Principal Typings for Explicit Substitutions Calculi*

Daniel Lima Ventura[1]**, Mauricio Ayala-Rincón[1]***, and Fairouz Kamareddine[2]

[1] Grupo de Teoria da Computação, Dep. de Matemática Universidade de Brasília, Brasília D.F., Brasil
[2] School of Mathematical and Computer Sciences Heriot-Watt University, Edinburgh, Scotland UK
{ventura,ayala}@mat.unb.br, fairouz@macs.hw.ac.uk

Abstract. Having principal typings (for short PT) is an important property of type systems. In simply typed systems, this property guarantees the possibility of a complete and terminating type inference mechanism. It is well-known that the simply typed λ-calculus has this property but recently J.B. Wells has introduced a system-independent definition of PT, which allows to prove that some type systems, e.g. the Hindley/Milner type system, do not satisfy PT. Explicit substitutions address a major computational drawback of the λ-calculus and allow the explicit treatment of the substitution operation to formally correspond to its implementation. Several extensions of the λ-calculus with explicit substitution have been given but some of which do not preserve basic properties such as the preservation of strong normalization. We consider two systems of explicit substitutions (λs_e and $\lambda\sigma$) and show that they can be accommodated with an adequate notion of PT. Specifically, our results are as follows:

- We introduce PT notions for the simply typed versions of the λs_e- and the $\lambda\sigma$-calculi and prove that they agree with Wells' notion of PT.
- We show that these versions satisfy PT by revisiting previously introduced type inference algorithms.

Key Words: lambda-calculus, explicit substitution, principal typings

1 Introduction

The development of well-behaved calculi of explicit substitutions is of great interest in order to bridge the formal study of the λ-calculus and its real implementations. Since β contraction depends on the definition of the operation

* Research supported by the CNPq Brazilian Research Council.
** Corresponding author, currently supported by a PhD scholarship of the CNPq at the Heriot-Watt University.
*** Author partially supported by the CNPq.

of substitution, which is informally given in the theory of λ-calculus, substitutions are in fact made explicit, but obscurely developed (that is, in an empirical manner), when most computational environments based on the λ-calculus are implemented. A remarkable exception is λProlog, for which its explicit substitutions calculus, the suspension calculus, has been extracted and formally studied [NaWi98].

In the study of making substitutions explicit, several alternatives rose out and all of them are directed to guarantee essential properties such as simulating beta-reduction, confluence, noetherianity (of the associated substitution calculus), subject reduction, having principal typings (for short PT), preservation of strong normalization etc. This is a non trivial task; for instance, the $\lambda\sigma$-calculus [ACCL91], that is one of the first proposed calculi of explicit substitutions, was reported to break the latter property some years after its introduction [Mel95]: this implies that infinite derivations starting from well-typed λ-terms are possible in this calculus, which is at least questionable for any mechanism supposed to simulate the λ-calculus explicitly. In this paper, the focus is on the PT property, which means that for any typable term M, there exists a type judgment $A \vdash M : \tau$, representing all possible typings $\langle A', \tau' \rangle$ for M. In the simply typed λ-calculus this corresponds to the existence of *more representative* typings. PT guarantees compositional type inference helping to find a complete/terminating type inference algorithm.

In section 2 we present the type-free λ-calculus in de Bruijn notation, the λs_e-calculus [KR97] and the $\lambda\sigma$-calculus [ACCL91]. In section 3 we present the type assignment systems background and then we present simply typed systems for each calculus we study. Then, we discuss the general notion of principal typings defined in [We2002] and present notions of principal typings for the λ-calculus in de Bruijn notation, the $\lambda\sigma$- and the λs_e-calculi and prove that they are adequate. In section 4 we conclude and present future work.

2 The type free calculi

2.1 λ-calculus in de Bruijn notation

Definition 1 (Set Λ_{dB}). *The syntax of λ-calculus in de Bruijn notation,* ***the*** λdB***-calculus****, is defined inductively as*

Terms $M ::= \underline{n} \,|\, (M\ M) \,|\, \lambda.M$ *where* $n \in \mathbb{N}^* = \mathbb{N} \setminus \{0\}$

Let M be a λ-term. If, in the tree representation of M, there are exactly n abstractors in the minimal path from the root position until the position of some subterm M_1, then M_1 is said **n-deep in** M. In other words, M_1 is in between n abstractors.

Definition 2. *We say that* $\underline{i}$ ***occurs as free index*** *in a term* M *if* $\underline{i+n}$ *is* n*-deep in* M*.*

Terms like $((((M_1\ M_2)\ M_3) \ldots)\ M_n)$ are written as usual $(M_1\, M_2\, \ldots\, M_n)$. The β-contraction definition in this notation needs a mechanism which detects and updates free indices of terms. It follows an operator similar to the one presented in [ARKa2001a].

Definition 3. *Let $M \in \Lambda_{dB}$ and $i \in \mathbb{N}$. The **i-lift** of M, denoted as M^{+i}, is defined inductively as*

1. $(M_1\, M_2)^{+i} = (M_1^{+i}\, M_2^{+i})$
2. $(\lambda.M_1)^{+i} = \lambda.M_1^{+(i+1)}$
3. $\underline{n}^{+i} = \begin{cases} \underline{n+1}\,, & \text{if } n > i \\ \underline{n}\,, & \text{if } n \leq i. \end{cases}$

The **lift** of a term M is its 0-lift, denoted as M^{+}. Intuitively, the lift of M corresponds to an increment by 1 of all free indices occurring in M. Using the i-lift, we are able to present the definition of the substitution used by β-contractions, similarly to the one presented in [ARKa2001a].

Definition 4. *Let $m, n \in \mathbb{N}^*$. The β-**substitution** for free occurrences of $\underline{n}$ in $M \in \Lambda_{dB}$ by term N, denoted as $\{\underline{n}\,/N\}M$, is defined inductively by*

1. $\{\underline{n}\,/N\}(M_1\ M_2) = (\{\underline{n}\,/N\}M_1\ \{\underline{n}\,/N\}M_2)$
2. $\{\underline{n}\,/N\}\lambda.M_1 = \lambda.\{\underline{n+1}\,/N^{+}\}M_1$
3. $\{\underline{n}\,/N\}\underline{m} = \begin{cases} \underline{m-1}\,, & \text{if } m > n \\ N, & \text{if } m = n \\ \underline{m}\,, & \text{if } m < n \end{cases}$

Observe that in item 2 of Def. 4, the lift operator is used to avoid captures of free indices in N. We present the β-contraction as defined in [ARKa2001a].

Definition 5. *β-**contraction** in λdB is defined by $(\lambda.M\, N) \rightarrow_{\beta} \{\underline{1}\,/N\}M$.*

Notice that item 3 in Definition 4, for $n = 1$, is the mechanism which does the substitution and updates the free indices in M as consequence of the lead abstractor elimination.

2.2 The λs_e-Calculus

The λs_e-calculus is a proper extension of the λdB-calculus. Two operators σ and φ are introduced for substitution and updating, respectively, to control the atomization of the substitution operation by arithmetic constraints.

Definition 6 (Set Λ_s of λs_e-terms). *The syntax of the λs_e-calculus, where $n, i, j \in \mathbb{N}^*$ and $k \in \mathbb{N}$ is given by*

Terms $M ::= \underline{n} \mid (M\ M) \mid \lambda.M \mid M\sigma^i M \mid \varphi^j_k M$

The term $M\sigma^i N$ represents the term $\{\underline{i}\,/N\}M$; i.e., the substitution of free occurrences of $\underline{i}$ in M by N, updating free variables in M (and in N). The term $\varphi^j_k M$ represents $j-1$ applications of the k-lift to the term M; i.e., $M^{+k^{(j-1)}}$. Table 1 contains the rewriting rules of the λs_e-calculus together with the rule (Eta), as given in [ARKa2001a].

$=_{s_e}$ denotes the equality for the associated substitution calculus, denoted as s_e, induced by all the rules except (σ-generation) and (Eta).

2.3 The $\lambda\sigma$-Calculus

The $\lambda\sigma$-calculus is given by a first-order rewriting system, which makes substitutions explicit by extending the language with two sorts of objects: **terms** and **substitutions** which are called $\lambda\sigma$-expression.

Definition 7 (Set Λ_σ of $\lambda\sigma$-expressions). *The $\lambda\sigma$-expressions consist of:*

Terms $M ::= \underline{1} \mid (M\ M) \mid \lambda.M \mid M[S]$

Substitutions $S ::= id \mid\ \uparrow\ \mid M.S \mid S \circ S$

$(\lambda.M\ N)$	$\longrightarrow$	$M\,\sigma^1 N$	(σ-generation)
$(\lambda.M)\sigma^i N$	$\longrightarrow$	$\lambda.(M\sigma^{i+1}N)$	(σ-λ-transition)
$(M_1\ M_2)\sigma^i N$	$\longrightarrow$	$((M_1\sigma^i N)\ (M_2\sigma^i N))$	(σ-app-trans.)
$\underline{n}\,\sigma^i N$	$\longrightarrow$	$\begin{cases} \underline{n-1} & \text{if } n > i \\ \varphi_0^i N & \text{if } n = i \\ \underline{n} & \text{if } n < i \end{cases}$	(σ-destruction)
$\varphi_k^i(\lambda.M)$	$\longrightarrow$	$\lambda.(\varphi_{k+1}^i M)$	(φ-λ-trans.)
$\varphi_k^i(M_1\ M_2)$	$\longrightarrow$	$((\varphi_k^i M_1)\ (\varphi_k^i M_2))$	(φ-app-trans.)
$\varphi_k^i\,\underline{n}$	$\longrightarrow$	$\begin{cases} \underline{n+i-1} & \text{if } n > k \\ \underline{n} & \text{if } n \le k \end{cases}$	(φ-destruction)
$(M_1\sigma^i M_2)\sigma^j N$	$\longrightarrow$	$(M_1\sigma^{j+1}N)\sigma^i(M_2\sigma^{j-i+1}N)$ if $i \le j$	(σ-σ-trans.)
$(\varphi_k^i M)\sigma^j N$	$\longrightarrow$	$\varphi_k^{i-1}M$ if $k < j < k+i$	(σ-φ-trans. 1)
$(\varphi_k^i M)\sigma^j N$	$\longrightarrow$	$\varphi_k^i(M\sigma^{j-i+1}N)$ if $k+i \le j$	(σ-φ-trans. 2)
$\varphi_k^i(M\sigma^j N)$	$\longrightarrow$	$(\varphi_{k+1}^i M)\sigma^j(\varphi_{k+1-j}^i N)$ if $j \le k+1$	(φ-σ-trans.)
$\varphi_k^i(\varphi_l^j M)$	$\longrightarrow$	$\varphi_l^j(\varphi_{k+1-j}^i M)$ if $l+j \le k$	(φ-φ-trans. 1)
$\varphi_k^i(\varphi_l^j M)$	$\longrightarrow$	$\varphi_l^{j+i-1}M$ if $l \le k < l+j$	(φ-φ-trans. 2)
$\lambda.(M\ \underline{1})$	$\longrightarrow$	N if $M =_{s_e} \varphi_0^2 N$	(*Eta*)

Table 1. The rewriting system of the λs_e-calculus with the Eta rule

Substitutions are lists of the form $N/\underline{i}$ indicating that the index $\underline{i}$ should be changed to the term N. The expression id represents a substitution of the form $\{\underline{1}/\underline{1}, \underline{2}/\underline{2}, \ldots\}$ whereas $\uparrow$ is the substitution $\{\underline{i+1}/\underline{i}\,|\,i \in \mathbb{N}^*\}$. The expression $S \circ S$ represents the composition of substitutions. Moreover, $\underline{1}[\uparrow^n]$, where $n \in \mathbb{N}^*$, codifies the de Bruijn index $\underline{n+1}$ and $\underline{i}[S]$ represents the value of $\underline{i}$ through the substitution S, which can be seen as a function $S(i)$. The substitution $M.S$ has the form $\{M/\underline{1}, S(i)/\underline{i+1}\}$, called the **cons of** M **in** S. $M[N.id]$ starts the simulation of the β-reduction of $(\lambda.M\ N)$ in $\lambda\sigma$. Thus, in addition to the substitution of the free occurrences of the index $\underline{1}$ by the corresponding term, free occurrences of indices should be decremented because of the elimination of the abstractor. Table 2 includes the rewriting system of the $\lambda\sigma$-calculus, as presented in [DoHaKi2000].

$(\lambda.M\ N)$	$\longrightarrow$	$M[N.id]$	(*Beta*)
$(M\ N)[S]$	$\longrightarrow$	$(M[S]\ N[S])$	(*App*)
$1[M.S]$	$\longrightarrow$	M	(*VarCons*)
$M[id]$	$\longrightarrow$	M	(*Id*)
$(\lambda.M)[S]$	$\longrightarrow$	$\lambda.(M[1.(S\circ\uparrow)])$	(*Abs*)
$(M[S])[T]$	$\longrightarrow$	$M[S \circ T]$	(*Clos*)
$id \circ S$	$\longrightarrow$	S	(*IdL*)
$\uparrow \circ (M.S)$	$\longrightarrow$	S	(*ShiftCons*)
$(S_1 \circ S_2) \circ S_3$	$\longrightarrow$	$S_1 \circ (S_2 \circ S_3)$	(*AssEnv*)
$(M.S) \circ T$	$\longrightarrow$	$M[T].(S \circ T)$	(*MapEnv*)
$S \circ id$	$\longrightarrow$	S	(*IdR*)
$1.\uparrow$	$\longrightarrow$	id	(*VarShift*)
$1[S].(\uparrow\circ S)$	$\longrightarrow$	S	(*Scons*)
$\lambda.(M\ \underline{1})$	$\longrightarrow$	N if $M =_\sigma N[\uparrow]$	(*Eta*)

Table 2. The rewriting system for the $\lambda\sigma$-calculus with the Eta rule

This system without (Eta) is equivalent to that of [ACCL91]. The associated substitution calculus, denoted as σ, is the one induced by all the rules except (Beta) and (Eta), and its equality is denoted as $=_\sigma$.

3 The Type Systems

Definition 8. *The syntax of the* **simple types** *and* **contexts** *is given by:*

Types $\tau ::= \alpha \,|\, \tau \to \tau$ ***Contexts*** $A ::= nil \,|\, \tau.A$

α ranges over **type variables**. A **type assignment system** $\mathcal{S}$ is a set of rules which allows some terms of a given system be associated with a type. A **context** gives the necessary information used by $\mathcal{S}$ rules to associate a type to a term. In the simply typed λ-calculus [Hi97], the typable terms are strongly normalizing. The ordered pair $\langle A, \tau\rangle$, of a context and a type, is called a **typing in** $\mathcal{S}$. For a term M, $A \vdash M : \tau$ denotes that M has type τ in context A, and $\langle A, \tau\rangle$ is called a **typing of** M. Let $\Theta = \langle A, \tau\rangle$ be a typing in $\mathcal{S}$. $\mathcal{S} \Vdash M : \Theta$ denotes that Θ is a typing of M in $\mathcal{S}$.

The contexts for λ-terms in de Bruijn notation are sequences of types. Let A be some context and $n \in \mathbb{N}$. Then $A_{<n}$ denotes the first $n-1$ types of A. Similarly we define $A_{>n}$, $A_{\leq n}$ and $A_{\geq n}$. Note that, for $A_{>n}$ and $A_{\geq n}$ the final nil element is included. For n=0, $A_{\leq 0}.A$=$A_{<0}.A$=A. The length of A is defined as $|nil|$=0 and, if A is not nil, $|A|$=1+$|A_{>1}|$. The addition of some type τ at the end of a context A is defined as $A.\tau$=$A_{\leq m}.\tau.nil$, where $|A|$=m.

Given a term M, an interesting question is whether it is typable in $\mathcal{S}$ or not. Note that, we are using a Curry-style/implicit typing, where in $\lambda.M$ we did not specify the type of the bound variable ($\underline{1}$). Such terms have many types, depending on the context. Another important question is whether given a term, its so-called most general typing can be found. An answer to this question, which represents any other answer, is called **principal typing**. Principal typing (which is context independent) is not to be confused with a principal type (which is context dependent). Let Θ be a typing in $\mathcal{S}$ and $\mathbf{Terms}_{\mathcal{S}}(\Theta)$=$\{M|\mathcal{S} \Vdash M{:}\Theta\}$. J.B. Wells introduced in [We2002] a system-independent definition of PT and proved that it generalizes previous system-specific definitions.

Definition 9 ([We2002]). *A typing Θ in system $\mathcal{S}$ is principal for some term M if $\mathcal{S} \Vdash M : \Theta$ and for any Θ' such that $\mathcal{S} \Vdash M : \Theta'$ we have that $\Theta \leq_{\mathcal{S}} \Theta'$, where $\Theta_1 \leq_{\mathcal{S}} \Theta_2 \iff Terms_{\mathcal{S}}(\Theta_1) \subseteq Terms_{\mathcal{S}}(\Theta_2)$.*

In simply typed systems the principal typing notion is tied to type substitution and weakening. **Weakening** allows one to add unnecessary information to contexts. **Type substitution** maps type variables to types. Given a type substitution s, the extension for functional types is straightforward as $s(\sigma{\to}\tau)$=$s(\sigma){\to}s(\tau)$ and the extension for sequential contexts as $s(nil)$=nil and $s(\tau.A)$=$s(\tau).s(A)$. The extension for typings is given by $s(\Theta)$=$\langle s(A), s(\tau)\rangle$.

3.1 Principal typings for the simply typed $\boldsymbol{\lambda}$-calculus in de Bruijn notation $\boldsymbol{TA_{\lambda dB}}$

Definition 10. ***(The System*** $TA_{\lambda dB}$***)*** *The* $TA_{\lambda dB}$ *typing rules are given by*

$$\text{(Var)} \quad \tau.A \vdash \underline{1} : \tau \qquad \text{(Varn)} \quad \frac{A \vdash \underline{n} : \tau}{\sigma.A \vdash \underline{n+1} : \tau}$$

$$\text{(Lambda)} \quad \frac{\sigma.A \vdash M : \tau}{A \vdash \lambda.M : \sigma \to \tau} \qquad \text{(App)} \quad \frac{A \vdash M : \sigma \to \tau \quad A \vdash N : \sigma}{A \vdash (M\ N) : \tau}$$

This system is similar to TA_λ([Hi97]). The rule (Varn) allows the construction of contexts as sequences.

Lemma 1. *Let* M *be a* λdB*-term. If* $A \vdash M : \tau$*, then* $A.\sigma \vdash M : \tau$*. Hence, the rule* $\frac{A \vdash M:\tau}{A.\sigma \vdash M:\tau}$($\lambda dB$*-weak) holds in the system* $TA_{\lambda dB}$*.*

Using (λdB-weak) and type substitution, we follow the definition of [We2002] for Hindley's Principal Typing to define principal typing for the λdB-calculus.

Definition 11. *A* ***principal typing*** *in* $TA_{\lambda dB}$ *of a term* M *is the typing* $\Theta = \langle A, \tau \rangle$ *such that*

1. $TA_{\lambda dB} \Vdash M : \Theta$
2. *If* $TA_{\lambda dB} \Vdash M : \Theta'$ *for any typing* $\Theta' = \langle A', \tau' \rangle$*, then there exists some substitution* s *such that* $s(A) = A'_{\leq |A|}.nil$ *and* $s(\tau) = \tau'$*.*

Observe that, given a principal typing $\langle A, \tau \rangle$ of M, the context A is the shortest context where M can be typable. In contrast to the λ-calculus with names, where the context from a principal typing of M is the smallest set because it declares types for exactly the free variables of M, the context from a principal typing in λdB may has some type declaration for variables not occurring in the term, to maintain the ordered structure of contexts. For example, a PT for $\underline{2}$ is $\langle \tau_1.\tau_2.nil, \tau_2 \rangle$.

As is the case for the simply typed λ-calculus with names, the best way to assure that Definition 11 is the correct translation of the PT concept, is to verify that Definition 11 corresponds to Definition 9.

Theorem 1. *A typing* Θ *is principal in* $TA_{\lambda dB}$ *according to Definition 11 iff* Θ *is principal in* $TA_{\lambda dB}$ *according to Definition 9.*

We now present a type inference algorithm for λdB-terms, similarly to the one in [AyMu2000] for λs_e, to verify whether $TA_{\lambda dB}$ has PT according to Definition 11. Given any term M, decorate each subterm with a new type variable as subscript and a new context variable as superscript, obtaining a new term denoted as M'. For example, for term $\lambda.(\underline{2}\ \underline{1})$ we have the decorated term $(\lambda.(\underline{2}\,^{A_1}_{\tau_1}\ \underline{1}\,^{A_2}_{\tau_2})^{A_3}_{\tau_3})^{A_4}_{\tau_4}$. Then, rules from Table 3 are applied to pairs of the form $\langle\langle R, E \rangle\rangle$, where R is a set of decorated terms and E a set of equations on type and context variables.

The inference rules in Table 3 are given according to the typing rules of $TA_{\lambda dB}$. Type inference for M starts with $\langle\langle R_0, \varnothing \rangle\rangle$, where R_0 is the set of all M' subterms. The rules from Table 3 are applied until one reaches $\langle\langle \varnothing, E_f \rangle\rangle$, where E_f is a set of first-order equations over context and type variables.

(Var)	$\langle\langle R \cup \{\underline{1}^{A}_{\tau}\}, E\rangle\rangle$	$\to \langle\langle R, E \cup \{A = \tau.A'\}\rangle\rangle$,where A' is a fresh context variable;
(Varn)	$\langle\langle R \cup \{\underline{n}^{A}_{\tau}\}, E\rangle\rangle$	$\to \langle\langle R, E \cup \{A = \tau'_1.\cdots.\tau'_{n-1}.\tau.A'\}\rangle\rangle$,where A' and $\tau'_1, \ldots, \tau'_{n-1}$ are fresh context and type variables;
(Lambda)	$\langle\langle R \cup \{(\lambda.M^{A_1}_{\tau_1})^{A_2}_{\tau_2}\}, E\rangle\rangle$	$\to \langle\langle R, E \cup \{\tau_2 = \tau^* \to \tau_1, A_1 = \tau^*.A_2\}\rangle\rangle$, where τ^* is a fresh type variable;
(App)	$\langle\langle R \cup \{(M^{A_1}_{\tau_1}\ N^{A_2}_{\tau_2})^{A_3}_{\tau_3}\}, E\rangle\rangle$	$\to \langle\langle R, E \cup \{A_1 = A_2, A_2 = A_3, \tau_1 = \tau_2 \to \tau_3\}\rangle\rangle$

Table 3. Rules for Type Inference in System $TA_{\lambda dB}$

Example 1. Let $M = \lambda.(\underline{2}\ \ \underline{1})$. Then $M' = (\lambda.(\underline{2}\,^{A_1}_{\tau_1}\ \ \underline{1}\,^{A_2}_{\tau_2})^{A_3}_{\tau_3})^{A_4}_{\tau_4}$ and $R_0 = \{\underline{2}\,^{A_1}_{\tau_1}, \underline{1}\,^{A_2}_{\tau_2}, (\underline{2}\,^{A_1}_{\tau_1}\ \ \underline{1}\,^{A_2}_{\tau_2})^{A_3}_{\tau_3}, (\lambda.(\underline{2}\,^{A_1}_{\tau_1}\ \ \underline{1}\,^{A_2}_{\tau_2})^{A_3}_{\tau_3})^{A_4}_{\tau_4}\}$. Using the rules in Table 3 we have the following reduction:

$\langle\langle R_0, \emptyset\rangle\rangle \to_{\text{Varn}}$

$\langle\langle R_1 = R_0 \setminus \{\underline{2}\,^{A_1}_{\tau_1}\}, E_1 = \{A_1 = \tau'_1.\tau_1.A'_1\}\rangle\rangle \to_{\text{Var}}$

$\langle\langle R_2 = R_1 \setminus \{\underline{1}\,^{A_2}_{\tau_2}\}, E_2 = E_1 \cup \{A_2 = \tau_2.A'_2\}\rangle\rangle \to_{\text{App}}$

$\langle\langle R_3 = R_2 \setminus \{(\underline{2}\,^{A_1}_{\tau_1}\ \ \underline{1}\,^{A_2}_{\tau_2})^{A_3}_{\tau_3}\}, E_3 = E_2 \cup \{A_1 = A_2, A_2 = A_3, \tau_1 = \tau_2 \to \tau_3\}\rangle\rangle \to_{\text{Lambda}}$

$\langle\langle \emptyset = R_3 \setminus \{(\lambda.(\underline{2}\,^{A_1}_{\tau_1}\ \ \underline{1}\,^{A_2}_{\tau_2})^{A_3}_{\tau_3})^{A_4}_{\tau_4}\}, E_4 = E_3 \cup \{\tau_4 = \tau^*_1 \to \tau_3, A_3 = \tau^*_1.A_4\}\rangle\rangle$

Thus, $E_4 = E_f$. Solving the trivial equation over context variables, i.e. $A_1 = A_2 = A_3$, and using variables of smaller subscripts, one gets $\{\tau_1 = \tau_2 \to \tau_3, \tau_4 = \tau^*_1 \to \tau_3, A_1 = \tau'_1.\tau_1.A'_1, A_1 = \tau_2.A'_2, A_1 = \tau^*_1.A_4\}$. Thus, simplifying one gets $\{\tau_1 = \tau_2 \to \tau_3, \tau_4 = \tau^*_1 \to \tau_3, \tau'_1.\tau_1.A'_1 = \tau_2.A'_2 = \tau^*_1.A_4\}$. From these equations one gets the most general unifier (mgu for short) $\tau_4 = \tau_2 \to \tau_3$ and $A_4 = (\tau_2 \to \tau_3).A'_1$, for the variables of interest. Since the context must be the shortest one, $A'_1 = nil$ and $\langle(\tau_2 \to \tau_3).nil, \tau_2 \to \tau_3\rangle$ is a principal typing of M.

From Definition 11 and by the uniqueness of the solutions of the type inference algorithm, one deduces that $TA_{\lambda dB}$ satisfies PT. The next theorem says that every typable term has a principal typing.

Theorem 2 (Principal Typings for $TA_{\lambda dB}$). *$TA_{\lambda dB}$ satisfies the property of having principal typings.*

3.2 Principal typings for $TA_{\lambda s_e}$, the simply typed λs_e

The typed version of λs_e presented is in Curry style, which we have verified to have the same properties which properties as the version in Church style presented in [ARKa2001a]. In particular, the properties in question being: weak normalisation (WN), confluence (CR) and subject reduction (SR). Thus, the syntax of λs_e-terms and the rules are the same as the untyped version.

Since the syntax of λs_e remains close to the λdB-calculus, to have a type assignment system for the λs_e-calculus we only need to add typing rules to $TA_{\lambda dB}$ for the two new kinds of terms.

Definition 12 (The System $TA_{\lambda s_e}$). *$TA_{\lambda s_e}$ is given by (Var), (Varn), (App), (Lambda) from Definiton 10 and the following new rules.*

$$\textit{(Sigma)}\ \frac{A_{\geq i} \vdash N : \rho \quad A_{<i}.\rho.A_{\geq i} \vdash M : \tau}{A \vdash M\sigma^i N : \tau} \qquad \textit{(Phi)}\ \frac{A_{\leq k}.A_{\geq k+i} \vdash M : \tau}{A \vdash \varphi^i_k M : \tau}$$

Weakening for λs_e is done the same way as for λdB, adding types at the end of a context, giving the following lemma.

Lemma 2 (Weakening for λs_e). *The rule (λs_e-weak) holds in System $TA_{\lambda s_e}$, where* $\dfrac{A \vdash M : \tau}{A.\sigma \vdash M : \tau}$ *(λs_e-weak).*

Consequently, the definition of principal typings in λs_e is the same as that for $TA_{\lambda dB}$. For the sake of completeness we repeat it here.

Definition 13 (Principal Typings in $TA_{\lambda s_e}$). *A **principal typing** of a term M in $TA_{\lambda s_e}$ is a typing $\Theta = \langle A, \tau \rangle$ such that*

1. $TA_{\lambda s_e} \Vdash M : \Theta$
2. *If $TA_{\lambda s_e} \Vdash M : \Theta'$ for any typing $\Theta' = \langle A', \tau' \rangle$, then there exists a substitution s such that $s(A) = A'_{\leq |A|}.nil$ and $s(\tau) = \tau'$.*

Theorem 3. *A typing Θ is principal in $TA_{\lambda s_e}$ according to Definition 13 iff Θ is principal in $TA_{\lambda s_e}$ according to Definition 9.*

We now present a type inference algorithm for the λs_e-calculus, similarly to that of [AyMu2000]. The algorithm is composed of the rules from Table 3 and the new rules in Table 4.

(Sigma)	$\langle\!\langle R \cup \{(M_{\tau_1}^{A_1} \sigma^i N_{\tau_2}^{A_2})_{\tau_3}^{A_3}\}, E \rangle\!\rangle \to$ $\langle\!\langle R, E \cup \{\tau_1 = \tau_3, A_1 = \tau'_1. \cdots .\tau'_{i-1}.\tau_2.A_2, A_3 = \tau'_1. \cdots .\tau'_{i-1}.A_2\}\rangle\!\rangle$, where $\tau'_1, \ldots, \tau'_{i-1}$ are new type variables and the sequence is empty if $i = 1$;
(Phi)	$\langle\!\langle R \cup \{(\varphi_k^i M_{\tau_1}^{A_1})_{\tau_2}^{A_2}\}, E \rangle\!\rangle \to$ $\langle\!\langle R, E \cup \{\tau_1 = \tau_2, A_2 = \tau'_1. \cdots .\tau'_{k+i-1}.A', A_1 = \tau'_1. \cdots .\tau'_k.A'\}\rangle\!\rangle$, where A' and $\tau'_1, \ldots, \tau'_{k+i-1}$ are new variables of context and type and if $k + i - 1, k = 0$ then the sequences $\tau'_1, \ldots, \tau'_{k+i-1}$ and $\tau'_1, \ldots, \tau'_k$, respectively, are empty.

Table 4. Type inference rules for the λs_e-Calculus

Similarly to the previous algorithm, the rules of the Table 4 were developed according to the rules of Definition 12. The decorated term associated with M, denoted as M', has a syntax close to the one of decorated λdB-terms: any subterm is decorated with its type and its context variables. The rules are applied to pairs $\langle\!\langle R, E \rangle\!\rangle$, starting from the pair $\langle\!\langle R_0, \varnothing \rangle\!\rangle$, as was done to $TA_{\lambda dB}$.

Example 2. For the λs_e-term $M = \lambda.((\underline{1}\,\sigma^2 \underline{2}\,)\ (\varphi_0^2\,\underline{2}\,))$, one obtains the corresponding R_0 from $M' = (\lambda.((\underline{1}_{\tau_1}^{A_1} \sigma^2 \underline{2}_{\tau_2}^{A_2}\,)_{\tau_3}^{A_3}\ (\varphi_0^2\,\underline{2}_{\tau_4}^{A_4}\,)_{\tau_5}^{A_5})_{\tau_6}^{A_6})_{\tau_7}^{A_7}$. Then, applying the rules in Table 3 and 4 to the pair $\langle\!\langle R_0, \emptyset \rangle\!\rangle$, obtaining the pair $\langle\!\langle \emptyset, E_f \rangle\!\rangle$, and simplifying E_f, similarly to the example 1, one obtains the system of equations $\{\,\tau_1 = \tau_4 \to \tau_6\ ,\ \tau_7 = \tau_1^* \to \tau_6\ ,\ \tau_1.A'_1 = \tau'_2.\tau_2.A_2\ ,\ \tau'_2.A_2 = \tau'_4.\tau'_3.\tau_4.A'_3 = \tau_1^*.A_7\ ,\ A_2 = \tau'_1.\tau_2.A'_2\,\}$ from which one has the mgu $\tau_7 = (\tau_2 \to \tau_6) \to \tau_6$ and $A_7 = \tau'_1.\tau_2.A'_2$ for variables of interest.

Theorem 4 (Principal Typings for $TA_{\lambda s_e}$). *$TA_{\lambda s_e}$ satisfies the property of having principal typings.*

3.3 Principal typings for $TA_{\lambda\sigma}$, the simply typed $\lambda\sigma$

The typing rules of the $\lambda\sigma$-calculus provide types for objects of sort term as well as for objects of sort substitution. An object of sort substitution, due to its semantics, can be viewed as a list of terms. Consequently, its type is a context. $S \triangleright A$ denotes that the object of sort substitution S has type A.

Definition 14 (The System $TA_{\lambda\sigma}$). *$TA_{\lambda\sigma}$ is given by the following typing rules.*

(var)	$\tau.A \vdash \underline{1} : \tau$	*(lambda)*	$\dfrac{\sigma.A \vdash M : \tau}{A \vdash \lambda.M : \sigma \to \tau}$
(app)	$\dfrac{A \vdash M : \sigma \to \tau \quad A \vdash N : \sigma}{A \vdash (M\ N) : \tau}$	*(clos)*	$\dfrac{A \vdash S \triangleright A' \qquad A' \vdash M : \tau}{A \vdash M[S] : \tau}$
(id)	$A \vdash id \triangleright A$	*(shift)*	$\tau.A \vdash \uparrow \triangleright A$
(cons)	$\dfrac{A \vdash M : \tau \qquad A \vdash S \triangleright A'}{A \vdash M.S \triangleright \tau.A'}$	*(comp)*	$\dfrac{A \vdash S \triangleright A'' \quad A'' \vdash S' \triangleright A'}{A \vdash S' \circ S \triangleright A'}$

Observe that the name of the typing rules begin with lower-case letters, while the rewriting rules with upper-case letters. As for λs_e, the typed version of the $\lambda\sigma$-calculus is presented in Curry style. We have verified that the Curry style version has WN, CR and SR as the Church style version of [DoHaKi2000].

For $TA_{\lambda\sigma}$ the notion of typing has to be adapted since the $\lambda\sigma$-expression of sort substitution is decorated with contexts variables as types and as contexts. Thus, one may say that $\Theta = \langle A, \mathbb{T}\rangle$ is a typing of a $\lambda\sigma$-expression in $TA_{\lambda\sigma}$, where $\mathbb{T}$ can be either a type or a context. If the analysed expression belongs to the λ-calculus, the notion of typing corresponds to that of $TA_{\lambda dB}$.

Lemma 3 (Weakening for $\lambda\sigma$). *Let M be a $\lambda\sigma$-term and S a $\lambda\sigma$-substitution. If $A \vdash M : \tau$, then $A.\sigma \vdash M : \tau$, for any type σ. Similarly, if $A \vdash S \triangleright A'$, then $A.\sigma \vdash S \triangleright A'.\sigma$. Hence, the rules ($\lambda\sigma$-tweak) and ($\lambda\sigma$-sweak) hold in System $TA_{\lambda\sigma}$, where*

$$\frac{A \vdash M : \tau}{A.\sigma \vdash M : \tau}(\lambda\sigma\text{-}tweak) \qquad\qquad \frac{A \vdash S \triangleright A'}{A.\sigma \vdash S \triangleright A'.\sigma}(\lambda\sigma\text{-}sweak)$$

Lemma 3 and type substitution allow us present a definition for PT in $TA_{\lambda\sigma}$.

Definition 15 (Principal Typings in $TA_{\lambda\sigma}$). *A **principal typing** of an expression M in $TA_{\lambda\sigma}$ is a typing $\Theta = \langle A, \mathbb{T}\rangle$ such that*
1. *$TA_{\lambda\sigma} \Vdash M : \Theta$*
2. *If $TA_{\lambda\sigma} \Vdash M : \Theta'$ for any typing $\Theta' = \langle A', \mathbb{T}'\rangle$, then there exists a substitution s such that $s(A) = A'_{\leq|A|}.nil$ and if $\mathbb{T}$ is a type, $s(\mathbb{T}) = \mathbb{T}'$, otherwise we have that $s(\mathbb{T}) = \mathbb{T}'_{\leq|\mathbb{T}|}.nil$.*

We might verify if this PT definition has a correspondence with Wells' system-independent definition [We2002].

Theorem 5. *A typing Θ is principal in $TA_{\lambda\sigma}$ according to Definition 15 iff Θ is principal in $TA_{\lambda\sigma}$ according to Definition 9.*

We now present an algorithm for type inference, to verify if $TA_{\lambda\sigma}$ has PT according to Definition 15. Thus, given an expression M, we will work with the decorated expression M' but the type for substitutions is a context as well. We use the same syntax for decorated expressions as in [Bo95].

(Var)	$\langle\!\langle R \cup \{\underline{1}_{\tau}^{A}\}, E\rangle\!\rangle$	$\rightarrow \langle\!\langle R, E \cup \{A = \tau.A'\}\rangle\!\rangle$,where A' is a fresh context variable;
(Lambda)	$\langle\!\langle R \cup \{(\lambda.M_{\tau_1}^{A_1})_{\tau_2}^{A_2}\}, E\rangle\!\rangle$	$\rightarrow \langle\!\langle R, E \cup \{\tau_2 = \tau^* \rightarrow \tau_1, A_1 = \tau^*.A_2\}\rangle\!\rangle$, where τ^* is a fresh type variable;
(App)	$\langle\!\langle R \cup \{(M_{\tau_1}^{A_1}\ N_{\tau_2}^{A_2})_{\tau_3}^{A_3}\}, E\rangle\!\rangle$	$\rightarrow \langle\!\langle R, E \cup \{A_1 = A_2, A_2 = A_3, \tau_1 = \tau_2 \rightarrow \tau_3\}\rangle\!\rangle$
(Clos)	$\langle\!\langle R \cup \{(M_{\tau_1}^{A_1}[S_{A_3}^{A_2}])_{\tau_2}^{A_4}\}, E\rangle\!\rangle$	$\rightarrow \langle\!\langle R, E \cup \{A_1 = A_3, A_2 = A_4, \tau_1 = \tau_2\}\rangle\!\rangle$
(Id)	$\langle\!\langle R \cup \{id_{A_2}^{A_1}\}, E\rangle\!\rangle$	$\rightarrow \langle\!\langle R, E \cup \{A_1 = A_2\}\rangle\!\rangle$
(Shift)	$\langle\!\langle R \cup \{\uparrow_{A_2}^{A_1}\}, E\rangle\!\rangle$	$\rightarrow \langle\!\langle R, E \cup \{A_1 = \tau'.A_2\}\rangle\!\rangle$,where τ' is a fresh type variable;
(Cons)	$\langle\!\langle R \cup \{(M_{\tau_1}^{A_1}.S_{A_3}^{A_2})_{A_5}^{A_4}\}, E\rangle\!\rangle$	$\rightarrow \langle\!\langle R, E \cup \{A_1 = A_2, A_2 = A_4, A_5 = \tau_1.A_3\}\rangle\!\rangle$
(Comp)	$\langle\!\langle R \cup \{(S_{A_2}^{A_1} \circ T_{A_4}^{A_3})_{A_6}^{A_5}\}, E\rangle\!\rangle$	$\rightarrow \langle\!\langle R, E \cup \{A_1 = A_4, A_2 = A_6, A_3 = A_5\}\rangle\!\rangle$

Table 5. Type inference rules for the $\lambda\sigma$-calculus

The inference rules presented in Table 5 are given according to the typing rules of the system $TA_{\lambda\sigma}$ presented in Definition 14. Similarly to the previous algorithm, the rules are applied to pairs $\langle\!\langle R, E\rangle\!\rangle$, where R is a set of subexpressions of M' and E a set of equations over type and context variables.

Example 3. For $M = (\underline{2}.id) \circ \uparrow$ one has $M' = (((\underline{1}_{\tau_1}^{A_1}[\uparrow_{A_3}^{A_2}])_{\tau_2}^{A_4}.id_{A_6}^{A_5})_{A_8}^{A_7} \circ \uparrow_{A_{10}}^{A_9})_{A_{12}}^{A_{11}}$. Then $R_0 = \{(\underline{1}_{\tau_1}^{A_1}[\uparrow_{A_3}^{A_2}])_{\tau_2}^{A_4}, ((\underline{1}_{\tau_1}^{A_1}[\uparrow_{A_3}^{A_2}])_{\tau_2}^{A_4}.id_{A_6}^{A_5})_{A_8}^{A_7}, (((\underline{1}_{\tau_1}^{A_1}[\uparrow_{A_3}^{A_2}])_{\tau_2}^{A_4}.id_{A_6}^{A_5})_{A_8}^{A_7} \circ \uparrow_{A_{10}}^{A_9})_{A_{12}}^{A_{11}}, \underline{1}_{\tau_1}^{A_1}, \uparrow_{A_3}^{A_2}, id_{A_6}^{A_5}, \uparrow_{A_{10}}^{A_9}\}$. Applying the rules from Table 5 to the pair $\langle\!\langle R_0, \emptyset\rangle\!\rangle$ until obtain the pair $\langle\!\langle \emptyset, E_f\rangle\!\rangle$ and simplifying E_f, as in example 1, one obtains the set of equations $\{\tau_1 = \tau_2, A_{11} = A_{12} = \tau_2.A_2, A_2 = \tau_1'.A_1, A_1 = \tau_1.A_1'\}$. From this equational system one obtains the mgu $A_{11}{=}A_{12}{=}\tau_1.\tau_1'.\tau_1.A_1'$, for the variables of interest. Thus, $\langle \tau_1.\tau_1'.\tau_1.nil, \tau_1.\tau_1'.\tau_1.nil\rangle$ is a principal typing of M.

Theorem 6 (Principal Typings for $TA_{\lambda\sigma}$). *$TA_{\lambda\sigma}$ satisfies the property of having principal typings.*

4 Conclusions and Future Work

We considered for λs_e and $\lambda\sigma$ particular notions of principal typings and gave respective definitions which we proved to agree with the system-independent notion of Wells in [We2002]. The adaptation of this general notion of principal typings for the $\lambda\sigma$ requires special attention, since this calculus enlarges the language of the λ-calculus by introducing a new sort of *substitution* objects, whose types are contexts. Thus, the provided PT notion has to deal with the principality of substitution objects as well. Then, the property of having principal typings is straightforwardly proved by revisiting type inference algorithms for the λs_e and the $\lambda\sigma$, previously presented in [AyMu2000] and [Bo95], respectively. The result is based on the correctness, completeness and uniqueness of solutions given by adequate first-order unification algorithms (e.g. see the unification algorithm given in [Hi97]).

The investigation of this property for more elaborated typing systems of explicit substitutions is an interesting work to be done.

References

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

[ARMoKa2005] M. Ayala-Rincón, F. de Moura, and F. Kamareddine. Comparing and Implementing Calculi of Explicit Substitutions with Eta-Reduction. *Annals of Pure and Applied Logic*, 134:5–41, 2005.

[ARKa2001a] M. Ayala-Rincón and F. Kamareddine. Unification via the λs_e-Style of Explicit Substitution. *The Logical Journal of the Interest Group in Pure and Applied Logics*, 9(4):489–523, 2001.

[AyMu2000] M. Ayala-Rincón and C. Muñoz. Explicit Substitutions and All That. *Revista Colombiana de Computación*, 1(1):47–71, 2000.

[Bo95] P. Borovanský. Implementation of Higher-Order Unification Based on Calculus of Explicit Substitutions. In M. Bartošek, J. Staudek, and J. Wiedermann, editors, *Proceedings of the SOFSEM'95: Theory and Practice of Informatics*, volume 1012 of *LNCS*, pages 363–368. Springer Verlag, 1995.

[deBru72] N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972.

[DoHaKi2000] G. Dowek, T. Hardin, and C. Kirchner. Higher-order Unification via Explicit Substitutions. *Information and Computation*, 157(1/2):183–235, 2000.

[Hi97] J. R. Hindley. *Basic Simple Type Theory.* Number 42 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1997.

[KR97] F. Kamareddine and A. Ríos. Extending a λ-calculus with Explicit Substitution which Preserves Strong Normalisation into a Confluent Calculus on Open Terms. *J. of Func. Programming*, 7:395–420, 1997.

[Mel95] P.-A. Melliès. Typed λ-calculi with explicit substitutions may not terminate. In Proc. of TLCA'95, volume 902 of *LNCS*, pages 328–334. Springer Verlag, 1995.

[NaWi98] G. Nadathur and D. S. Wilson. A Notation for Lambda Terms A Generalization of Environments. *Theoretical Computer Science*, 198:49–98, 1998.

[We2002] J. Wells. The essence of principal typings. In *Proc. 29th International Colloquium on Automata, Languages and Programming, ICALP 2002*, volume 2380 of *LNCS*, pages 913–925. Springer Verlag, 2002.

A Proofs

The proofs are divided in three parts: A.1, where the proof of weakening for each type system is given; A.2, where the three proofs of the correspondence between system-independent and system-specific definition of PT are merged in one; and A.3, where the three proofs of PT are also merged.

A.1 Proofs of weakening

Proof (Lemma 1 (Weakening for $TA_{\lambda dB}$)).

Let $A \vdash M : \tau$. We will prove a more general result stating, for $i \in \mathbb{N}$, that $A_{\leq i}.\sigma.A_{>i} \vdash M^{+i} : \tau$. The proof is done by induction on the structure of M. Note that if $i \geq m$, where $m = |A|$, then σ is added at the end of A.

1) $M = \underline{n}$: Suppose $A \vdash \underline{n} : \tau$. If $n \leq i$, then $\underline{n}^{+i} = \underline{n}$. The σ addition at the i+1-th position changes only types of indices $\geq \underline{i+1}$, thus one has trivially that $A_{\leq i}.\sigma.A_{>i} \vdash \underline{n} : \tau$. If $n > i$, then $\underline{n}^{+i} = \underline{n+1}$. By $(Varn)$ i times one has $A_{>i} \vdash \underline{n-i} : \tau$. Thus, by $(Varn)$ applied $i+1$ times, one has that $A_{\leq i}.\sigma.A_{>i} \vdash \underline{n+1} : \tau$.
2) $M = (M_1\, M_2)$: Suppose $A \vdash (M_1\, M_2) : \tau$. By (App), $A \vdash M_1 : \rho \to \tau$ and $A \vdash M_2 : \rho$. By induction hypothesis (IH), $A_{\leq i}.\sigma.A_{>i} \vdash M_1^{+i} : \rho \to \tau$ and $A_{\leq i}.\sigma.A_{>i} \vdash M_2^{+i} : \rho$. Thus, by (App), $A_{\leq i}.\sigma.A_{>i} \vdash (M_1^{+i}\, M_2^{+i}) : \tau$.
3) $M = \lambda.N$: Suppose $A \vdash \lambda.N : \tau$. By $(Lambda)$ one has that $\rho.A \vdash N : \mu$, where $\tau = \rho \to \mu$. By IH one has $\rho.A_{\leq i}.\sigma.A_{>i} \vdash N^{+(i+1)} : \mu$. Thus, by $(Lambda)$, $A_{\leq i}.\sigma.A_{>i} \vdash \lambda.N^{+(i+1)} : \rho \to \mu = \tau$.

Since all the information about M free indices is in context A, one has that a maximum value for a free index occurrence, at 0-deep in M, is $m = |A|$. Consequently, $M^{+j} = M$ for any $j \geq m$. Thus, for $i = m$, we have that $A.\sigma \vdash M : \tau$, for any type σ. Then a weak rule for $TA_{\lambda dB}$ is admissible, adding types at the end of the context. A type addition in any other position of context A would require updating some free indices, then M^{+i} would correspond to a different function from the one to which the term M corresponds. □

Proof (Lemma 2 (Weakening for $TA_{\lambda s_e}$)). Induction on the structure of M.

1) $M = \underline{n}$: Let $A \vdash \underline{n} : \tau$. Since the type addition at the end of A does not change any free index type, one has trivially that $A.\sigma \vdash \underline{n} : \tau$.
2) $M = (M_1\, M_2)$: Let $A \vdash (M_1\, M_2) : \tau$. By (App), $A \vdash M_1 : \rho \to \tau$ and $A \vdash M_2 : \rho$, for some ρ. By IH, $A.\sigma \vdash M_1 : \rho \to \tau$ and $A.\sigma \vdash M_2 : \rho$. Thus, by (App), $A.\sigma \vdash (M_1\, M_2) : \tau$.
3) $M = \lambda.N$: Let $A \vdash \lambda.N : \tau$. By $(Lambda)$, $\rho.A \vdash N : \mu$, where $\tau = \rho \to \mu$. By IH, $\rho.A.\sigma \vdash N : \mu$. Thus, by $(Lambda)$, $A.\sigma \vdash \lambda.N : \tau$.
4) $M = M_1 \sigma^i M_2$: Let $A \vdash M_1 \sigma^i M_2 : \tau$. By $(Sigma)$, $A_{\geq i} \vdash M_2 : \rho$ and $A_{<i}.\rho.A_{\geq i} \vdash M_1 : \tau$. By IH, $A_{\geq i}.\sigma \vdash M_2 : \rho$ and $A_{<i}.\rho.A_{\geq i}.\sigma \vdash M_1 : \tau$. Thus, by $(Sigma)$, $A.\sigma \vdash M_1 \sigma^i M_2 : \tau$.
5) $M = \varphi^i_k N$: Let $A \vdash \varphi^i_k N : \tau$. By (Phi), $A_{\leq k}.A_{\geq k+i} \vdash N : \tau$. By IH, $A_{\leq k}.A_{\geq k+i}.\sigma \vdash N : \tau$. Thus, by (Phi), $A.\sigma \vdash \varphi^i_k N : \tau$. □

The proof of Lemma 3 needs some auxiliary definitions and lemmas.

Definition 16. *Let M be a $\lambda\sigma$-expression. Define $\|\cdot\| : \Lambda_\sigma \to \mathbb{N}$ as*

$$\begin{aligned} \|(M\ N)\| &= \|M\| + \|N\| & \|\underline{1}\,\| &= 0 \\ \|\lambda.M\| &= \|M\| & \|id\| &= 0 \\ \|M[S]\| &= \|M\| + \|S\| & \|\uparrow\| &= 0 \\ \|S \circ T\| &= \|S\| + \|T\| & \|M.S\| &= 1 + \|M\| + \|S\| \end{aligned}$$

Lemma 4. *In $\lambda\sigma$, if $\|S\| = 0$ and $A \vdash S \rhd A'$, then $A.\sigma \vdash S \rhd A'.\sigma$.*

Proof. By induction on the structure of S where $\|S\| = 0$.

1) $S = id$: By (id) one has $A.\sigma \vdash id \rhd A'.\sigma$, trivially.
2) $S = \uparrow$: Let $A \vdash \uparrow \rhd A'$ where, by (shift), $A = \tau.A'$. Thus $A.\sigma \vdash \uparrow \rhd A'.\sigma$.
3) $S = S_1 \circ S_2$: Let $A \vdash S_1 \circ S_2 \rhd A'$. By (comp), one has that $A \vdash S_2 \rhd A''$ and $A'' \vdash S_1 \rhd A'$, for some A''. By IH one has $A.\sigma \vdash S_2 \rhd A''.\sigma$ and $A''.\sigma \vdash S_1 \rhd A'.\tau$. Thus, by (comp), $A.\sigma \vdash S_1 \circ S_2 \rhd A'.\sigma$.

□

Lemma 5. *In $\lambda\sigma$, if $\|M\| = 0$ and $A \vdash M : \tau$, then $A.\sigma \vdash M : \tau$.*

Proof. By induction on on the structure of M where $\|M\| = 0$.

1) $M = \underline{1}$: Let $A \vdash \underline{1} : \tau$. By (var) one has that $A = \tau.A'$, for some A'. Thus one has $A.\sigma \vdash \underline{1} : \tau$, trivially.
2) $M = (M_1\ M_2)$: Let $A \vdash (M_1\ M_2) : \tau$. By (app), $A \vdash M_1 : \rho \to \tau$ and $A \vdash M_2 : \rho$, for some ρ. By IH, $A.\sigma \vdash M_1 : \rho \to \tau$ and $A.\sigma \vdash M_2 : \rho$. Thus, by (app), $A.\sigma \vdash (M_1\ M_2) : \tau$.
3) $M = \lambda.N$: Let $A \vdash \lambda.N : \tau$. By (lambda), $\rho.A \vdash N : \mu$, where $\tau = \rho \to \mu$. By IH, $\rho.A.\sigma \vdash N : \mu$. Thus, by (lambda), $A.\sigma \vdash \lambda.N : \tau$.
4) $M = N[S]$: Let $A \vdash N[S] : \tau$. By (clos), $A \vdash S \rhd A'$ and $A' \vdash N : \tau$, for some A'. Since $\|N[S]\| = \|N\| + \|S\| = 0$, by Lemma 4, $A.\sigma \vdash S \rhd A'.\sigma$. By IH, $A'.\sigma \vdash N : \tau$. Thus, by (clos), $A.\sigma \vdash N[S] : \tau$.

□

Proof (Lemma 3 (Weakening for $TA_{\lambda\sigma}$)). By induction on the structure of M with subinduction on $\|\cdot\|$, having Lemmas 4 and 5 as induction base (IB).

1) $M = \underline{1}$: Let $A \vdash \underline{1} : \tau$. By (var), $A = \tau.A'$ for some A'. Thus $A.\sigma \vdash \underline{1} : \tau$.
2) $M = (M_1\ M_2)$: Let $A \vdash (M_1\ M_2) : \tau$. By (app) one has that $A \vdash M_1 : \rho \to \tau$ and $A \vdash M_2 : \rho$, for some ρ. By IH on structure one has $A.\sigma \vdash M_1 : \rho \to \tau$ and $A.\sigma \vdash M_2 : \rho$. Thus, by (app), $A.\sigma \vdash (M_1\ M_2) : \tau$.
3) $M = \lambda.N$: Let $A \vdash \lambda.N : \tau$. By (lambda), $\rho.A \vdash N : \mu$, where $\tau = \rho \to \mu$. By IH, $\rho.A.\sigma \vdash N : \mu$. Thus, by (lambda), $A.\sigma \vdash \lambda.N : \tau$.
4) $M = N[S]$: Let $A \vdash N[S] : \tau$. By (clos), $A \vdash S \rhd A'$ and $A' \vdash N : \tau$, for some A'. By IH, $A'.\sigma \vdash N : \tau$. Substitution S has to be examined. If $\|N\| > 0$, then by IH on $\|\cdot\|$, as $\|N[S]\| > \|S\|$, one has that $A.\sigma \vdash S \rhd A'.\sigma$. Else, if $\|N\| = 0$ then:

- If $\|S\| = 0$, then Lemma 4 can be applied.
- Otherwise, $S = P.T$ or $S = S_1 \circ S_2$. If $S = P.T$, then by (cons), $A \vdash P : \rho$ and $A \vdash T \rhd A''$, where $A' = \rho.A''$. As $\|P\|, \|T\| < \|S\| = \|N[S]\|$, by IH on $\|\cdot\|$, $A.\sigma \vdash P : \rho$ and $A.\sigma \vdash T \rhd A''.\sigma$. Thus, by (cons), $A.\sigma \vdash P.T \rhd A'.\sigma$. If $S = S_1 \circ S_2$, then by (comp), $A \vdash S_2 \rhd A''$ and $A'' \vdash S_1 \rhd A'$, for some A''. If $\|S_1\|, \|S_2\| > 0$, the result holds by IH on $\|\cdot\|$. Otherwise, at least one of the substitutions has $\|\cdot\|$ greater than 0. Using induction on the structure of S where $\|S\| > 0$, the result holds. Then, $A.\sigma \vdash S_2 \rhd A''.\sigma$ and $A''.\sigma \vdash S_1 \rhd A'.\sigma$. Thus, by (comp), $A.\sigma \vdash S_1 \circ S_2 \rhd A'.\sigma$.

Finally, by (clos), one has that $A.\sigma \vdash N[S] : \tau$.

□

A.2 Proof of Correspondence

Proof (Theorems 1, 3 and 5). The proofs are an adapted version of that given by Wells in [We2002]. Our adaptation deals with de Bruijn indices rather than variables and the proof for $\lambda\sigma$ has an adaptation to deal with substitutions too. Let $u \in \{\lambda dB, \lambda s_e, \lambda\sigma\}$ and $\mathcal{O}_u$ be the index updating operator of each calculus. In other words, $\mathcal{O}_{\lambda dB}(M) = M^+$, $\mathcal{O}_{\lambda s_e}(M) = \varphi_0^2 M$ and $\mathcal{O}_{\lambda\sigma}(M) = M[\uparrow]$. Let $\mathcal{O}_u^1 = \mathcal{O}_u$ and $\mathcal{O}_u^{n+1}(M) = \mathcal{O}_u(\mathcal{O}_u^n(M))$. For a type τ, let $\mathcal{T}(\tau)$ be the set of type variables occurring in τ. For brevity, $\underline{1}[\uparrow^n]$ is denoted as $\underline{n+1}$.

⇒ proof: Let $\Theta_u = \langle A_u, \tau_u\rangle$ be a PT of some term M_u, according to Definitions 11, 13 and 15, and $\Theta'_u = \langle A'_u, \tau'_u\rangle$ be a typing of M_u. By the PT definition for each type system, there is a type substitution s such that $s(A_u) = (A'_u)_{\leq |A_u|}.nil$ and $s(\tau_u) = \tau'_u$. Since $TA_u \Vdash M : \Theta_u$ implies $TA_u \Vdash M : s(\Theta_u)$, for any type substitution s, we have $\Theta_u \leq_{TA_u} s(\Theta_u)$. By the weakening admissible rule for each type system ((λdB-weak), (λs_e-weak)) and ($\lambda\sigma$-tweak), we have $s(\Theta_u) \leq_{TA_u} \Theta'_u$. Thus, Θ_u is PT of M_u, according to Definition 9.

The proof for a $\lambda\sigma$-substitution S with PT $\Theta = \langle A, B\rangle$ according to Definiton 15 and typing $\Theta' = \langle A', B'\rangle$ is similar to the proof for $\lambda\sigma$-terms, using the proper weakening rule ($\lambda\sigma$-sweak).

⇐ proof: Let $\Theta_u = \langle A_u, \tau_u\rangle$ be a PT of some term M_u, according to Definitions 11, 13 and 15, and $\Theta'_u = \langle A'_u, \tau'_u\rangle$ be a typing of M_u which is not PT according to these definitions. Then, there exists a type substitution s such that $s(A_u) = (A'_u)_{\leq |A_u|}.nil$ and $s(\tau_u) = \tau'_u$ and there does not exist any substitution s' such that $s'(A'_u) = (A_u)_{\leq |A'_u|}.nil$ and $s'(\tau'_u) = \tau_u$.

1. If $s(A_u) \neq A'_u$, then $m_u = |A_u| < |A'_u|$. Let $N_u = (\lambda.\mathcal{O}_u(M_u)\ \underline{m_u+1})$.
2. If $s(A_u) = A'_u$, let α be a type variable. Define the functions ϕ_1^u, ϕ_2^u by:

$$\phi_1^u(\alpha,\alpha) = \lambda.\lambda.\big(\underline{1}\ (\underline{2}\ \ \underline{4})\,(\underline{2}\ \ \underline{3})\big)$$

$$\phi_1^u(\sigma\to\tau,\alpha) = \begin{cases} \lambda.\lambda.\big(\underline{1}\ (\underline{3}\ \ \underline{2})\,(\mathcal{O}_u^3(\lambda.\phi_1^u(\sigma,\alpha))\ \ \underline{2})\big), & \text{if } \alpha \in \mathcal{T}(\sigma) \\ \lambda.\big(\mathcal{O}_u^2(\lambda.\phi_1^u(\tau,\alpha))\ (\underline{2}\ \ \underline{1})\big), & \text{otherwise} \end{cases}$$

$$\phi_2^u(\alpha,\alpha) \;=\; \lambda.\lambda.\big(\underline{1}\ (\underline{2}\ \ \underline{3})\,(\underline{2}\ \ \underline{4})\big)$$

$$\phi_2^u(\sigma\to\tau,\alpha) \;=\; \begin{cases} \lambda.\lambda.\big(\underline{1}\ (\underline{4}\ \ \underline{2})\,(\mathcal{O}_u^2(\lambda.\phi_1^u(\sigma,\alpha))\ \ \underline{2})\big), & \text{if } \alpha\in\mathcal{T}(\sigma)\\ \lambda.\big(\mathcal{O}_u(\lambda.\phi_1^u(\tau,\alpha))\ (\underline{3}\ \ \underline{1})\big), & \text{otherwise}\end{cases}$$

(a) Suppose $s(\alpha_u)$ is not a type variable for $\alpha_u\in\mathcal{T}(\Theta_u)$
   i. Suppose $\alpha_u\in\mathcal{T}(\tau_u)$.
      Let $N_u=\Big(\lambda.\big(\lambda.\underline{2}\ \ \lambda.(\mathcal{O}_u(\lambda.\phi_2^u(\tau_u,\alpha_u))\ \lambda.\underline{2})\big)\ M_u\Big)$.
   ii. Suppose $\alpha_u\in\mathcal{T}((A_u)_{i_u})$.
      Let $N_u=\big(\lambda.\mathcal{O}_u(M_u)\ \lambda.(\lambda.\lambda.\phi_2^u((A_u)_{i_u},\alpha_u)\ \underline{i_u{+}1}\ \lambda.\underline{2})\big)$.
(b) Suppose $s(\alpha_u^1)=s(\alpha_u^2)=\beta$ for distinct $\alpha_u^1,\alpha_u^2\in\mathcal{T}(\Theta_u)$
   i. Suppose $\alpha_u^j\in\mathcal{T}((A_u)_{i_{u,j}})$ for $j\in\{1,2\}$.
      Let $P_u^j=\big(\lambda.\phi_1^u((A_u)_{i_{u,j}},\alpha_u^j)\ \ \underline{i_{u,j}{+}1}\big)$
      and $P_u=\lambda.\lambda.\big(\underline{1}\ \mathcal{O}_u(P_u^1)\,\mathcal{O}_u(P_u^2)\big)$. Let $N_u=\big(\lambda.\lambda.\underline{2}\ M_u\,P_u\big)$.
   ii. Suppose $\alpha_u^1\in\mathcal{T}((A_u)_{i_u})$ and $\alpha_u^2\in\mathcal{T}(\tau_u)$.
      Let $P_u=\lambda.\lambda.\big(\underline{1}\ \big(\mathcal{O}_u(\lambda.\phi_1^u((A_u)_{i_u},\alpha_u^1))\ \ \underline{i_u{+}3}\big)\,\mathcal{O}_u(\phi_2^u(\tau_u,\alpha_u^2))\big)$ and $N_u=\big(\lambda.(\lambda.\underline{2}\ \ P_u)\ M_u\big)$.
   iii. Suppose $\alpha_u^i\in\mathcal{T}(\tau_u)$ for $i\in\{1,2\}$.
      Let $P_u=\lambda.\lambda.\big(\underline{1}\ \mathcal{O}_u(\phi_2^u(\tau_u,\alpha_u^1))\ \mathcal{O}_u(\phi_2^u(\tau_u,\alpha_u^2))\big)$
      and $N_u=\big(\lambda.(\lambda.\underline{2}\ \ P_u)\ M_u\big)$.

Then, $N_u\in Terms_{TA_u}(\Theta'_u)\setminus Terms_{TA_u}(\Theta_u)$. Thus, $\Theta'_u\not\leq_{TA_u}\Theta_u$.

As consequence, if Θ'_u is not PT according to Definitions 11, 13 and 15, Θ'_u is not PT according to Definition 9.

Let M be a $\lambda\sigma$-substitution S and $\Theta=\langle A,B\rangle$ be a PT of S, according to Definition 15, and $\Theta'=\langle A',B'\rangle$ be a typing of S which is not PT according to this definition. Then, there is a type substitution s such that $s(A)=A'_{\leq|A|}.nil$ and $s(B)=B'_{\leq|B|}.nil$ and there is no substitution s' such that $s'(A')=A_{\leq|A'|}.nil$ and $s'(B')=B_{\leq|B'|}.nil$.

1. Suppose $s(A)\neq A'$. Then, $m=|A|<|A'|$.
   Let $S_i{=}(\underline{1}\,.\,\underline{2}\,.\cdots.\,\underline{m{+}1}\,.{\uparrow}^{m+1})$ and $T=S\circ S_i$.
2. Otherwise, $s(A)=A'$. Let ϕ_1 be $\phi_1^{\lambda\sigma}$ and ϕ_2 be $\phi_2^{\lambda\sigma}$ as defined above.
   (a) Suppose $s(\alpha)$ is not a type variable for $\alpha\in\mathcal{T}(\Theta)$
      i. Suppose $\alpha\in\mathcal{T}(B_i)$. Let $N{=}\Big(\lambda.\big(\lambda.\underline{2}\ \ \lambda.((\lambda.\phi_2(B_i,\alpha))[\uparrow]\ \lambda.\underline{2})\big)\ \ \underline{i}\Big)$ and let $S'_i=(\underline{1}\,.\,\underline{2}\,.\cdots.\,\underline{i{-}1}\,.N.{\uparrow}^i)$. Let $T=S'_i\circ S$.
      ii. Suppose $\alpha{\in}\mathcal{T}(A_i)$. Let N and S'_i be as above. Let $T{=}S\circ S'_i$.
   (b) Suppose $s(\alpha_1)=s(\alpha_2)=\beta$ for distinct $\alpha_1,\alpha_2\in\mathcal{T}(\Theta)$
      i. Suppose $\alpha_j\in\mathcal{T}(A_{i_j})$ for $j{\in}\{1,2\}$. Let $P_j{=}\big(\lambda.\phi_1(A_{i_j},\alpha_j)\ \ \underline{i_j{+}1}\big)$ and $P{=}\lambda.\lambda.\big(\underline{1}\ \ P_1[\uparrow]\ P_2[\uparrow]\big)$. Let $N_j{=}\big(\lambda.\lambda.\underline{2}\ \ \underline{i_j}\ \ P\big)$, where $j\in\{1,2\}$ and let $S_{i_j}{=}(\underline{1}\,.\,\underline{2}\,.\cdots.\,\underline{i_j{-}1}\,.N_j.\,{\uparrow}^{i_j})$. Let $T=S\circ S_{i_j}$.
      ii. Suppose $\alpha_j\in\mathcal{T}(B_{i_j}),\ j\in\{1,2\}$. Let $P_j=\big(\lambda.\phi_1(B_{i_j},\alpha_j)\ \ \underline{i_j{+}1}\big)$. Then, for P, N_j and S_{i_j} as above, let $T=S_{i_j}\circ S$.
      iii. Suppose $\alpha_1\in\mathcal{T}(A_i)$ and $\alpha_2\in\mathcal{T}(B_j)$. Let $N=\big(\lambda.(\lambda.\underline{2}\ P)\ \underline{j}\,[S]\big)$, where $P=\lambda.\lambda.\big(\underline{1}\ \big((\lambda.\phi_1(A_i,\alpha_1))[\uparrow]\ \ \underline{i{+}3}\big)\,\phi_2(B_j,\alpha_2)[\uparrow]\big)$. Let $T=\big(\underline{1}\,[S].\,\underline{2}\,[S].\cdots.\,\underline{j-1}\,[S].N.({\uparrow}^j\circ S)\big)$.

Then, $T \in Terms_{TA_{\lambda\sigma}}(\Theta') \setminus Terms_{TA_{\lambda\sigma}}(\Theta)$. Thus, $\Theta' \nleq_{TA_{\lambda\sigma}} \Theta$

As consequence, if a typing Θ' of some $\lambda\sigma$-substitution is not PT according to Definition 15, then Θ' is not PT according to Definition 9. □

A.3 Proof of PT

Proof (Theorems 2, 4 and 6). Let M be any term (expression in $\lambda\sigma$) and M' its decorated version. Let R_0 be the set of all sub-terms (sub-expressions) of M'. Starting with the pair $\langle\langle R_0, \emptyset\rangle\rangle$ and applying the rules of the type inference algorithm in Table 3, 4 or 5 one obtains a final pair after a finite number of steps, because after each step the number of elements in the set of decorated sub-terms(sub-expressions) of the pair is decremented. By the uniqueness in the decomposition of the sub-terms (sub-expressions) in each calculus, a unique rule can be applied to each element of R_0. Thus, the process finishes with a pair $\langle\langle \emptyset, E_f\rangle\rangle$, where E_f is a set of first-order equations over context and type variables, according to the rules of the type systems $TA_{\lambda dB}$, $TA_{\lambda s_e}$ and $TA_{\lambda\sigma}$ respectively. An adequate first-order unification algorithm, e.g. see [Hi97], is then applied. And by the correctness, completeness and uniqueness of first-order unification, one has that the algorithm will find an mgu if M is typable. Otherwise, the algorithm will report that there are no unifiers. Consequently, the typing systems $TA_{\lambda dB}$, $TA_{\lambda s_e}$ and $TA_{\lambda\sigma}$ satisfy PT. □