma 26 and of the subject reduction property, we conclude that $\langle\!\langle .\rangle\!\rangle_{\rho\zeta}$ is invariant under reduction on legal terms.

Corollary 27 *For every valuation (ρ,ζ) and terms M, N such that $M \rightarrow_{mix} N$ and $(\![M]\!)_\rho, (\![N]\!)_\rho \in \mathsf{Proto}$, we have $\langle\!\langle M\rangle\!\rangle_{\rho\zeta} = \langle\!\langle N\rangle\!\rangle_{\rho\zeta}$.*

In order to prove the main theorem, we must establish that the model behaves as expected. It requires a standard soundness argument. In the sequel, we call a context a finite list of variables $\Delta = y_1, \ldots, y_n$ such that for $i = 1, \ldots, n$, $y_i \notin \mathsf{FV}(\epsilon y_j)$ $(\forall j \leq i)$. One can check that for every well-typed term M, $\mathsf{FV}(M)$ can be ordered into a context.

Definition 28 *Let Δ be a context. A valuation (ρ,ζ)* satisfies *Δ (denoted $(\rho,\zeta) \models \Delta$) if for every $x \in \Delta$,*

(i) $\vdash \rho x : (\![\epsilon x]\!)_\rho$,
(ii) $\rho x \in \langle\!\langle \epsilon x\rangle\!\rangle_{\rho\zeta}$,
(iii) $\langle\!\langle x\rangle\!\rangle_{\rho\zeta} \in a((\![x]\!)_\rho)$.

We say that $\models M : A$ if

(i) $\vdash (\![M]\!)_\rho : (\![A]\!)_\rho$,
(ii) $(\![M]\!)_\rho \in \langle\!\langle A\rangle\!\rangle_{\rho\zeta}$,
(iii) $\langle\!\langle M\rangle\!\rangle_{\rho\zeta} \in a((\![M]\!)_\rho)$,

for every valuation (ρ,ζ) satisfying $\mathsf{FV}(M) \cup \mathsf{FV}(A)$.

Fact 29 *Let (ρ,ζ) be a valuation satisfying Δ. Let $x \notin \Delta$ and $x \notin \mathsf{FV}(\epsilon y)$ for all $y \in \Delta$. Then for every $C \in a(x)$, $\rho(x := x), \zeta(x := C)$ satisfies $\Delta \cup \{x\}$.*

As $a(x) \neq \emptyset$, valuations can always be extended to a larger context while preserving satisfaction. We can now prove the main technical result of this paper.

Proposition 30 (Soundness) $\vdash M : A \quad \Rightarrow \quad \models M : A.$

Proof: by induction on the length of derivations.

- *Axiom:* if $\vdash s_1 : s_2$ is an axiom, then it is easy to show $\models s_1 : s_2$.
- *Start:* assume $\vdash x : A$ is deduced from $\vdash A : s$ by a start rule. Then $\epsilon x = A$. Assume (ρ, ζ) satisifies $\mathsf{FV}(A) \cup \{x\}$. By definition of satisfaction, $\vdash \rho x : ([A])_\rho$, $\rho x \in \langle\langle A \rangle\rangle_{\rho\zeta}$ and $\langle\langle x \rangle\rangle_{\rho\zeta} \in a(\rho x)$, so we are done.
- *Function symbol:* assume $\vdash f(t_1, \ldots, t_n) : \tau$ is deduced by a function rule from $\vdash t_i : \sigma_i$ for $i = 1, \ldots, n$ where f is a function symbol of datum $(\sigma_1, \ldots, \sigma_n \to \tau)$. Assume $(\rho, \zeta) \models \mathsf{FV}(f(t_1, \ldots, t_n))$.
  $\vdash ([f(t_1, \ldots, t_n)])_\rho : \tau$ follows immediately from the induction hypothesis.
  Next one has to prove that $([f(t_1, \ldots, t_n)])_\rho \in \langle\langle \tau \rangle\rangle_{\rho\zeta}$. This is an immediate consequence of Lemma 24.
  Finally, we need to prove $\langle\langle f(t_1, \ldots, t_n) \rangle\rangle_{\rho\zeta} \in a(([f(t_1, \ldots, t_n)])_\rho)$. This is easy because $([f(t_1, \ldots, t_n)])_\rho \notin \mathsf{Proto}$.
- *Product:* assume $\vdash \Pi x : A.B : s_3$ is deduced by a formation rule from $\vdash A : s_1$ and $\vdash B : s_2$. Let (ρ, ζ) be a valuation such that$(\rho, \zeta) \models \mathsf{FV}(\Pi x : A.B)$.
  We prove $\vdash ([\Pi x : A.B])_\rho : s_3$. By induction hypothesis, $\vdash ([A])_\rho : s_1$. By fact 29,
  $$\rho(x := x), \zeta(x := C) \models \mathsf{FV}(\Pi x : A.B) \cup \{x\}$$
  for every $C \in a(x)$. Hence $\vdash ([B])_{\rho,(x:=x)} : s_2$ by induction hypothesis. By the product rule, $\vdash \Pi x : ([A])_\rho.([B])_{\rho,(x:=x)} : s_3$. As $\Pi x : ([A])_\rho.([B])_{\rho,(x:=x)} = ([\Pi x : A.B])_\rho$, we conclude (i) holds.
  Next we show $([\Pi x : A.B])_\rho \in \langle\langle s_3 \rangle\rangle_{\rho\zeta}$. By definition of $\langle\langle . \rangle\rangle_{\rho\zeta}$, it is equivalent to show that $([\Pi x : A.B])_\rho$ is strongly normalising (we already know that (i) holds). By induction hypothesis, $([A])_\rho \in \langle\langle s_1 \rangle\rangle_{\rho\zeta} \subseteq \mathsf{SN}$ and $([B])_{\rho'} \in \langle\langle s_2 \rangle\rangle_{\rho'\zeta'} \subseteq \mathsf{SN}$ for every valuation (ρ', ζ') satisfying $\mathsf{FV}(B)$. Let $C \in a(x)$. Then $\rho(x := x), \zeta(x := C) \models \mathsf{FV}(\Pi x : A.B) \cup \{x\}$. Hence $([B])_{\rho(x:=x)} \in \mathsf{SN}$ and $([\Pi x : A.B])_\rho \in \mathsf{SN}$.
  Finally, we show $\langle\langle \Pi x : A.B \rangle\rangle_{\rho\zeta} \in a(([\Pi x : A.B])_\rho)$. By (i), we know that $([\Pi x : A.B])_\rho \in \mathsf{Type}$, so we have to prove that $\langle\langle \Pi x : A.B \rangle\rangle_{\rho\zeta}$ is a $([\Pi x : A.B])_\rho$-saturated set. As $([A])_\rho$ is a type, it follows by induction hypothesis that $\langle\langle A \rangle\rangle_{\rho\zeta}$ is a $([A])_\rho$-saturated set. Besides, $([B])_{\rho(x:=x)}$ is a type and by the substitution lemma, $([B])_{\rho(x:=N)}$ is a type whenever $\vdash N : \epsilon x$. Hence $\langle\langle B \rangle\rangle_{\rho(x:=N),\zeta(x:=Q)}$ is a $([B])_{\rho(x:=N)}$-saturated set whenever $\rho(x := N), \zeta(x := Q) \models \mathsf{FV}(B)$ (equivalently for every $(N, Q) \in \mathcal{E}_{\rho\zeta}(A)$). We conclude $\langle\langle \Pi x : A.B \rangle\rangle_{\rho\zeta}$ is a $([\Pi x : A.B])_\rho$-saturated set.
- *Application:* assume $\vdash M\ N : B[N/x]$ is deduced from $\vdash M : \Pi x : A.B$ and $\vdash N : A$ by an application rule. Let (ρ, ζ) be a valuation satisfying $\mathsf{FV}(M) \cup \mathsf{FV}(B[N/x])$.
  First, we show that $\vdash ([MN])_\rho : ([B[N/x]])_\rho$. Consider the valuation (ρ', ζ') defined by
  $$\rho' y = \begin{cases} \rho y \text{ if } y \in \mathsf{FV}(M) \cup \mathsf{FV}(B[N/x]) \\ y \quad \text{otherwise} \end{cases}$$

and

$$\zeta' y = \begin{cases} \zeta y \text{ if } y \in \mathsf{FV}(M) \cup \mathsf{FV}(B[N/x]) \\ C_y \text{ otherwise} \end{cases}$$

where C_y is an arbitrary element of $a(y)$. Then

$$(\rho', \zeta') \models \mathsf{FV}(MN) \cup \mathsf{FV}(\Pi x : A.B)$$

By induction hypothesis, we have
- $\vdash ([M])_{\rho'} : ([\Pi x : A.B])_{\rho'}$;
- $\vdash ([N])_{\rho'} : ([A])_{\rho'}$.

Hence $\vdash ([MN])_{\rho'} : ([B])_{\rho',(x:=x)}[([N])_{\rho'}/x]$. In other words, $\vdash ([MN])_{\rho'} : ([B[N/x]])_{\rho'}$. As ρ and ρ' coincide on $\mathsf{FV}(M) \cup \mathsf{FV}(B[N/x])$, we conclude that (i) holds.
Next, we show that $([MN])_\rho \in \langle\langle B[N/x]\rangle\rangle_{\rho\zeta}$. Note that it is equivalent to show $([MN])_{\rho'} \in \langle\langle B[N/x]\rangle\rangle_{\rho'\zeta'}$ where (ρ', ζ') is defined as above. By induction hypothesis, we know that $\vdash ([N])_{\rho'} : ([A])_{\rho'}$, $([N])_{\rho'} \in \langle\langle A\rangle\rangle_{\rho'\zeta'}$ and $\langle\langle N\rangle\rangle_{\rho'\zeta'} \in a(([N])_{\rho'})$. Hence, $(([N])_{\rho'}, \langle\langle N\rangle\rangle_{\rho'\zeta'}) \in \mathcal{E}_{\rho'\zeta'}(A)$. By induction hypothesis, $([M])_{\rho'} \in \langle\langle \Pi x : A.B\rangle\rangle_{\rho'\zeta'}$. Hence

$$([MN])_{\rho'} \in \langle\langle B\rangle\rangle_{\rho'(x:=([N])_{\rho'}),\zeta'(x:=\langle\langle N\rangle\rangle_{\rho'\zeta'})}$$

By Lemma 26, $\langle\langle B[N/x]\rangle\rangle_{\rho'\zeta'} = \langle\langle B\rangle\rangle_{\rho'(x:=([N])_{\rho'}),\zeta'(x:=\langle\langle N\rangle\rangle_{\rho'\zeta'})}$. So we are done.
Finally, we prove that $\langle\langle MN\rangle\rangle_{\rho\zeta} \in a(([MN])_\rho)$. There are two cases two distinguish. If $([MN])_\rho \notin \mathsf{Proto}$, then $a(([MN])_\rho) = \{\{\emptyset\}\}$ and $\langle\langle MN\rangle\rangle_{\rho\zeta} = \{\emptyset\}$, so we are done. Otherwise, $([M])_\rho \in \mathsf{Proto}$. By induction hypothesis, $\langle\langle M\rangle\rangle_{\rho\zeta} \in a(([M])_\rho)$ and $\langle\langle N\rangle\rangle_{\rho\zeta} \in a(([N])_\rho)$. Hence $(\langle\langle M\rangle\rangle_{\rho\zeta})_{([N])_\rho} \langle\langle N\rangle\rangle_{\rho\zeta} \in a(([MN])_\rho)$.

- *abstraction:* assume $\vdash \lambda x : A.t : \Pi x : A.B$ is deduced by an abstraction rule from $\vdash t : B$ and $\vdash \Pi x : A.B : s$. Let (ρ, ζ) be a valuation satisfying $\mathsf{FV}(\lambda x : A.t) \cup \mathsf{FV}(\Pi x : A.B)$.
We prove $\vdash ([\lambda x : A.t])_\rho : ([\Pi x : A.B])_\rho$. By induction hypothesis, $\vdash ([\Pi x : A.B])_\rho : s$. By Fact 29, $\rho(x := x), \zeta(x := C) \models \mathsf{FV}(t)$ for every $C \in a(x)$. Hence $\vdash ([t])_{\rho(x:=x)} : ([A])_{\rho(x:=x)}$. As x is not free in A, we have $([A])_{\rho(x:=x)} = ([A])_\rho$. We can apply the abstraction rule to conclude.
Next we prove that $([\lambda x : A.t])_\rho \in \langle\langle \Pi x : A.B\rangle\rangle_{\rho\zeta}$. This amounts to showing that for every $(N, Q) \in \mathcal{E}_{\rho\zeta}(A)$, we have

$$([\lambda x : A.t])_\rho\ N \in \langle\langle B\rangle\rangle_{\rho(x:=N),\zeta(x:=Q)}$$

By definition of saturated sets, this follows from

$$([t])_{\rho(x:=N)} \in \langle\langle B\rangle\rangle_{\rho(x:=N),\zeta(x:=Q)}$$

which is a direct consequence of the induction hypothesis.
Finally we prove $\langle\langle \lambda x : A.t\rangle\rangle_{\rho\zeta} \in a(([\lambda x : A.t])_\rho)$. There are two cases to distinguish. If $([\lambda x : A.t])_\rho \notin \mathsf{Proto}$, this is an easy consequence of the definitions. Otherwise, we have to prove that for every $B \in \mathsf{cone}(([\lambda x : A.t])_\rho)$ and

$c \in a(B)$, $\langle\!\langle t\rangle\!\rangle_{\rho(x:=B),\zeta(x:=c)} \in a((\![\lambda x : A.t]\!)_\rho B)$. By the generation lemma, it follows that $\vdash B : (\![A]\!)_\rho$, hence $(\rho(x := B), \zeta(x := c))$ satisfies $\mathsf{FV}(t)$. The result is a consequence of the induction hypothesis.

- *expansion/reduction:* assume $\vdash M : B$ is deduced from $\vdash M : A$ and $\vdash B : s$ using the expansion/reduction rule. Let (ρ, ζ) be a valuation satisfying $\mathsf{FV}(M) \cup \mathsf{FV}(B)$. As before, we can extend the valuation into a new valuation (ρ', ζ') such that (ρ', ζ') satisfies $\mathsf{FV}(M) \cup \mathsf{FV}(B) \cup \mathsf{FV}(A)$ and coincides with (ρ, ζ) on $\mathsf{FV}(M) \cup \mathsf{FV}(B)$.
  To prove $\vdash (\![M]\!)_{\rho'} : (\![B]\!)_{\rho'}$, note that $(\![A]\!)_{\rho'} \to (\![B]\!)_{\rho'}$ or $(\![B]\!)_{\rho'} \to (\![B]\!)_{\rho'}$. Besides, it follows from the induction hypothesis that:
  - $\vdash (\![M]\!)_{\rho'} : (\![A]\!)_{\rho'}$;
  - $\vdash (\![B]\!)_{\rho'} : s$.

  We conclude by the conversion rule.
  To prove $(\![M]\!)_\rho \in \langle\!\langle B\rangle\!\rangle_{\rho\zeta}$, we just apply Corollary 27.
  Finally, $\langle\!\langle M\rangle\!\rangle_{\rho\zeta} \in a((\![M]\!)_\rho)$ is immediate from the induction hypothesis.□

Corollary 31 $\vdash M : A \quad \Rightarrow \quad M \in \mathsf{SN}$.

Proof: for every derivation $\vdash M : A$, consider the valuation (ρ, ζ) such that $\rho(x) = x$ for every $x \in V$ and $\zeta(x) = \mathsf{max}(x)$ where max is defined on pseudo-terms by $\prec$-induction:

- if $M \in \mathsf{Type}$, $\mathsf{max}(M) = \mathsf{SN}(M)$;
- if $M \in \mathsf{Proto}$, $\mathsf{max}(M) = (\lambda x : a(B).\mathsf{max}(M\ B))_{B \in \mathsf{cone}(M)}$;
- otherwise, $\mathsf{max}(M) = \{\emptyset\}$.

Then $(\rho, \zeta) \models \mathsf{FV}(M) \cup \mathsf{FV}(A)$. It follows from Proposition 30 that $M \in \langle\!\langle A\rangle\!\rangle_{\rho,\zeta}$. As $\langle\!\langle A\rangle\!\rangle_{\rho,\zeta} \subseteq \mathsf{SN}$, we conclude.

5 Applications of the main theorem

Theorem 11 has several important consequences. On the one hand, we recover all the known results about algebraic pure type systems. On the other hand, we obtain new results for algebraic higher-order logic and for the calculus of constructions with infinitely many universes:

- Systems of the algebraic λ-cube are strongly normalising provided R-reduction is strongly normalising on algebraic terms ([2, 18]).
- Algebraic higher-order logic is strongly normalising provided R-reduction is strongly normalising on algebraic terms.
- The algebraic calculus of constructions with universes are strongly normalising provided all rewrite systems are left-linear and R-reduction is strongly normalising on algebraic terms.

These results follow from Theorem 11 by proving that the systems are stratified (we already know that the have the subject reduction property). For the algebraic calculus of constructions and the systems of the algebraic cube, this is rather easy. A prototype can only be of type kind and kinds are of the form:

- $*$,
- $\Pi x : A.B$ where A and B are kinds,
- $\Pi x : A.B$ where B is a kind and A is a type.

Note that we are implicitely assuming that algebraic data live in $*$ as in [2]; it is easy to adapt the proof to the other case. One can define a measure ν on kinds as follows:

- $\nu(*) = 1$,
- $\nu(\Pi x : A.B) = \nu(A) + \nu(B) + 1$ if A and B are kinds,
- $\nu(\Pi x : A.B) = \nu(B) + 1$ if B is a kind and A is a type.

Note that the measure is preserved by conversion. By uniqueness of types, this yields a measure μ on prototypes: define $\mu(M) = n$ if for some A, $\vdash (M) : A$ and $\nu(A) = n$. Then for every P, Q,

$$P \prec Q \Rightarrow \mu(P) < \mu(Q)$$

Hence the systems of the algebraic λ-cube are stratified. A similar technique applies to algebraic higher-order logic.

For the algebraic calculus of constructions with universes, the proof is more involved and requires a quasi-normalisation argument, as developed in [19]. The quasi-normalisation theorem shows that every type has a weak head normal form. This enables us to give a measure on types. As before, we can invoke uniqueness of types to turn this measure into a measure μ for prototypes with the property that $P \prec Q \Rightarrow \mu(P) < \mu(Q)$ for every pseudo-terms P, Q. Note that in this case it is crucial to know subject reduction and confluence of reduction on normal terms before the strong normalisation proof so we must restrict ourselves to left-linear rewriting system. For such systems, the combined reduction is confluent on the set of pseudo-terms of the algebraic pure type system (this follows from [21]).

We want to close this section by making a few remarks about the generality of the criterion. The criterion is not as general as it could seem. We believe that a pure type system (with a countable set of sorts) is stratified if and only if it can be embedded in the calculus of constructions with universes. One can easily find pure type systems which are strongly normalising without being stratified. The easiest example is probably obtained by adding to the polymorphic λ-calculus a new sort $\triangle$ and an axiom $\triangle : *$. So not every strongly normalising pure type system is stratified. Yet every pure type system of interest is stratified and our proof therefore applies to all of those systems.

6 Conclusion

We have introduced in the unified framework of algebraic pure type systems a large class of algebraico-functional languages which includes all the systems considered in the literature so far. In this general framework, we have been able to address modularity questions. We have given a general criterion for algebraic pure type systems to be strongly normalising and shown that all the usual algebraic pure type systems meet this criterion. One nice aspect of the proof is that it gives a uniform treatment of all the usual algebraic pure type systems and emphasizes the fact that proving strong normalisation for algebraic pure type systems is not essentially more difficult than proving strong normalisation for pure type systems. It would be interesting to extend the present work to more powerful type systems: possible extensions to be considered are first-order inductive types (i.e. inductive types generated by first-order signatures, see for example [22]), congruence types (an extension of algebraic pure type systems in which data come equipped with an elimination principle, see [5])... However, we feel more enclined to focus on two important problems which remain unsolved:

- there is no direct proof of subject reduction in algebraic pure type systems. This is a serious drawback of the framework which we hope could be remedied. However, we do not know of any proof technique which would solve the problem. Note that a positive answer to the Expansion Postponement problem ([26]) could yield a positive solution to our problem.
- the approach we chose here is uniform in the sense that algebraic pure type systems are treated simultaneously with pure type systems. Yet in practice, one would like to know that an algebraic pure type system is strongly normalising if its underlying pure type system is. Note that such a result would require a purely syntactic proof as no assumption is made on the algebraic pure type system. One idea would be to try to use a generalisation of Dougherty's results ([9]). However, it requires to prove subject reduction and also that the algebraic pure type system is strongly normalising with β-reduction. One approach would be to try to define a β-reduction-preserving mapping from the algebraic pure type system to its underlying pure type system.

Acknowledgements

This work was partially supported by the Esprit BRA project "TYPES" (Types for Proofs and Programs).

References

1. F. Barbanera and M. Fernandez. Combining first and higher order rewrite systems with type assignment systems. In M.Bezem and Groote [6], pages 60–74.
2. F. Barbanera, M. Fernandez, and H. Geuvers. Modularity of strong normalisation and confluence in the algebraic λ-cube. In *Proceedings of LICS'94*, pages 406–415. IEEE Press, 1994.
3. H.P. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Oxford Science Publications, 1992.
4. G. Barthe. Combining algebraic rewriting and induction in the calculus of constructions. Manuscript, 1995.

5. G. Barthe and H. Geuvers. Congruence types. To be presented at CSL'95, 1995.
6. M. Bezem and J.F. Groote, editors. *Proceedings of TLCA*, volume 664 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.
7. V. Breazu-Tannen. Combining algebra and higher-order types. In *Proceedings of LICS'88*, pages 82–90. IEEE Press, 1988.
8. V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic strong normalisation. *Theoretical Computer Science*, 83:3–28, 1990.
9. D. Dougherty. Adding algebraic rewriting to the untyped lambda calculus. *Information and Computation*, 101:251–267, 1992.
10. M. Fernandez. *Modèles de calcul multiparadigmes fondés sur la réécriture.* PhD thesis, Université Paris-Sud Orsay, 1993.
11. M. Fernandez and J-P.Jouannaud. Modularity of termination of term-rewriting systems revisited. In *Recent Trends in Data Type Specification*, volume 906 of *Lecture Notes in Computer Science*, pages 255–272, 1994.
12. J. Gallier. On Girard's "candidats de réducibilité. In P. Odifreddi, editor, *Logic and Computer Science*, pages 123–203. Academic Press, 1990.
13. H. Geuvers. *Logics and type systems.* PhD thesis, University of Nijmegen, 1993.
14. H. Geuvers. A short and flexible proof of strong normalisation for the calculus of constructions. In *Proceedings of TYPES'94*, Lecture Notes in Computer Science, 1995. To appear.
15. H. Geuvers and B. Werner. On the Church-Rosser property for expressive type systems and its consequence for their metatheoretic study. In *Proceedings of LICS'94*, pages 320–329. IEEE Press, 1994.
16. J-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur.* PhD thesis, Université Paris 7, 1972.
17. H.Geuvers and M-J. Nederhof. A modular proof of strong normalisation for the calculus of constructions. *Journal of Functional Programming*, 1:155–189, 1991.
18. J-P. Jouannaud and M. Okada. Executable higher-order algebraic specification languages. In *Proceedings of LICS'91*, pages 350–361. IEEE Press, 1991.
19. Z. Luo. *Computation and Reasoning: A Type Theory for Computer Science.* Number 11 in International Series of Monographs on Computer Science. Oxford University Press, 1994.
20. A. Middeldorp. *Modular properties of term rewriting systems.* PhD thesis, University of Amsterdam, 1990.
21. F. Müller. Confluence of the lambda calculus with left-linear algebraic rewriting. *Information Processing Letters*, 41:293–299, 1992.
22. C. Paulin-Mohring. Inductive definitions in the system Coq. Rules and properties. In Bezem and Groote [6], pages 328–345.
23. W. Tait. A realisability interpretation of the theory of species. In R. Parikh, editor, *Logic Colloquium 73*, volume 453 of *Lectures Notes in Mathematics*, pages 240–251, 1975.
24. J. Terlouw. Strong normalisation in type systems: a model-theoretical approach. In *Dirk van Dalen Festschrift*, pages 161–190. University of Utrecht, 1993. To appear in Annals of Pure and Applied Logic.
25. Y. Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1):128–143, 1987.
26. L. van Benthem Jutting, J. McKinna, and R. Pollack. Checking algorithms for pure type systems. In H. Barendregt and T. Nipkow, editors, *Proceedings of TYPES'93*, volume 806 of *Lecture Notes in Computer Science*, pages 19–61. Springer-Verlag, 1994.