Principal Typings for Explicit Substitutions Calculi*

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

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

Abstract. Having principal typings (for short PT) is an important property of type systems. This property guarantees the possibility of type deduction which means it is possible to develop a complete and terminating type inference mechanism. It is well-known that the simply typed λ-calculus has this property, but recently, J. Wells has introduced a system-independent definition of PT which allows to prove that some type systems do not satisfy PT. The main computational drawback of the λ-calculus is the implicitness of the notion of substitution, a problem which in the last years gave rise to a number of extensions of the λ-calculus where the operation of substitution is treated explicitly. Unfortunately, some of these extensions do not necessarily preserve basic properties of the simply typed λ-calculus such as preservation of strong normalization. We consider two systems of explicit substitutions ($\lambda\sigma$ and λs_e) and we show that they can be accommodated with an adequate notion of PT. Specifically, our results can be summarized as follows:
• We introduce PT notions for the simply typed versions of the $\lambda\sigma$ and the λs_e-calculus according to Wells' system-independent notion of PT.
• We show that these versions of the $\lambda\sigma$ and the λs_e satisfy PT by revisiting previously introduced type inference algorithms.

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 of substitution, which is informally given in the theory of λ-calculus, explicit substitutions calculi are in fact made explicit, but obscurely developed (in and *ad hoc* 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 [10].

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 (SR for short), having principal typings (for short PT), preservation of strong normalization etc. This is a non trivial task; for instance, the $\lambda\sigma$-calculus [1], that is one of the first proposed calculi of explicit substitutions, was reported to break the latter property after some years of its introduction [9]: 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. Here our focus is on the PT property, which means that for any typable term a, there exists a type judgment $\Gamma \vdash a : A$, representing all possible typings for a, where for a typing of a one understands the pair (Γ, A). In the simply typed λ-calculus this corresponds to the existence of *more representative* typings. PT guarantees compositional type inference helping in making a complete/terminating type inference algorithm.

In section 2 we present the type-free versions of λ-calculus in de Bruijn notation, $\lambda\sigma$ and λs_e-calculi. In section 3 we present the type assignment systems background and then we present simply typed systems for

* Research supported by the CNPq. Accepted for presentation at HOR 2007.
** Corresponding author, currently supported by a PhD scholarship of the Brazilian CNPq Research Council at the Heriot-Watt University.
*** Author partially supported by the Brazilian CNPq Research Council.

each calculus. Following type systems presentation, we discuss the general notion of principal typings defined in [11] and present notions of principal typings for λ-calculus in de Bruijn notation, $\lambda\sigma$ and λs_e and prove they are adequate ones. Then we conclude and present future work.

2 The type free calculi

N. G. de Bruijn presented in [6] a λ-term notation where indices replace variable names. This notation is computationally adequated as, for instance, eliminates α-conversions. Besides that, it has the same properties of λ-calculus with names. It follows a presentation of λ-calculus in de Bruijn notation and two explicit substitution calculi, $\lambda\sigma$ and λs_e, which uses de Bruijn notation as well.

2.1 λ-calculus in de Bruijn notation

Definition 1. *The set Λ_{dB} of* ***λ-terms in de Bruijn notation*** *is defined inductively as*
Terms $a ::= \underline{n} \mid (a\ a) \mid \lambda.a$ *where* $n \in \mathbb{N}^* = \mathbb{N} \setminus \{0\}$

Terms like $((((a_1\ a_2)\ a_3)\dots)\ a_n)$ are written as usual $(a_1\, a_2\, \dots\, a_n)$. This simple description of λ-terms structure in de Bruijn notation turns easy the application of induction technique on its structure. An important concept concerning term structure is the *depth* of a subterm.

Definition 2. *Let a be a λ-term. A subterm a_1 of a is* ***n-deep in*** *a, if the least possible index of any free variable in a_1 is greater than n. In other words, a_1 is in between n abstractors.*

Let $i \in \mathbb{N}^*$. We say that $\underline{i}$ **occurs as free index** in a term a if any occurrence of $\underline{i+n}$ is n-deep in a. The β-contraction definition in this notation needs a mechanism which detects and update free indices of terms. It follows an operator similar to the one presented in [3].

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

1. $(a_1\, a_2)^{+i} = (a_1^{+i}\, a_2^{+i})$ 2. $(\lambda.a_1)^{+i} = \lambda.a_1^{+(i+1)}$ 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 a is its 0-lift, denoted as a^+. Intuitively, the lift of a corresponds to increment by 1 all free indices occurring in a. Using the i-lift, we are able to present the definition of the substitution used by β-contractions, similar to the one presented in [3].

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

1. $\{\underline{n}/b\}(a_1\ a_2) = (\{\underline{n}/b\}a_1\ \{\underline{n}/b\}a_2)$
2. $\{\underline{n}/b\}\lambda.a_1 = \lambda.\{\underline{n+1}/b^+\}a_1$
3. $\{\underline{n}/b\}\underline{m} = \begin{cases} \underline{m-1}, & \text{if } m > n \\ b, & \text{if } m = n \\ \underline{m}, & \text{if } m < n. \end{cases}$

Observe that, in item 2 of Definition 4, the lift operator is used to avoid captures of free indices in b. We present the β-contraction as defined in [3].

Definition 5. *β-**contraction** of λ-terms in de Bruijn notation is defined as $(\lambda.a\ b) \to_\beta \{\underline{1}/b\}a$.*

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

2.2 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**.

Definition 6. *The syntax of the type free $\lambda\sigma$-calculus is given by*
Terms $a ::= \underline{1} \mid (a\ a) \mid \lambda.a \mid a[s]$ ***Substitutions*** $s ::= id \mid \uparrow \mid a.s \mid s \circ s$

Substitutions are lists of the form $b/\underline{i}$ indicating that the index $\underline{i}$ should be changed to the term b. id represents a substitution of the form $\{\underline{1}/\underline{1}, \underline{2}/\underline{2}, \dots\}$ and $\uparrow$ is the substitution $\{\underline{i+1}/\underline{i} \mid i \in \mathbb{N}^*\}$. $s \circ s$ represents the composition of substitutions. $\underline{1}[\uparrow^n]$, where $n \in \mathbb{N}^*$, codifies the de Bruijn index $\underline{n+1}$. $\underline{i}[s]$ represents the value of $\underline{i}$ through the substitution s, which can be seen as a function $s(i)$. The substitution $a.s$ has the form $\{a/\underline{1}, s(i)/\underline{i+1}\}$, called the **cons of** a **in** s. $a[b.id]$ starts the simulation of the β-reduction of $(\lambda.a\ b)$ 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. The Table 1 includes the rewriting system of the $\lambda\sigma$-calculus, as presented in [7].

$(\lambda.a\ b)$	$\longrightarrow a[b.id]$	$(Beta)$	$\uparrow \circ (a.s)$	$\longrightarrow s$	$(ShiftCons)$
$(a\ b)[s]$	$\longrightarrow (a[s]\ b[s])$	(App)	$(s_1 \circ s_2) \circ s_3$	$\longrightarrow s_1 \circ (s_2 \circ s_3)$	$(AssEnv)$
$1[a.s]$	$\longrightarrow a$	$(VarCons)$	$(a.s) \circ t$	$\longrightarrow a[t].(s \circ t)$	$(MapEnv)$
$a[id]$	$\longrightarrow a$	(Id)	$s \circ id$	$\longrightarrow s$	(IdR)
$(\lambda.a)[s]$	$\longrightarrow \lambda.(a[1.(s \circ \uparrow)])$	(Abs)	$1.\uparrow$	$\longrightarrow id$	$(VarShift)$
$(a[s])[t]$	$\longrightarrow a[s \circ t]$	$(Clos)$	$1[s].(\uparrow \circ s)$	$\longrightarrow s$	$(Scons)$
$id \circ s$	$\longrightarrow s$	(IdL)	$\lambda.(a\ \underline{1})$	$\rightarrow\ b$ if $a =_\sigma b[\uparrow]$	(Eta)

Table 1. The rewriting system for the $\lambda\sigma$-calculus with Eta rule

This system without (Eta) is equivalent to that of [1]. 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 $=_\sigma$.

2.3 The λs_e-Calculus

In contrast with the $\lambda\sigma$-calculus, the λs_e-calculus has a sole sort of objects maintaining its syntax closer to the λ-calculus. The λs_e-calculus controls the atomization of the substitution operation by introducing the use of arithmetic constraints through two operators σ and φ, for substitution and updating, respectively.

Definition 7. *The syntax of the untyped λs_e-calculus, where $n, i, j \in \mathbb{N}^*$ and $k \in \mathbb{N}$ is given as*
Terms $a ::= \underline{n} \mid (a\ a) \mid \lambda.a \mid a\,\sigma^i a \mid \varphi_k^j\, a$

The term $a\,\sigma^i b$ represents the term $\{\underline{i}/b\}a$; i.e., substitution of free occurrences of $\underline{i}$ in a by b, updating free variables in a (and in b). The term $\varphi_k^j\, a$ represents $j-1$ applications of the k-lift to the term a; i.e., $a^{+k^{(j-1)}}$. Table 2 contains the rewriting rules of the λs_e-calculus and the rule (Eta), as given in [3].

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

3 The Type Systems

Definition 8. *The syntax of the* **simple types** *and* **contexts** *is given as follows:*
Types $A ::= K \mid A \to A$ ***Contexts*** $\Gamma ::= nil \mid A.\Gamma$

$(\lambda.a\ b)$	$\longrightarrow$	$a\,\sigma^1 b$	(σ-generation)
$(\lambda.a)\,\sigma^i b$	$\longrightarrow$	$\lambda.(a\,\sigma^{i+1} b)$	(σ-λ-transition)
$(a_1\ a_2)\,\sigma^i b$	$\longrightarrow$	$((a_1\,\sigma^i b)\ (a_2\,\sigma^i b))$	(σ-app-transition)
$\underline{n}\,\sigma^i b$	$\longrightarrow$	$\begin{cases} \underline{n-1} & \text{if } n > i \\ \varphi_0^i\, b & \text{if } n = i \\ \underline{n} & \text{if } n < i \end{cases}$	(σ-destruction)
$\varphi_k^i\,(\lambda.a)$	$\longrightarrow$	$\lambda.(\varphi_{k+1}^i\, a)$	(φ-λ-transition)
$\varphi_k^i\,(a_1\ a_2)$	$\longrightarrow$	$((\varphi_k^i\, a_1)\ (\varphi_k^i\, a_2))$	(φ-app-transition)
$\varphi_k^i\,\underline{n}$	$\longrightarrow$	$\begin{cases} \underline{n+i-1} & \text{if } n > k \\ \underline{n} & \text{if } n \le k \end{cases}$	(φ-destruction)
$(a_1\,\sigma^i a_2)\,\sigma^j b$	$\longrightarrow$	$(a_1\,\sigma^{j+1} b)\,\sigma^i(a_2\,\sigma^{j-i+1} b)$ if $i \le j$	(σ-σ-transition)
$(\varphi_k^i\, a)\,\sigma^j b$	$\longrightarrow$	$\varphi_k^{i-1}\, a$ if $k < j < k+i$	(σ-φ-transition 1)
$(\varphi_k^i\, a)\,\sigma^j b$	$\longrightarrow$	$\varphi_k^i\,(a\,\sigma^{j-i+1} b)$ if $k+i \le j$	(σ-φ-transition 2)
$\varphi_k^i\,(a\,\sigma^j b)$	$\longrightarrow$	$(\varphi_{k+1}^i\, a)\,\sigma^j(\varphi_{k+1-j}^i\, b)$ if $j \le k+1$	(φ-σ-transition)
$\varphi_k^i\,(\varphi_l^j\, a)$	$\longrightarrow$	$\varphi_l^j\,(\varphi_{k+1-j}^i\, a)$ if $l+j \le k$	(φ-φ-transition 1)
$\varphi_k^i\,(\varphi_l^j\, a)$	$\longrightarrow$	$\varphi_l^{j+i-1}\, a$ if $l \le k < l+j$	(φ-φ-transition 2)
$\lambda.(a\ \underline{1}\,)$	$\rightarrow$	b if $a =_{s_e} \varphi_0^2\, b$	(Eta)

Table 2. The rewriting system of the λs_e-calculus with Eta rule

K ranges over **type variables**. A **type assignment system** 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 S rules to associate a type to a term. In simply typed λ-calculi [8], the typable terms are strongly normalizing. In other words, the computation corresponding to the typed term stops. The ordered pair (Γ, A), of a context and a type, is called a **typing in** S. For a term a, $\Gamma \vdash a : A$ denotes that a has type A in context Γ, and (Γ, A) is called a **typing of** a. Let $\tau = (\Gamma, A)$ be a typing in S. $S \rhd a : \tau$ denotes that τ is a typing of a in S. Let $\mathcal{T}(A)$ be the set of type variables occurring in A. Then, an extension for contexts and typings can be defined as $\mathcal{T}(B.\Gamma) = \mathcal{T}(B) \cup \mathcal{T}(\Gamma)$, where $\mathcal{T}(nil) = \emptyset$, and $\mathcal{T}((\Gamma, A)) = \mathcal{T}(\Gamma) \cup \mathcal{T}(A)$.

The contexts for λ-terms in de Bruijn notation are sequences of types. Let Γ be some context and $n \in \mathbb{N}$. Then $\Gamma_{<n}$ denotes the first $n-1$ types of Γ. Similarly we define $\Gamma_{>n}$, $\Gamma_{\le n}$ and $\Gamma_{\ge n}$. For $n = 0$, $\Gamma_{\le 0}.\Gamma = \Gamma_{<0}.\Gamma = \Gamma$. The length of Γ is defined as $|A.\Gamma'| = 1 + |\Gamma'|$ and $|nil| = 0$. The addition of some type A at the end of a context Γ is defined as $\Gamma.A = \Gamma_{\le m}.A.nil$, where $|\Gamma| = m$.

Given a term a, an interesting question is whether it is typable in S or not. Note that, we are using the so-called Curry-style or implicit typing, where in terms of the form $\lambda.a$ 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 in some sense 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 τ be a typing in S and $\mathbf{Terms_S}(\tau)$=$\{a | S \rhd a{:}\tau\}$. J. Wells introduced in [11] a system-independent definition of PT and proved that it generalises previous system-specific definitions.

Definition 9 ([11]). *A typing τ in system S is principal for some term a if $S \rhd a : \tau$ and for any τ' such that $S \rhd a : \tau'$ we have that $\tau \le_S \tau'$, where $\tau_1 \le_S \tau_2 \iff Terms_S(\tau_1) \subseteq Terms_S(\tau_2)$.*

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 $s(A{\rightarrow}B) = s(A){\rightarrow}s(B)$ and the extension for sequential contexts as $s(A.\Gamma) = s(A).s(\Gamma)$ and $s(nil) = nil$. The extension for typings is done as $s(\tau) = (s(\Gamma), s(A))$.

3.1 Principal typings for the simply typed λ-calculus in de Bruijn notation

Definition 10. ***(The System $TA_{\lambda dB}$)*** *The* ***System of Simple Types for λ-Calculus in de Bruijn Notation****, denoted as $TA_{\lambda dB}$, is given by the following typing rules*

$$(\lambda dB\text{-var}) \quad A.\Gamma \vdash \underline{1} : A \qquad (\lambda dB\text{-varn}) \quad \frac{\Gamma \vdash \underline{n} : B}{A.\Gamma \vdash \underline{n+1} : B}$$

$$(\lambda dB\text{-lambda}) \ \frac{A.\Gamma \vdash b : B}{\Gamma \vdash \lambda.b : A \to B} \qquad (\lambda dB\text{-app}) \ \frac{\Gamma \vdash a : A \to B \quad \Gamma \vdash b : A}{\Gamma \vdash (a\ b) : B}$$

This system is similar to TA_λ([8]). The rule (λdB-varn) allows the construction of contexts as sequences.

Example 1. Deduction of $A \to B.nil \vdash \lambda.(\underline{2}\ \underline{1}) : A \to B$

$$\frac{\dfrac{\dfrac{A \to B.nil \vdash \underline{1} : A \to B}{A.A \to B.nil \vdash \underline{2} : A \to B}(\lambda dB\text{-varn}) \qquad A.A \to B.nil \vdash \underline{1} : A (\lambda dB\text{-var})}{A.A \to B.nil \vdash (\underline{2}\ \underline{1}) : B}(\lambda dB\text{-app})}{A \to B.nil \vdash \lambda.(\underline{2}\ \underline{1}) : A \to B}(\lambda dB\text{-lambda})$$

Lemma 1. *Let $i \in \mathbb{N}$. If $\Gamma \vdash_{TA_{\lambda dB}} a : A$, then $\Gamma_{\leq i}.B.\Gamma_{>i} \vdash_{TA_{\lambda dB}} a^{+i} : A$.*

Proof. Induction on a structure. Note that if $i \geq m$, where $m = |\Gamma|$, then B is added at the end of Γ.

1) $a = \underline{n}$: Suppose $\Gamma \vdash \underline{n} : A$. If $n \leq i$, then $\underline{n}^{+i} = \underline{n}$. The B addition at i+1-th position changes only types of indices greater or equal to $\underline{i+1}$, thus one has trivially that $\Gamma_{\leq i}.B.\Gamma_{>i} \vdash \underline{n} : A$. If $n > i$, then $\underline{n}^{+i} = \underline{n+1}$. By ($TA_{dB}$-varn) i times one has $\Gamma_{>i} \vdash \underline{n-i} : A$. Thus, by ($TA_{dB}$-varn) applied $i+1$ times, one has that $\Gamma_{\leq i}.B.\Gamma_{>i} \vdash \underline{n+1} : A$.
2) $a = (b\,c)$: Suppose $\Gamma \vdash (b\,c) : A$. By (TA_{dB}-app) one has that $\Gamma \vdash b : C \to A$ and $\Gamma \vdash c : C$. By IH one has $\Gamma_{\leq i}.B.\Gamma_{>i} \vdash b^{+i} : C \to A$ and $\Gamma_{\leq i}.B.\Gamma_{>i} \vdash c^{+i} : C$. Thus, by ($TA_{dB}$-app), $\Gamma_{\leq i}.B.\Gamma_{>i} \vdash (b^{+i}\,c^{+i}) : A$.
3) $a = \lambda.b$: Suppose $\Gamma \vdash \lambda.b : A$. By ($TA_{dB}$-lambda) one has that $C.\Gamma \vdash b : D$, where $A = C \to D$. By IH one has $C.\Gamma_{\leq i}.B.\Gamma_{>i} \vdash b^{+(i+1)} : A$. Thus, by ($TA_{dB}$-lambda), $\Gamma_{\leq i}.B.\Gamma_{>i} \vdash \lambda.b^{+(i+1)} : C \to D = A$. □

Proposition 1. *The rule (λdB-weak) is admissible in System $TA_{\lambda dB}$, where* $\dfrac{\Gamma \vdash a : A}{\Gamma.B \vdash a : A}(\lambda dB\text{-weak})$.

Proof. Let $\Gamma \vdash_{TA_{\lambda dB}} a : A$. As for λ-calculus with names, all information about a free varibles is in context Γ. Then, a maximum value for a free index occurrence, at 0-deep in a, is $m = |\Gamma|$. Consequently, $a^{+j} = a$ for any $j \geq m$. Using the statement of Lemma 1 for $i = m$, we have that $\Gamma.B \vdash_{TA_{\lambda dB}} a : A$, for any type B. 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 Γ would require updating some free indices, then a^{+i} would correspond to a different function from the one to which term a corresponds. □

Using the rule (λdB-weak) and type substitution, we can define principal typing for the λ-calculus in de Bruijn notation similarly to the definition of [11] for Hindley's Principal Typing.

Definition 11. *A* ***principal typing*** *in $TA_{\lambda dB}$ of a λ-term a is the typing $\tau = (\Gamma, B)$ such that*

1. *$TA_{\lambda dB} \rhd a : \tau$*
2. *If $TA_{\lambda dB} \rhd a : \tau'$ for any typing $\tau' = (\Gamma', B')$, then exists some substitution s such that $s(\Gamma) = \Gamma'_{\leq |\Gamma|}.nil$ and $s(B) = B'$.*

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 τ is principal in $TA_{\lambda dB}$ according to Definition 11 iff τ is principal in $TA_{\lambda dB}$ according to Definition 9.*

Proof. $\Rightarrow$ *proof:* Let $\tau = (\Gamma, B)$ be a PT of some term a, according to Definition 11, and $\tau' = (\Gamma', B')$ be a typing of a. By Definition 11, we have that exists a type substitution s such that $s(\Gamma) = \Gamma'_{\leq|\Gamma|}.nil$ and $s(B) = B'$. By the property which says that if $TA_{\lambda dB} \rhd a : \tau$, then $TA_{\lambda dB} \rhd a : s(\tau)$, for any type substitution s, we have $\tau \leq_{TA_{\lambda dB}} s(\tau)$. By ($\lambda dB$-weak), we have that $s(\tau) \leq_{TA_{\lambda dB}} \tau'$. Thus, τ is PT of a, according to Definition 9.

$\Leftarrow$ *proof:* Let $\tau = (\Gamma, B)$ be a PT of some term a, according to Definition 11, and $\tau' = (\Gamma', B')$ be a typing of a which is not PT according to this definition. Then, exists a type substitution s such that $s(\Gamma) = \Gamma'_{\leq|\Gamma|}.nil$ and $s(B) = B'$ and does not exist any substitution s' such that $s'(\Gamma') = \Gamma_{\leq|\Gamma'|}.nil$ and $s'(B') = B$.

1. Suppose $s(\Gamma) \neq \Gamma'$. Then, $m = |\Gamma| < |\Gamma'|$. Let $b = (\lambda.a^+\ \underline{m+1}\,)$.
2. Otherwise, $s(\Gamma) = \Gamma'$. Let K be a type variable. Define the function ϕ_1 as:

$$\begin{aligned}
\phi_1(K,K) &= \lambda.\lambda.\big(\underline{1}\ (\underline{2}\ \ \underline{4})\ (\underline{2}\ \ \underline{3})\big)\\
\phi_1(A\to B,K) &= \lambda.\lambda.\big(\underline{1}\ (\underline{3}\ \ \underline{2})\ ((\lambda.\phi_1(A,K))^{+^3}\ \ \underline{2})\big), \quad \text{if } K\in\mathcal{T}(A)\\
\phi_1(A\to B,K) &= \lambda.\big((\lambda.\phi_1(B,K))^{+^2}\ (\underline{2}\ \ \underline{1})\big), \quad \text{otherwise}
\end{aligned}$$

and define function ϕ_2 as:

$$\begin{aligned}
\phi_2(K,K) &= \lambda.\lambda.\big(\underline{1}\ (\underline{2}\ \ \underline{3})\ (\underline{2}\ \ \underline{4})\big)\\
\phi_2(A\to B,K) &= \lambda.\lambda.\big(\underline{1}\ (\underline{4}\ \ \underline{2})\ ((\lambda.\phi_1(A,K))^{+^2}\ \ \underline{2})\big), \quad \text{if } K\in\mathcal{T}(A)\\
\phi_2(A\to B,K) &= \lambda.\big((\lambda.\phi_1(B,K))^{+}\ (\underline{3}\ \ \underline{1})\big), \quad \text{otherwise}
\end{aligned}$$

(a) Suppose $s(K)$ is not a type variable for $K \in \mathcal{T}(\tau)$
   i. Suppose $K \in \mathcal{T}(B)$. Let $b = \Big(\lambda.\big(\lambda.\underline{2}\ \ \lambda.((\lambda.\phi_2(B,K))^+\ \lambda.\underline{2})\big)\ a\Big)$.
   ii. Suppose $K \in \mathcal{T}(\Gamma_i)$. Let $b = \big(\lambda.a^+\ \lambda.(\lambda.\lambda.\phi_2(\Gamma_i,K)\ \underline{i+1}\ \lambda.\underline{2})\big)$.

(b) Suppose $s(K_1) = s(K_2) = L$ for distinct $K_1, K_2 \in \mathcal{T}(\tau)$
   i. Suppose $K_j \in \mathcal{T}(\Gamma_{i_j})$ for $j \in \{1,2\}$. Let $p_j = \big(\lambda.\phi_1(\Gamma_{i_j}, K_j)\ \ \underline{i_j+1}\big)$ and $p = \lambda.\lambda.\big(\underline{1}\ p_1^+\ p_2^+\big)$. Let $b = \big(\lambda.\lambda.\underline{2}\ a\, p\big)$.
   ii. Suppose $K_1 \in \mathcal{T}(\Gamma_i)$ and $K_2 \in \mathcal{T}(B)$. Let $p = \lambda.\lambda.\big(\underline{1}\ \big((\lambda.\phi_1(\Gamma_i,K_1))^+\ \ \underline{i+3}\big)\ (\phi_2(B,K_2))^+\big)$ and $b = \big(\lambda.(\lambda.\underline{2}\ \ p)\ a\big)$.
   iii. Suppose $K_i \in \mathcal{T}(B)$ for $i \in \{1,2\}$. Let $p = \lambda.\lambda.\big(\underline{1}\ (\phi_2(B,K_1))^+\ (\phi_2(B,K_2))^+\big)$ and $b = \big(\lambda.(\lambda.\underline{2}\ \ p)\ a\big)$.

Then, $b \in Terms_{TA_{\lambda dB}}(\tau') \setminus Terms_{TA_{\lambda dB}}(\tau)$. Thus, $\tau' \not\leq_{TA_{\lambda dB}} \tau$.

As consequence, if τ' is not PT according to Definition 11, τ' is not PT according to Definition 9. □

A type inference algorithm for terms from $TA_{\lambda dB}$ is presented, similar to the one in [4] for λs_e. Given any term a, decorate each subterm with a new type variable as subscript and a new context variable as superscript, obtaining a new term denoted as a'. For example, for term $\lambda.(\underline{2}\ \ \underline{1}\,)$ we have the decorated term $(\lambda.(\underline{2}\ {}^{\Gamma_1}_{A_1}\ \ \underline{1}\ {}^{\Gamma_1}_{A_2})^{\Gamma_3}_{A_3})^{\Gamma_4}_{A_4}$. Then, rules from Table 3 are applied to pairs of the form $\langle R, E\rangle$, where R is a set of decorated terms and E a set of equations on type and context variables.

Type inference for a starts with $\langle R_0, \varnothing\rangle$, where R_0 is the set of all a' subterms. The rules from Table 3 are applied until reaches $\langle \varnothing, E_f\rangle$, where E_f is a set of first-order equations over context and type variables.

Example 2. Let $a = \lambda.(\underline{2}\ \ \underline{1}\,)$. Then $a' = (\lambda.(\underline{2}\ {}^{\Gamma_1}_{A_1}\ \ \underline{1}\ {}^{\Gamma_2}_{A_2})^{\Gamma_3}_{A_3})^{\Gamma_4}_{A_4}$ and $R_0 = \{(\underline{2}\ {}^{\Gamma_1}_{A_1}\ \ \underline{1}\ {}^{\Gamma_2}_{A_2})^{\Gamma_3}_{A_3}, (\lambda.(\underline{2}\ {}^{\Gamma_1}_{A_1}\ \ \underline{1}\ {}^{\Gamma_2}_{A_2})^{\Gamma_3}_{A_3})^{\Gamma_4}_{A_4}, \underline{2}\ {}^{\Gamma_1}_{A_1}, \underline{1}\ {}^{\Gamma_2}_{A_2}\}$. Using the rules in Table 3 we have the following reduction

$$\begin{aligned}
\langle R_0, \emptyset\rangle &\to \langle R_1 = R_0 \setminus \{\underline{2}\ {}^{\Gamma_1}_{A_1}\}, E_1 = \{\Gamma_1 = A_1'.A_1.\Gamma_1'\}\rangle && \text{(Varn)}\\
&\to \langle R_2 = R_1 \setminus \{\underline{1}\ {}^{\Gamma_2}_{A_2}\}, E_2 = E_1 \cup \{\Gamma_2 = A_2.\Gamma_2'\}\rangle && \text{(Var)}\\
&\to \langle R_3 = R_2 \setminus \{(\underline{2}\ {}^{\Gamma_1}_{A_1}\ \ \underline{1}\ {}^{\Gamma_2}_{A_2})^{\Gamma_3}_{A_3}\}, E_3 = E_2 \cup \{\Gamma_1 = \Gamma_2, \Gamma_2 = \Gamma_3, A_1 = A_2 \to A_3\}\rangle && \text{(App)}\\
&\to \langle R_4 = R_3 \setminus \{(\lambda.(\underline{2}\ {}^{\Gamma_1}_{A_1}\ \ \underline{1}\ {}^{\Gamma_2}_{A_2})^{\Gamma_3}_{A_3})^{\Gamma_4}_{A_4}\}, E_4 = E_3 \cup \{A_4 = A_1^* \to A_3, \Gamma_3 = A_1^*.\Gamma_4\}\rangle && \text{(Lambda)}
\end{aligned}$$

(Var)	$\langle R \cup \{\underline{1}_A^{\Gamma}\}, E\rangle$	$\rightarrow \langle R, E \cup \{\Gamma = A.\Gamma'\}\rangle$,where Γ' is a fresh context variable;
(Varn)	$\langle R \cup \{\underline{n}_A^{\Gamma}\}, E\rangle$	$\rightarrow \langle R, E \cup \{\Gamma = A_1'.\cdots.A_{n-1}'.A.\Gamma'\}\rangle$,where $A_1', \ldots, A_{n-1}'$ and Γ' are fresh type and context variables;
(Lambda)	$\langle R \cup \{(\lambda.a_{A_1}^{\Gamma_1})_{A_2}^{\Gamma_2}\}, E\rangle$	$\rightarrow \langle R, E \cup \{A_2 = A^* \rightarrow A_1, \Gamma_1 = A^*.\Gamma_2\}\rangle$, where A^* is a fresh type variable;
(App)	$\langle R \cup \{(a_{A_1}^{\Gamma_1}\ b_{A_2}^{\Gamma_2})_{A_3}^{\Gamma_3}\}, E\rangle$	$\rightarrow \langle R, E \cup \{\Gamma_1 = \Gamma_2, \Gamma_2 = \Gamma_3, A_1 = A_2 \rightarrow A_3\}\rangle$

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

Thus, $E_4 = E_f$. Solving the trivial equation over context variables, i.e. $\Gamma_1 = \Gamma_2 = \Gamma_3$, and using variables of smaller subscripts, one gets $\{A_1 = A_2 \rightarrow A_3, A_4 = A_1^* \rightarrow A_3, \Gamma_1 = A_1'.A_1.\Gamma_1', \Gamma_1 = A_2.\Gamma_2', \Gamma_1 = A_1^*.\Gamma_4\}$. Thus, simplifying one gets $\{A_1 = A_2 \rightarrow A_3, A_4 = A_1^* \rightarrow A_3, A_1'.A_1.\Gamma_1' = A_2.\Gamma_2' = A_1^*.\Gamma_4\}$. From these equations one gets the most general unifier (mgu for short) $A_4 = A_2 \rightarrow A_3$ and $\Gamma_4 = (A_2 \rightarrow A_3).\Gamma_1'$, for the variables of interest.

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.*

Proof. For a term a type inference starts with the pair $\langle R_0, \varnothing\rangle$, where R_0 is the set of all a' subterms. As each subterm is an atomic term, an application or an abstraction, mutually exclusive, only one of these rules can be applied on an element of R_0. The rule (Varn) can be extended, allowing application on term $\underline{1}_A^{\Gamma}$, however, the rules are kept according to system $TA_{\lambda dB}$. Thus, each rule from Table 3 corresponds to one $TA_{\lambda dB}$ rule and includes new equations in E about context and type variables, which satisfies the corresponding $TA_{\lambda dB}$ rule. Each application of one of the above rules decrements the number of elements in set R. Thus, after a finite number of steps we have the pair $\langle \varnothing, E_n\rangle$, where E_n is a set of equations about context and type variables, representing a first order unification problem. It can be seen that: if the unification fails, the term is not typable; otherwise, a mgu, which applied to the outermost context and type variables, gives Γ and A such that $\Gamma \vdash a : A$ and (Γ, A) satifies the condition of Definition 11. Thus, we have the PT of a in $TA_{\lambda dB}$. □

3.2 Principal typings for the simply typed $\lambda\sigma$

The typed version is presented in Curry style, instead of Church style presented in [7]. Thus, the syntax of $\lambda\sigma$-terms and the rules are the same as the untyped version.

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. $s \rhd \Gamma$ denotes that the object of sort substitution s has type Γ.

Definition 12 (The System $TA_{\lambda\sigma}$). *The **System of Simple Types for** $\lambda\sigma$, denoted as $TA_{\lambda\sigma}$, is given by the following typing rules.*

$$\textit{(var)}\quad A.\Gamma \vdash \underline{1} : A \qquad \textit{(lambda)}\quad \frac{A.\Gamma \vdash b : B}{\Gamma \vdash \lambda.b : A \rightarrow B}$$

$$\textit{(app)}\quad \frac{\Gamma \vdash a : A \rightarrow B \quad \Gamma \vdash b : A}{\Gamma \vdash (a\ b) : B} \qquad \textit{(clos)}\quad \frac{\Gamma \vdash s \rhd \Gamma' \qquad \Gamma' \vdash a : A}{\Gamma \vdash a[s] : A}$$

$$\textit{(id)}\quad \Gamma \vdash id \rhd \Gamma \qquad \textit{(shift)}\quad A.\Gamma \vdash \uparrow \rhd \Gamma$$

$$\textit{(cons)}\quad \frac{\Gamma \vdash a : A \qquad \Gamma \vdash s \rhd \Gamma'}{\Gamma \vdash a.s \rhd A.\Gamma'} \qquad \textit{(comp)}\quad \frac{\Gamma \vdash s'' \rhd \Gamma'' \quad \Gamma'' \vdash s' \rhd \Gamma'}{\Gamma \vdash s' \circ s'' \rhd \Gamma'}$$

Observe that the name of the typing rules begin with lower-case letters, while the rewriting rules with upper-case letters. We have verified that this version of $\lambda\sigma$ in Curry style has the same properties as the version of $\lambda\sigma$ in Church style given in [7].

Example 3. In TA_λ (and $TA_{\lambda\sigma}$), $(B.C.nil, (C{\to}A){\to}A)$ is a typing of $a = (\lambda.(\lambda.(\underline{1}\ \underline{2}))\ \underline{2})$. By $(Beta)$ and (Abs), a reduces to $b = \lambda.((\underline{1}\ \underline{2})[\underline{1}.(\underline{2}.id)\circ\uparrow])$. Note that $\underline{2}$ abbreviates $\underline{1}[\uparrow]$. We show that $B.C.nil \vdash b : (C \to A) \to A$. For brevity, contexts in a deduction are denoted without nil.

First, we have

$$\frac{B.C \vdash \uparrow \rhd C\,(\text{shift}) \qquad C \vdash \underline{1} : C\,(\text{var})}{B.C \vdash \underline{1}[\uparrow] : C}(\text{clos}).$$

Analogously we have $C \to A.C.B.C \vdash \underline{1}[\uparrow] : C$. Furthermore,

$$\frac{C \to A.B.C \vdash \uparrow \rhd B.C\,(\text{shift}) \qquad \dfrac{B.C \vdash \underline{2} : C \quad B.C \vdash id \rhd B.C\,(\text{id})}{B.C \vdash \underline{2}.id \rhd C.B.C}(\text{cons})}{C \to A.B.C \vdash (\underline{2}.id)\circ\uparrow \rhd C.B.C}(\text{comp})$$

Hence,

$$\frac{C \to A.B.C \vdash \underline{1} : C \to A\,(\text{var}) \quad C \to A.B.C \vdash (\underline{2}.id)\circ\uparrow \rhd C.B.C}{C \to A.B.C \vdash \underline{1}.(\underline{2}.id)\circ\uparrow \rhd C \to A.C.B.C}(\text{cons})$$

Also,

$$\frac{C \to A.C.B.C \vdash \underline{1} : C \to A\,(\text{var}) \quad C \to A.C.B.C \vdash \underline{2} : C}{C \to A.C.B.C \vdash (\underline{1}\ \underline{2}) : A}(\text{app})$$

Consequently,

$$\frac{\dfrac{C \to A.B.C \vdash \underline{1}.(\underline{2}.id)\circ\uparrow \rhd C \to A.C.B.C \qquad C \to A.C.B.C \vdash (\underline{1}\ \underline{2}) : A}{C \to A.B.C \vdash (\underline{1}\ \underline{2})[\underline{1}.(\underline{2}.id)\circ\uparrow] : A}(\text{clos})}{B.C \vdash \lambda.((\underline{1}\ \underline{2})[\underline{1}.(\underline{2}.id)\circ\uparrow]) : (C \to A) \to A}(\text{lambda})$$

Since subterms of $\lambda\sigma$-terms can be of sort either term or substitution, we will enclose both sorts by the denomination $\lambda\sigma$-expression (sub-expression). 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 $\tau = (\Gamma, \mathbb{T})$ 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 2 (Weakening for $\lambda\sigma$). *Let a be a $\lambda\sigma$-term and s a $\lambda\sigma$-substitution. If $\Gamma \vdash a : A$, then $\Gamma.B \vdash a : A$, for any type B. Similarly, if $\Gamma \vdash s \rhd \Gamma'$, then $\Gamma.B \vdash s \rhd \Gamma'.B$.*

Proof. see appendix.

Proposition 2. *The rules $(\lambda\sigma\text{-tweak})$ and $(\lambda\sigma\text{-sweak})$ are admissible in System $TA_{\lambda\sigma}$, where*

$$\frac{\Gamma \vdash a : A}{\Gamma.B \vdash a : A}(\lambda\sigma\text{-tweak}) \qquad \frac{\Gamma \vdash s \rhd \Gamma'}{\Gamma.B \vdash s \rhd \Gamma'.B}(\lambda\sigma\text{-sweak})$$

The rules given in Proposition 2 and the type substitution allow us present a definition for PT in $TA_{\lambda\sigma}$.

Definition 13 (Principal Typings in $TA_{\lambda\sigma}$). *A **principal typing** of an expression a in $TA_{\lambda\sigma}$ is a typing $\tau = (\Gamma, \mathbb{T})$ such that*

1. *$TA_{\lambda\sigma} \rhd a : \tau$*
2. *If $TA_{\lambda\sigma} \rhd a : \tau'$ for any typing $\tau' = (\Gamma', \mathbb{T}')$, then there exists a substitution s such that $s(\Gamma) = \Gamma'_{\leq|\Gamma|}.nil$ 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 [11].

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

Proof. For brevity, $\underline{1}\,[\uparrow^n]$ is denoted as $\underline{n+1}$.
Let a be a $\lambda\sigma$-term.

$\Rightarrow$ *proof:* Let $\tau = (\Gamma, B)$ be a PT of a, according to Definition 13, and $\tau' = (\Gamma', B')$ be a typing of a. By Definition 13, we have that exists a type substitution s such that $s(\Gamma) = \Gamma'_{\leq|\Gamma|}.nil$ and $s(B) = B'$. By the property which says that if $TA_{\lambda\sigma} \rhd a : \tau$, then $TA_{\lambda\sigma} \rhd a : s(\tau)$, for any type substitution s, we have $\tau \leq_{TA_{\lambda\sigma}} s(\tau)$. By ($\lambda\sigma$-tweak), we have that $s(\tau) \leq_{TA_{\lambda dB}} \tau'$. Thus, τ is PT of a, according to Definition 9.

$\Leftarrow$ *proof:* Let $\tau = (\Gamma, B)$ be a PT of a, according to Definition 13, and $\tau' = (\Gamma', B')$ be a typing of a which is not PT according to this definition. Then, exists a type substitution s such that $s(\Gamma) = \Gamma'_{\leq|\Gamma|}.nil$ and $s(B) = B'$ and does not exist any substitution s' such that $s'(\Gamma') = \Gamma_{\leq|\Gamma'|}.nil$ and $s'(B') = B$.

1. Suppose $s(\Gamma) \neq \Gamma'$. Then, $m = |\Gamma| < |\Gamma'|$. Let $b = \big(\lambda.(a[\uparrow])\ \underline{m+1}\big)$.
2. Otherwise, $s(\Gamma) = \Gamma'$. Let K be a type variable. Define the function ϕ_1 as:

$$\begin{aligned}
\phi_1(K,K) &= \lambda.\lambda.\big(\underline{1}\ (\underline{2}\ \ \underline{4})\,(\underline{2}\ \ \underline{3})\big)\\
\phi_1(A\to B,K) &= \lambda.\lambda.\big(\underline{1}\ (\underline{3}\ \ \underline{2})\,((\lambda.\phi_1(A,K))[\uparrow^3]\ \ \underline{2})\big), \quad \text{if } K\in\mathcal{T}(A)\\
\phi_1(A\to B,K) &= \lambda.\big((\lambda.\phi_1(B,K))[\uparrow^2]\ (\underline{2}\ \ \underline{1})\big), \quad \text{otherwise}
\end{aligned}$$

   and define function ϕ_2 as:

$$\begin{aligned}
\phi_2(K,K) &= \lambda.\lambda.\big(\underline{1}\ (\underline{2}\ \ \underline{3})\,(\underline{2}\ \ \underline{4})\big)\\
\phi_2(A\to B,K) &= \lambda.\lambda.\big(\underline{1}\ (\underline{4}\ \ \underline{2})\,((\lambda.\phi_1(A,K))[\uparrow^2]\ \ \underline{2})\big), \quad \text{if } K\in\mathcal{T}(A)\\
\phi_2(A\to B,K) &= \lambda.\big((\lambda.\phi_1(B,K))[\uparrow]\ (\underline{3}\ \ \underline{1})\big), \quad \text{otherwise}
\end{aligned}$$

   (a) Suppose $s(K)$ is not a type variable for $K \in \mathcal{T}(\tau)$
      i. Suppose $K \in \mathcal{T}(B)$. Let $b = \Big(\lambda.\big(\lambda.\underline{2}\ \ \lambda.((\lambda.\phi_2(B,K))[\uparrow]\ \lambda.\underline{2})\big)\ a\Big)$.
      ii. Suppose $K \in \mathcal{T}(\Gamma_i)$. Let $b = \big(\lambda.(a[\uparrow])\ \lambda.(\lambda.\lambda.\phi_2(\Gamma_i,K)\ \underline{i+1}\ \lambda.\underline{2})\big)$.
   (b) Suppose $s(K_1) = s(K_2) = L$ for distinct $K_1, K_2 \in \mathcal{T}(\tau)$
      i. Suppose $K_j{\in}\mathcal{T}(\Gamma_{i_j})$ for $j{\in}\{1,2\}$. Let $p_j{=}\big(\lambda.\phi_1(\Gamma_{i_j},K_j)\ \underline{i_j+1}\big)$ and $p{=}\lambda.\lambda.\big(\underline{1}\ p_1[\uparrow]\ p_2[\uparrow]\big)$. Let $b = \big(\lambda.\lambda.\underline{2}\ a\,p\big)$.
      ii. Suppose $K_1{\in}\mathcal{T}(\Gamma_i)$ and $K_2{\in}\mathcal{T}(B)$. Let $p = \lambda.\lambda.\big(\underline{1}\ \big((\lambda.\phi_1(\Gamma_i,K_1))[\uparrow]\ \underline{i+3}\big)\,\phi_2(B,K_2)[\uparrow]\big)$ and $b = \big(\lambda.(\lambda.\underline{2}\ \ p)\ a\big)$.
      iii. Suppose $K_i \in \mathcal{T}(B)$ for $i \in \{1,2\}$. Let $p = \lambda.\lambda.\big(\underline{1}\ \phi_2(B,K_1)[\uparrow]\ \phi_2(B,K_2)[\uparrow]\big)$ and let $b = \big(\lambda.(\lambda.\underline{2}\ \ p)\ a\big)$.

   Then, $b \in Terms_{TA_{\lambda\sigma}}(\tau') \setminus Terms_{TA_{\lambda\sigma}}(\tau)$. Thus, $\tau' \not\leq_{TA_{\lambda\sigma}} \tau$.

As consequence, if a typing τ' of some $\lambda\sigma$-term is not PT according to Definition 13, τ' is not PT according to Definition 9.
Let a be a $\lambda\sigma$-substitution t.

$\Rightarrow$ *proof:* Let $\tau = (\Gamma, \Delta)$ be a PT of t, according to Definition 13, and $\tau' = (\Gamma', \Delta')$ be a typing of t. By Definition 13, we have that exists a type substitution s such that $s(\Gamma) = \Gamma'_{\leq|\Gamma|}.nil$ and $s(\Delta) = \Delta'_{\leq|\Delta|}.nil$. By the property which says that if $TA_{\lambda\sigma} \rhd t : \tau$, then $TA_{\lambda\sigma} \rhd t : s(\tau) = (s(\Gamma), s(\Delta))$, for any type substitution s, we have $\tau \leq_{TA_{\lambda\sigma}} s(\tau)$. By ($\lambda\sigma$-sweak), we have that $s(\tau) \leq_{TA_{\lambda\sigma}} \tau'$. Thus, τ is PT of t, according to Definition 9.

$\Leftarrow$ *proof:* Let $\tau = (\Gamma, \Delta)$ be a PT of t, according to Definition 13, and $\tau' = (\Gamma', \Delta')$ be a typing of t which is not PT according to this definition. Then, exists a type substitution s such that $s(\Gamma) = \Gamma'_{\leq|\Gamma|}.nil$ and $s(\Delta) = \Delta'_{\leq|\Delta|}.nil$ and does not exist any substitution s' such that $s'(\Gamma') = \Gamma_{\leq|\Gamma'|}.nil$ and $s'(\Delta') = \Delta_{\leq|\Delta'|}.nil$.

1. Suppose $s(\Gamma) \neq \Gamma'$. Then, $m = |\Gamma| < |\Gamma'|$. Let $s_i = (\underline{1}.\underline{2}.\cdots.\underline{m+1}.\uparrow^{m+1})$ and $r = t \circ s_i$.
2. Otherwise, $s(\Gamma) = \Gamma'$. Let K be a type variable. Define the functions ϕ_1 and ϕ_2 as above.
   (a) Suppose $s(K)$ is not a type variable for $K \in \mathcal{T}(\tau)$
      i. Suppose $K \in \mathcal{T}(\Delta_i)$. Let $b = \Big(\lambda.\big(\lambda.\underline{2}\ \ \lambda.((\lambda.\phi_2(\Delta_i,K))[\uparrow]\ \lambda.\underline{2})\big)\ \ \underline{i}\Big)$ and s'_i be a $\lambda\sigma$-substitution such that $= (\underline{1}.\underline{2}.\cdots.\underline{i-1}.b.\uparrow^i)$. Let $r = s'_i \circ t$.

ii. Suppose $K \in \mathcal{T}(\Gamma_i)$. Let $b = \Big(\lambda.\big(\lambda.\underline{2}\ \ \lambda.((\lambda.\phi_2(\Gamma_i, K))[\uparrow]\ \lambda.\underline{2})\big)\ \ \underline{i}\Big)$ and let s'_i be as above. Let $r = t \circ s'_i$.

(b) Suppose $s(K_1) = s(K_2) = L$ for distinct $K_1, K_2 \in \mathcal{T}(\tau)$

i. Suppose $K_j \in \mathcal{T}(\Gamma_{i_j})$ for $j \in \{1,2\}$. Let $p_j = \big(\lambda.\phi_1(\Gamma_{i_j}, K_j)\ \ \underline{i_j + 1}\big)$ and $p = \lambda.\lambda.\big(\underline{1}\ \ p_1[\uparrow]\ p_2[\uparrow]\big)$. Let $b_j = \big(\lambda.\lambda.\underline{2}\ \ \underline{i_j}\ p\big)$, where j can be either 1 or 2 and let $s_{i_j} = (\underline{1}.\underline{2}.\cdots.\underline{i_j - 1}.b_j.\ \uparrow^{i_j})$. Let $r = t \circ s_{i_j}$.

ii. Suppose $K_j \in \mathcal{T}(\Delta_{i_j})$ for $j \in \{1,2\}$. Let $p_j = \big(\lambda.\phi_1(\Delta_{i_j}, K_j)\ \ \underline{i_j + 1}\big)$. Then, for p, b_j and s_{i_j} as above, let $r = s_{i_j} \circ t$.

iii. Suppose $K_1 \in \mathcal{T}(\Gamma_i)$ and $K_2 \in \mathcal{T}(\Delta_j)$. Let $b = \big(\lambda.(\lambda.\underline{2}\ \ p)\ \ \underline{j}\,[t]\big)$, where $p = \lambda.\lambda.\big(\underline{1}\ \big((\lambda.\phi_1(\Gamma_i, K_1))[\uparrow]\ \ \underline{i+3}\big)\ \phi_2(\Delta_j, K_2)[\uparrow]\big)$. Let $r = \big(\underline{1}\,[t].\underline{2}\,[t].\cdots.\underline{j-1}\,[t].b.(\uparrow^j \circ t)\big)$.

Then, $r \in Terms_{TA_{\lambda\sigma}}(\tau') \setminus Terms_{TA_{\lambda\sigma}}(\tau)$. Thus, $\tau' \not\leq_{TA_{\lambda\sigma}} \tau$

As consequence, if a typing τ' of some $\lambda\sigma$-substitution is not PT according to Definition 13, τ' is not PT according to Definition 9. □

An algorithm for type inference is presented, to verify if $\mathrm{TA}_{\lambda\sigma}$ has PT according to Definition 13. Thus, given an expression a, we will work with the decorated expression a' but the type for substitutions is a context as well. We use the same syntax for decorated expressions as in [5].

(Var)	$\langle R \cup \{\underline{1}_A^{\Gamma}\}, E\rangle$	$\to \langle R, E \cup \{\Gamma = A.\Gamma'\}\rangle$,where Γ' is a fresh context variable;
(Lambda)	$\langle R \cup \{(\lambda.a_{A_1}^{\Gamma_1})_{A_2}^{\Gamma_2}\}, E\rangle$	$\to \langle R, E \cup \{A_2 = A^* \to A_1, \Gamma_1 = A^*.\Gamma_2\}\rangle$, where A^* is a fresh type variable;
(App)	$\langle R \cup \{(a_{A_1}^{\Gamma_1}\ b_{A_2}^{\Gamma_2})_{A_3}^{\Gamma_3}\}, E\rangle$	$\to \langle R, E \cup \{\Gamma_1 = \Gamma_2, \Gamma_2 = \Gamma_3, A_1 = A_2 \to A_3\}\rangle$
(Clos)	$\langle R \cup \{(a_{A_1}^{\Gamma_1}[s_{\Gamma_3}^{\Gamma_2}])_{A_2}^{\Gamma_4}\}, E\rangle$	$\to \langle R, E \cup \{\Gamma_1 = \Gamma_3, \Gamma_2 = \Gamma_4, A_1 = A_2\}\rangle$
(Id)	$\langle R \cup \{id_{\Gamma_2}^{\Gamma_1}\}, E\rangle$	$\to \langle R, E \cup \{\Gamma_1 = \Gamma_2\}\rangle$
(Shift)	$\langle R \cup \{\uparrow_{\Gamma_2}^{\Gamma_1}\}, E\rangle$	$\to \langle R, E \cup \{\Gamma_1 = A'.\Gamma_2\}\rangle$,where A' is a fresh type variable;
(Cons)	$\langle R \cup \{(a_{A_1}^{\Gamma_1}.s_{\Gamma_3}^{\Gamma_2})_{\Gamma_5}^{\Gamma_4}\}, E\rangle$	$\to \langle R, E \cup \{\Gamma_1 = \Gamma_2, \Gamma_2 = \Gamma_4, \Gamma_5 = A_1.\Gamma_3\}\rangle$
(Comp)	$\langle R \cup \{(s_{\Gamma_2}^{\Gamma_1} \circ t_{\Gamma_4}^{\Gamma_3})_{\Gamma_6}^{\Gamma_5}\}, E\rangle$	$\to \langle R, E \cup \{\Gamma_1 = \Gamma_4, \Gamma_2 = \Gamma_6, \Gamma_3 = \Gamma_5\}\rangle$

Table 4. Type inference rules for the $\lambda\sigma$-calculus

The inference rules presented in the Table 4 are given according to the typing rules of the system $TA_{\lambda\sigma}$ presented in the Definition 12. The rules are applied to pairs $\langle R, E\rangle$, starting from the pair $\langle R_0, \emptyset\rangle$, as was done to $TA_{\lambda dB}$.

Example 4. For $a = (\underline{2}.id) \circ \uparrow$ one has $a' = (((\underline{1}_{A_1}^{\Gamma_1}\,[\uparrow_{\Gamma_3}^{\Gamma_2}])_{A_2}^{\Gamma_4}.id_{\Gamma_6}^{\Gamma_5})_{\Gamma_8}^{\Gamma_7} \circ \uparrow_{\Gamma_{10}}^{\Gamma_9})_{\Gamma_{12}}^{\Gamma_{11}}$. Then

$$R_0 = \{\underline{1}_{A_1}^{\Gamma_1}, \uparrow_{\Gamma_3}^{\Gamma_2}, id_{\Gamma_6}^{\Gamma_5}, \uparrow_{\Gamma_{10}}^{\Gamma_9}, (\underline{1}_{A_1}^{\Gamma_1}\,[\uparrow_{\Gamma_3}^{\Gamma_2}])_{A_2}^{\Gamma_4}, ((\underline{1}_{A_1}^{\Gamma_1}\,[\uparrow_{\Gamma_3}^{\Gamma_2}])_{A_2}^{\Gamma_4}.id_{\Gamma_6}^{\Gamma_5})_{\Gamma_8}^{\Gamma_7}, (((\underline{1}_{A_1}^{\Gamma_1}\,[\uparrow_{\Gamma_3}^{\Gamma_2}])_{A_2}^{\Gamma_4}.id_{\Gamma_6}^{\Gamma_5})_{\Gamma_8}^{\Gamma_7} \circ \uparrow_{\Gamma_{10}}^{\Gamma_9})_{\Gamma_{12}}^{\Gamma_{11}}\}$$

Applying the rules from Table 4 to the pair $\langle R_0, \emptyset\rangle$, until obtain the pair $\langle \emptyset, E_f\rangle$ and simplifying E_f, as in example 2, one obtains the set of equations $\{A_1 = A_2, \Gamma_{11} = \Gamma_{12} = A_2.\Gamma_2, \Gamma_2 = A'_1.\Gamma_1, \Gamma_1 = A_1.\Gamma'_1\}$

From this equational system one obtains the mgu $\Gamma_{11} = \Gamma_{12} = A_1.A'_1.A_1.\Gamma'_1$, for the variables of interest.

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

Proof. Let a be any $\lambda\sigma$-expression and a' its associated decorated expression. Let R_0 be the set of all sub-expression of a'. Starting with the pair $\langle R_0, \emptyset\rangle$ and applying the rules of the type inference algorithm in the Table 4 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-expressions of the pair is decremented. By the uniqueness in the decomposition

of the sub-expressions of the $\lambda\sigma$, one has that a unique rule can be applied to each sub-expression of R_0. Thus, the process finish with a pair $\langle \emptyset, E_f \rangle$, where E_f is a set of first-order equations over context and type variables, according to the rules of the type system $TA_{\lambda\sigma}$. An adequate first-order unification algorithm is then applied. And by the correctness, completeness and uniqueness of first-order unification, one has that the algorithm will find a mgu in the case that a is typable. Otherwise, the algorithm will report that there are no unifier. Consequently, the typing system $TA_{\lambda\sigma}$ satisfies PT. □

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

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

$$\textit{(Var)}\quad A.\Gamma \vdash \underline{1} : A \qquad \textit{(Varn)}\quad \frac{\Gamma \vdash \underline{n} : B}{A.\Gamma \vdash \underline{n+1} : B}$$

$$\textit{(Lambda)}\quad \frac{A.\Gamma \vdash b : B}{\Gamma \vdash \lambda.b : A \to B} \qquad \textit{(App)}\quad \frac{\Gamma \vdash a : A \to B \quad \Gamma \vdash b : A}{\Gamma \vdash (a\ b) : B}$$

$$\textit{(Sigma)}\quad \frac{\Gamma_{\geq i} \vdash b : B \quad \Gamma_{<i}.B.\Gamma_{\geq i} \vdash a : A}{\Gamma \vdash a\,\sigma^i b : A} \qquad \textit{(Phi)}\quad \frac{\Gamma_{\leq k}.\Gamma_{\geq k+i} \vdash a : A}{\Gamma \vdash \varphi^i_k\, a : A}$$

As for $\lambda\sigma$, the typed version of λs_e-calculus presented is in Curry style, which has the same properties of the version in Church style presented in [3].

Example 5. As in the Example 3, starting from the λ-term $(\lambda.\lambda.(\underline{1}\ \ \underline{2}))\,\underline{2}$ applying rules of λs_e one obtains the λs_e-term $\lambda.((\underline{1}\,\sigma^2\underline{2})\ (\varphi^2_0\,\underline{2}))$. The deduction of $B.C.nil \vdash_{TA_{\lambda s_e}} \lambda.((\underline{1}\,\sigma^2\underline{2})\ (\varphi^2_0\,\underline{2})) : (C \to A) \to A$ is presented. For brevity, contexts in a deduction are denoted without *nil*.

Initially, one has $\dfrac{C \vdash \underline{1} : C\,(\text{Var})}{B.C \vdash \underline{2} : C}(\text{Varn})$. Thus

$$\dfrac{\dfrac{\dfrac{B.C \vdash \underline{2} : C \quad C \to A.C.B.C \vdash \underline{1} : C \to A}{C \to A.B.C \vdash \underline{1}\,\sigma^2\underline{2} : C \to A}(\text{Sigma}) \quad \dfrac{B.C \vdash \underline{2} : C}{C \to A.B.C \vdash \varphi^2_0\,\underline{2} : C}(\text{Phi})}{C \to A.B.C \vdash ((\underline{1}\,\sigma^2\underline{2})\ (\varphi^2_0\,\underline{2})) : A}(\text{App})}{B.C \vdash \lambda.((\underline{1}\,\sigma^2\underline{2})\ (\varphi^2_0\,\underline{2})) : (C \to A) \to A}(\text{Lambda})$$

Lemma 3 (Weakening for λs_e). *Let a be a λs_e-term. If $\Gamma \vdash a : A$, then $\Gamma.B \vdash a : A$, for any type B.*

Proof. Induction on a structure.

1) $a = \underline{n}$: Let $\Gamma \vdash \underline{n} : A$. Since the type addition at the end of Γ do not change any free index type, one has trivially that $\Gamma.B \vdash \underline{n} : A$.
2) $a = (b\ c)$: Let $\Gamma \vdash (b\ c) : A$. By (App) one has that $\Gamma \vdash b : C \to A$ and $\Gamma \vdash c : C$, for some C. By IH one has $\Gamma.B \vdash b : C \to A$ and $\Gamma.B \vdash c : C$. Thus, by (App), $\Gamma.B \vdash (b\ c) : A$
3) $a = \lambda.b$: Let $\Gamma \vdash \lambda.b : A$. By $(Lambda)$ one has that $C.\Gamma \vdash b : D$, where $A = C \to D$. By IH one has $C.\Gamma.B \vdash b : D$. Thus, by $(Lambda)$, $\Gamma.B \vdash \lambda.b : A$
4) $a = b\,\sigma^i c$: Let $\Gamma \vdash b\,\sigma^i c : A$. By $(Sigma)$ one has that $\Gamma_{\geq i} \vdash c : C$ and $\Gamma_{<i}.C.\Gamma_{\geq i} \vdash b : A$. By IH one has $\Gamma_{\geq i}.B \vdash c : C$ and $\Gamma_{<i}.C.\Gamma_{\geq i}.B \vdash b : A$. Thus, by $(Sigma)$, $\Gamma.B \vdash b\,\sigma^i c : A$.
5) $a = \varphi^i_k b$: Let $\Gamma \vdash \varphi^i_k b : A$. By (Phi) one has that $\Gamma_{\leq k}.\Gamma_{\geq k+i} \vdash b : A$. By IH one has $\Gamma_{\leq k}.\Gamma_{\geq k+i}.B \vdash b : A$. Thus, by (Phi), $\Gamma.B \vdash \varphi^i_k b : A$. □

Proposition 3. *The rule $(\lambda s_e\text{-}weak)$ is admissible in System $TA_{\lambda s_e}$, where* $\dfrac{\Gamma \vdash a : A}{\Gamma.B \vdash a : A}(\lambda s_e\text{-}weak)$.

Definition 15 (Principal Typings in $TA_{\lambda s_e}$). *A **principal typing** of a term a in $TA_{\lambda s_e}$ is a typing $\tau = (\Gamma, B)$ such that*

1. *$TA_{\lambda s_e} \rhd a : \tau$*
2. *If $TA_{\lambda s_e} \rhd a : \tau'$ for any typing $\tau' = (\Gamma', B')$, then there exists a substitution s such that $s(\Gamma) = \Gamma'_{\leq |\Gamma|}.nil$ and $s(B) = B'$.*

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

Proof. $\Rightarrow$ *proof:* Let $\tau = (\Gamma, B)$ be a PT of some term a, according to Definition 15, and $\tau' = (\Gamma', B')$ be a typing of a. By Definition 15, we have that exists a type substitution s such that $s(\Gamma) = \Gamma'_{\leq|\Gamma|}.nil$ and $s(B) = B'$. By the property which says that if $TA_{\lambda s_e} \rhd a : \tau$, then $TA_{\lambda s_e} \rhd a : s(\tau)$, for any type substitution s, we have $\tau \leq_{TA_{\lambda s_e}} s(\tau)$. By ($\lambda s_e$-weak), we have that $s(\tau) \leq_{TA_{\lambda s_e}} \tau'$. Thus, τ is PT of a, according to Definition 9.

$\Leftarrow$ *proof:* Let $\tau = (\Gamma, B)$ be a PT of some term a, according to Definition 15, and $\tau' = (\Gamma', B')$ be a typing of a which is not PT according to this definition. Then, exists a type substitution s such that $s(\Gamma) = \Gamma'_{\leq|\Gamma|}.nil$ and $s(B) = B'$ and does not exist any substitution s' such that $s'(\Gamma') = \Gamma_{\leq|\Gamma'|}.nil$ and $s'(B') = B$.

1. Suppose $s(\Gamma) \neq \Gamma'$. Then, $m = |\Gamma| < |\Gamma'|$. Let $b = (\lambda.(\varphi_0^2\, a)\ \underline{m+1}\,)$.
2. Otherwise, $s(\Gamma) = \Gamma'$. Let K be a type variable. Define the function ϕ_1 as:

$$\begin{aligned}
\phi_1(K,K) &= \lambda.\lambda.\big(\underline{1}\ (\underline{2}\ \ \underline{4})\,(\underline{2}\ \ \underline{3})\big)\\
\phi_1(A\to B,K) &= \lambda.\lambda.\big(\underline{1}\ (\underline{3}\ \ \underline{2})\,(\varphi_0^4\,(\lambda.\phi_1(A,K))\ \ \underline{2}\,)\big), \quad \text{if } K\in\mathcal{T}(A)\\
\phi_1(A\to B,K) &= \lambda.\big(\varphi_0^3\,(\lambda.\phi_1(B,K))\ (\underline{2}\ \ \underline{1}\,)\big), \quad \text{otherwise}
\end{aligned}$$

and define function ϕ_2 as:

$$\begin{aligned}
\phi_2(K,K) &= \lambda.\lambda.\big(\underline{1}\ (\underline{2}\ \ \underline{3})\,(\underline{2}\ \ \underline{4})\big)\\
\phi_2(A\to B,K) &= \lambda.\lambda.\big(\underline{1}\ (\underline{4}\ \ \underline{2})\,(\varphi_0^3\,(\lambda.\phi_1(A,K))\ \ \underline{2}\,)\big), \quad \text{if } K\in\mathcal{T}(A)\\
\phi_2(A\to B,K) &= \lambda.\big(\varphi_0^2\,(\lambda.\phi_1(B,K))\ (\underline{3}\ \ \underline{1}\,)\big), \quad \text{otherwise}
\end{aligned}$$

(a) Suppose $s(K)$ is not a type variable for $K \in \mathcal{T}(\tau)$
   i. Suppose $K \in \mathcal{T}(B)$. Let $b = \Big(\lambda.\big(\lambda.\underline{2}\ \ \lambda.(\varphi_0^2\,(\lambda.\phi_2(B,K))\ \lambda.\underline{2}\,)\big)\ a\Big)$.
   ii. Suppose $K \in \mathcal{T}(\Gamma_i)$. Let $b = \big(\lambda.(\varphi_0^2\, a)\ \lambda.(\lambda.\lambda.\phi_2(\Gamma_i,K)\ \underline{i+1}\ \lambda.\underline{2}\,)\big)$.

(b) Suppose $s(K_1) = s(K_2) = L$ for distinct $K_1, K_2 \in \mathcal{T}(\tau)$
   i. Suppose $K_j \in \mathcal{T}(\Gamma_{i_j})$ for $j \in \{1,2\}$. Let $p_j = \big(\lambda.\phi_1(\Gamma_{i_j}, K_j)\ \ \underline{i_j+1}\,\big)$ and $p = \lambda.\lambda.\big(\underline{1}\ \ \varphi_0^2\, p_1\ \varphi_0^2\, p_2\big)$. Let $b = \big(\lambda.\lambda.\underline{2}\ a\, p\big)$.
   ii. Suppose $K_1 \in \mathcal{T}(\Gamma_i)$ and $K_2 \in \mathcal{T}(B)$. Let $p = \lambda.\lambda.\big(\underline{1}\ \big(\varphi_0^2\,(\lambda.\phi_1(\Gamma_i,K_1))\ \ \underline{i+3}\,\big)\,\varphi_0^2\,(\phi_2(B,K_2))\big)$ and $b = \big(\lambda.(\lambda.\underline{2}\ \ p)\ a\big)$.
   iii. Suppose $K_i \in \mathcal{T}(B)$ for $i \in \{1,2\}$. Let $p = \lambda.\lambda.\big(\underline{1}\ \varphi_0^2\,(\phi_2(B,K_1))\ \varphi_0^2\,(\phi_2(B,K_2))\big)$ and $b = \big(\lambda.(\lambda.\underline{2}\ \ p)\ a\big)$.

Then, $b \in Terms_{TA_{\lambda s_e}}(\tau') \setminus Terms_{TA_{\lambda s_e}}(\tau)$. Thus, $\tau' \not\leq_{TA_{\lambda s_e}} \tau$.

As consequence, if τ' is not PT according to Definition 15, τ' is not PT according to Definition 9. □

A type inference algorithm for the λs_e-calculus is presented, similarly to that of [4]. The decorated term associated with a, denoted as a', has syntax closer to the one of decorated λ-terms: any subterm is decorated with its type and its context variables.

Similarly to the previous algorithm, the rules of Table 5, developed according to the rules of Definition 14, are applied to pairs $\langle R, E\rangle$, where R is a set of decorated subterms of a' and E a set of equations over type and context variables.

Example 6. For the term $a = \lambda.((\underline{1}\,\sigma^2\underline{2}\,)\ (\varphi_0^2\,\underline{2}\,))$ one has $a' = (\lambda.((\underline{1}_{A_1}^{\Gamma_1}\,\sigma^2\underline{2}_{A_2}^{\Gamma_2}\,)_{A_3}^{\Gamma_3}\ (\varphi_0^2\,\underline{2}_{A_4}^{\Gamma_4}\,)_{A_5}^{\Gamma_5})_{A_6}^{\Gamma_6})_{A_7}^{\Gamma_7}$. Then

$$\begin{aligned}
R_0 = \big\{\ &\underline{1}_{A_1}^{\Gamma_1}\ ,\ \underline{2}_{A_2}^{\Gamma_2}\ ,\ (\underline{1}_{A_1}^{\Gamma_1}\,\sigma^2\underline{2}_{A_2}^{\Gamma_2}\,)_{A_3}^{\Gamma_3}\ ,\ \underline{2}_{A_4}^{\Gamma_4}\ ,\ (\varphi_0^2\,\underline{2}_{A_4}^{\Gamma_4}\,)_{A_5}^{\Gamma_5}\ ,\ ((\underline{1}_{A_1}^{\Gamma_1}\,\sigma^2\underline{2}_{A_2}^{\Gamma_2}\,)_{A_3}^{\Gamma_3}\ (\varphi_0^2\,\underline{2}_{A_4}^{\Gamma_4}\,)_{A_5}^{\Gamma_5})_{A_6}^{\Gamma_6},\\
&(\lambda.((\underline{1}_{A_1}^{\Gamma_1}\,\sigma^2\underline{2}_{A_2}^{\Gamma_2}\,)_{A_3}^{\Gamma_3}\ (\varphi_0^2\,\underline{2}_{A_4}^{\Gamma_4}\,)_{A_5}^{\Gamma_5})_{A_6}^{\Gamma_6})_{A_7}^{\Gamma_7}\ \big\}
\end{aligned}$$

Applying the rules in the Table 5 to the pair $\langle R_0, \emptyset\rangle$, until obtaining the pair $\langle \emptyset, E_f\rangle$, and simplifying E_f, similarly to the example 2, one obtains the system of equations $\big\{\, A_1 = A_4 \to A_6\ ,\ A_7 = A_1^* \to A_6\ ,\ A_1.\Gamma_1' = A_2'.A_2.\Gamma_2\ ,\ A_2'.\Gamma_2 = A_4'.A_3'.A_4.\Gamma_3' = A_1^*.\Gamma_7\ ,\ \Gamma_2 = A_1'.A_2.\Gamma_2' \,\big\}$ from which one has the mgu $A_7 = (A_2 \to A_6) \to A_6$ and $\Gamma_7 = A_1'.A_2.\Gamma_2'$ for variables of interest.

(Var)	$\langle R \cup \{\underline{1}_A^\Gamma\}, E\rangle$	$\rightarrow \langle R, E \cup \{\Gamma = A.\Gamma'\}\rangle$,where Γ' is a fresh context variable;
(Varn)	$\langle R \cup \{\underline{n}_A^\Gamma\}, E\rangle$	$\rightarrow \langle R, E \cup \{\Gamma = A'_1.\cdots.A'_{n-1}.A.\Gamma'\}\rangle$,where $A'_1, \ldots, A'_{n-1}$ and Γ' are fresh type and context variables;
(Lambda)	$\langle R \cup \{(\lambda.a_{A_1}^{\Gamma_1})_{A_2}^{\Gamma_2}\}, E\rangle$	$\rightarrow \langle R, E \cup \{A_2 = A^* \rightarrow A_1, \Gamma_1 = A^*.\Gamma_2\}\rangle$, where A^* is a fresh type variable;
(App)	$\langle R \cup \{(a_{A_1}^{\Gamma_1}\ b_{A_2}^{\Gamma_2})_{A_3}^{\Gamma_3}\}, E\rangle$	$\rightarrow \langle R, E \cup \{\Gamma_1 = \Gamma_2, \Gamma_2 = \Gamma_3, A_1 = A_2 \rightarrow A_3\}\rangle$
(Sigma)	$\langle R \cup \{(a_{A_1}^{\Gamma_1}\ \sigma^i b_{A_2}^{\Gamma_2})_{A_3}^{\Gamma_3}\}, E\rangle$	$\rightarrow \langle R, E \cup \{A_1 = A_3, \Gamma_1 = A'_1.\cdots.A'_{i-1}.A_2.\Gamma_2, \Gamma_3 = A'_1.\cdots.A'_{i-1}.\Gamma_2, \}\rangle$, where $A'_1, \ldots, A'_{i-1}$ are new type variables and the sequence is empty if $i = 1$;
(Phi)	$\langle R \cup \{(\varphi_k^i\ a_{A_1}^{\Gamma_1})_{A_2}^{\Gamma_2}\}, E\rangle$	$\rightarrow \langle R, E \cup \{A_1 = A_2, \Gamma_2 = A'_1.\cdots.A'_{k+i-1}.\Gamma', \Gamma_1 = A'_1.\cdots.A'_k.\Gamma'\}\rangle$, where Γ' and $A'_1, \ldots, A'_{k+i-1}$ are new context and type variables and if $k + i - 1, k = 0$ then the respectively sequences $A'_1, \ldots, A'_{k+i-1}$ and $A'_1, \ldots, A'_k$ are empty.

Table 5. Type inference rules for the λs_e-Calculus

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

Proof. Let a be any λs_e-term and a' its decorated version. Let R_0 be o set of decorated subterms of a'. Starting from the pair $\langle R_0, \emptyset\rangle$ and applying the type inference rules in the Table 5, one has that after each application of a rule the number of decorated subterms in R_0 decreases. Thus, the process terminates in a finite number of steps. By the uniqueness in decomposition of subterms of the λs_e by these rules, one has that a unique rule can be applied to each subterm of R_0. When the process finishes, one obtains a pair of the form $\langle \emptyset, E_f\rangle$, where E_f is a set of first-order equations over type and context variables, according to the typing rules of $TA_{\lambda s_e}$. By applying any adequate first-order unification algorithm to E_f, one could obtain a sole mgu in the case the term a is typable. Otherwise, a cannot be typable and the algorithm will report the inexistence of unifiers. Consequently, the typing system $TA_{\lambda s_e}$ satisfies PT. □

4 Conclusions and Future Work

We consider for $\lambda\sigma$ and λs_e particular notions of principal typings and presented respective definitions which are proved to agree with the system-independent notion introduced by Wells in [11]. 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. then the provided PT notion has to deal with the principality of substitutions as well. Then, the property of having principal typings is straightforwardly proved by revisiting type inference algorithms for the $\lambda\sigma$ and the λs_e, previously presented in [5] and [4], respectively. The result is based on the correctness, completeness and uniqueness of solutions given by adequate first-order unification algorithms.

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

References

1. M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit Substitutions. *J. of Functional Programming*, 1(4):375–416, 1991.
2. 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.
3. 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.
4. M. Ayala-Rincón and C. Muñoz. Explicit Substitutions and All That. *Revista Colombiana de Computación*, 1(1):47–71, 2000.

5. 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.
6. 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.
7. G. Dowek, T. Hardin, and C. Kirchner. Higher-order Unification via Explicit Substitutions. *Information and Computation*, 157(1/2):183–235, 2000.
8. J. R. Hindley. *Basic Simple Type Theory*. Number 42 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1997.
9. 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.
10. G. Nadathur and D. S. Wilson. A Notation for Lambda Terms A Generalization of Environments. *Theoretical Computer Science*, 198:49–98, 1998.
11. 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

A.1 Proofs of weakening and PT for $TA_{\lambda\sigma}$

Some auxiliar definitions and lemmas are necessary to prove Lemma 2.

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

$$\begin{aligned} \|(a\ b)\| &= \|a\| + \|b\| & \|\underline{1}\| &= 0 \\ \|\lambda.a\| &= \|a\| & \|id\| &= 0 \\ \|a[s]\| &= \|a\| + \|s\| & \|\uparrow\| &= 0 \\ \|s \circ t\| &= \|s\| + \|t\| & \|a.t\| &= 1 + \|a\| + \|t\| \end{aligned}$$

Lemma 4. *Let s be a $\lambda\sigma$-substitution such that $\|s\| = 0$. If $\Gamma \vdash s \rhd \Gamma'$, then $\Gamma.B \vdash s \rhd \Gamma'.B$*

Proof. Induction on s structure.

1) $s = id$: By (id) it has $\Gamma.B \vdash id \rhd \Gamma'.B$, trivially.
2) $s = \uparrow$: Let $\Gamma \vdash \uparrow \rhd \Gamma'$ where, by (shift), $\Gamma = A.\Gamma'$. Thus $\Gamma.B \vdash \uparrow \rhd \Gamma'.B$
3) $s = u \circ t$: Let $\Gamma \vdash u \circ t \rhd \Gamma'$. By (comp), it has that $\Gamma \vdash t \rhd \Gamma''$ and $\Gamma'' \vdash u \rhd \Gamma'$, for some Γ''. By induction hypothesis(IH) it has $\Gamma.B \vdash t \rhd \Gamma''.B$ and $\Gamma''.B \vdash u \rhd \Gamma'.B$. Thus, by (comp), $\Gamma.B \vdash u \circ t \rhd \Gamma'.B$. □

Lemma 5. *Let a be a $\lambda\sigma$-term such that $\|a\| = 0$. If $\Gamma \vdash a : A$, then $\Gamma.B \vdash a : A$.*

Proof. Induction on a structure.

1) $a = \underline{1}$: Let $\Gamma \vdash \underline{1} : A$. By (var) it has that $\Gamma = A.\Gamma'$, for some Γ'. Thus it has $\Gamma.B \vdash \underline{1} : A$, trivially.
2) $a = (b\ c)$: Let $\Gamma \vdash (b\ c) : A$. By (app) it has that $\Gamma \vdash b : C \to A$ and $\Gamma \vdash c : C$, for some C. By IH it has $\Gamma.B \vdash b : C \to A$ and $\Gamma.B \vdash c : C$. Thus, by (app), $\Gamma.B \vdash (b\ c) : A$
3) $a = \lambda.b$: Let $\Gamma \vdash \lambda.b : A$. By (lambda) it has that $C.\Gamma \vdash b : D$, where $A = C \to D$. By IH it has $C.\Gamma.B \vdash b : D$. Thus, by (lambda), $\Gamma.B \vdash \lambda.b : A$
4) $a = b[s]$: Let $\Gamma \vdash b[s] : A$. By (clos) is has that $\Gamma \vdash s \rhd \Gamma'$ and $\Gamma' \vdash b : A$, for some Γ'. It has $\|b[s]\| = \|b\| + \|s\| = 0$. Then, by Lemma 4, $\Gamma.B \vdash s \rhd \Gamma'.B$. By IH it has that $\Gamma'.B \vdash b : A$. Thus, by (clos), $\Gamma.B \vdash b[s] : A$. □

Proof (Lemma 2). Induction on a structure with subinduction on $\|\cdot\|$, having Lemmas 4 and 5 as induction base(IB).

1) $a = \underline{1}$: Let $\Gamma \vdash \underline{1} : A$. By (var) it has that $\Gamma = A.\Gamma'$, for some Γ'. Thus it has $\Gamma.B \vdash \underline{1} : A$, trivially.
2) $a = (b\ c)$: Let $\Gamma \vdash (b\ c) : A$. By (app) it has that $\Gamma \vdash b : C \to A$ and $\Gamma \vdash c : C$, for some C. By IH on structure it has $\Gamma.B \vdash b : C \to A$ and $\Gamma.B \vdash c : C$. Thus, by (app), $\Gamma.B \vdash (b\ c) : A$
3) $a = \lambda.b$: Let $\Gamma \vdash \lambda.b : A$. By (lambda) it has that $C.\Gamma \vdash b : D$, where $A = C \to D$. By IH on structure it has $C.\Gamma.B \vdash b : D$. Thus, by (lambda), $\Gamma.B \vdash \lambda.b : A$
4) $a = b[s]$: Let $\Gamma \vdash b[s] : A$. By (clos) is has that $\Gamma \vdash s \rhd \Gamma'$ and $\Gamma' \vdash b : A$, for some Γ'. By IH on structure it has $\Gamma'.B \vdash b : A$. Substitution s has to be examined. If $\|b\| > 0$, then by IH on $\|\cdot\|$, as $\|b[s]\| > \|s\|$, it has that $\Gamma.B \vdash s \rhd \Gamma'.B$.
   Else, if $\|b\| = 0$:
   - If $\|s\| = 0$, then Lemma 4 can be applied.
   - Otherwise, $s = c.t$ or $s = u \circ t$. If $s = c.t$, then by (cons) it has that $\Gamma \vdash c : C$ and $\Gamma \vdash t \rhd \Gamma''$, where $\Gamma' = C.\Gamma''$. As $\|c\|, \|t\| < \|s\| = \|b[s]\|$, by IH on $\|\cdot\|$ it has $\Gamma.B \vdash c : C$ and $\Gamma.B \vdash t \rhd \Gamma''.B$. Thus, by (cons), $\Gamma.B \vdash c.t \rhd \Gamma'.B$. If $s = u \circ t$, then by (comp) it has that $\Gamma \vdash t \rhd \Gamma''$ and $\Gamma'' \vdash u \rhd \Gamma'$, for some Γ''. If $\|u\|, \|t\| > 0$, the result holds by IH on $\|\cdot\|$. Otherwise, at least one of the substitutions has $\|\cdot\|$ greater than 0. Using induction on substitution s structure, where $\|s\| > 0$, the result holds. Then, it has that $\Gamma.B \vdash t \rhd \Gamma''.B$ and $\Gamma''.B \vdash u \rhd \Gamma'.B$. Thus, by (comp), $\Gamma.B \vdash u \circ t \rhd \Gamma'.B$.

   Finally, by (clos), it has that $\Gamma.B \vdash b[s] : A$. □