# 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 $\lambda$-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 $\lambda$-calculus and allow the explicit treatment of the substitution operation to formally correspond to its implementation. Several extensions of the $\lambda$-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 ($\lambda s_e$ 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 $\lambda s_e$- 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 $\lambda$-calculus and its real implementations. Since $\beta$ contraction depends on the definition of the substitution operations, which is informally given in the theory of $\lambda$-calculus, they are in fact made explicit, but obscurely developed (that is, in an empirical manner), when

⋆ Corresponding author, supported by a PhD scholarship of the CNPq.
⋆⋆ Partially supported by the CNPq.

most computational environments based on the $\lambda$-calculus are implemented. A remarkable exception is $\lambda$Prolog, 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], which 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 $\lambda$-terms are possible in this calculus, raising serious questions for any mechanism supposed to simulate the $\lambda$-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 $M$. For a discussion about the difference between principal type and principal typing see [Jim96]. In the simply typed $\lambda$-calculus this corresponds to the existence of *more representative* typings. PT guarantees compositional type inference and plays a crucial role in helping one to find a complete/terminating type inference algorithm.

In section 2 we present the type-free $\lambda$-calculus in de Bruijn notation, the $\lambda s_e$-calculus [KR97] and the $\lambda\sigma$-calculus [ACCL91]. In section 3 we present the relevant backgrounds for the type assignment systems we consider 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 $\lambda$-calculus in de Bruijn notation, the $\lambda\sigma$- and the $\lambda s_e$-calculi and prove that they are adequate. In section 4 we conclude and present future work. Detailed proofs and examples are included in an extended version of this work available at `www.mat.unb/~ayala/publications.html`.

## 2 The type free calculi

### 2.1 The $\lambda$-calculus in de Bruijn notation

**Definition 1 (Set $\Lambda_{dB}$).** *The syntax of $\lambda$-calculus in de Bruijn notation, **the $\lambda dB$-calculus**, is defined inductively as*

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

Terms like $((((M_1\ M_2)\ M_3) \dots)\ M_n)$ are written as usual $(M_1\, M_2\, \dots\, M_n)$. Let $M$ be a $\lambda$-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 to be **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 $M$.*

The $\beta$-contraction definition for $\Lambda_{dB}$ needs a mechanism which detects and updates free indices. Below, we give an operator similar to the one 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 $M$. For instance, $(\lambda.(\underline{1}\ \underline{3}))^+ = \lambda.(\underline{1}\ \underline{4})$. Using the i-lift, we are able to present the definition of the substitution used by $\beta$-contractions, as done in [ARKa2001a].

**Definition 4.** *Let $m, n \in \mathbb{N}^*$. The* ***$\beta$-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 $\beta$-contraction as defined in [ARKa2001a].

**Definition 5.** ***$\beta$-contraction*** *in $\lambda dB$ 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 $\lambda s_e$-Calculus

The $\lambda s_e$-calculus is a proper extension of the $\lambda dB$-calculus. Two operators $\sigma$ and $\varphi$ are introduced for substitution and updating, respectively, to control the atomization of the substitution operation by arithmetic constraints.

**Definition 6 (Set $\Lambda_s$ of $\lambda s_e$-terms).**

*The syntax of the $\lambda s_e$-calculus, where $n, i, j \in \mathbb{N}^*$ and $k \in \mathbb{N}$ is given by*

***Terms*** $M ::= \underline{n} \mid (M\ M) \mid \lambda.M \mid M\sigma^i M \mid \varphi_k^j M$

The term $M\sigma^i N$ represents the term $\{\underline{i}/N\}M$; i.e., the substitution of the free occurrences of $\underline{i}$ in $M$ by $N$, updating free variables in $M$ (and in $N$). The term $\varphi_k^j M$ 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 $\lambda s_e$-calculus together with the rule (Eta), as given in [ARKa2001a]. The bottom seven rules on table 1 are those which extend the $\lambda s$-calculus to $\lambda s_e$ ([KR97]) with the rule ($Eta$) ([ARKa2001a]). They ensure confluence of the $\lambda s_e$-calculus on open terms and the application to the higher order unification problem. Hence, those rules are not the focus of this paper.

$=_{s_e}$ denotes the equality for the associated substitution calculus, denoted as $s_e$, induced by all the rules except ($\sigma$-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 $\lambda\sigma$-expressions.

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

**Table 1.** The rewriting system of the $\lambda s_e$-calculus with the Eta rule

**Definition 7 (Set $\Lambda_\sigma$ of $\lambda\sigma$-expressions).** *The $\lambda\sigma$-expressions consist of:*
***Terms*** $M ::= \underline{1} \mid (M\ M) \mid \lambda.M \mid M[S]$
***Substitutions*** $S ::= id \mid\ \uparrow\ \mid M.S \mid S \circ S$

Substitutions can intuitively be thought of as 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 $\{\underline{1}/\underline{1}, \underline{2}/\underline{2}, \dots\}$ whereas $\uparrow$ is the substitution $\{\underline{i+1}/\underline{i} \mid i \in \mathbb{N}^*\}$. 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 $\{M/\underline{1}, S(i)/\underline{i+1}\}$ and is called the **cons of** $M$ **in** $S$. $M[N.id]$ starts the simulation of the $\beta$-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].

| | | | | | | |
|---|---|---|---|---|---|---|
| $(\lambda.M\ N)$ | $\longrightarrow$ | $M[N.id]$ | ($Beta$) | $(\lambda.M)[S]$ | $\longrightarrow$ | $\lambda.(M[1.(S\circ\uparrow)])$ ($Abs$) |
| $(M\ N)[S]$ | $\longrightarrow$ | $(M[S]\ N[S])$ | ($App$) | $\uparrow \circ (M.S)$ | $\longrightarrow$ | $S$ ($ShiftCons$) |
| $M[id]$ | $\longrightarrow$ | $M$ | ($Id$) | $(S_1 \circ S_2) \circ S_3$ | $\longrightarrow$ | $S_1 \circ (S_2 \circ S_3)$ ($AssEnv$) |
| $1[S].(\uparrow \circ S)$ | $\longrightarrow$ | $S$ | ($Scons$) | $(M.S) \circ T$ | $\longrightarrow$ | $M[T].(S \circ T)$ ($MapEnv$) |
| $(M[S])[T]$ | $\longrightarrow$ | $M[S \circ T]$ | ($Clos$) | $1.\uparrow$ | $\longrightarrow$ | $id$ ($VarShift$) |
| $id \circ S$ | $\longrightarrow$ | $S$ | ($IdL$) | $1[M.S]$ | $\longrightarrow$ | $M$ ($VarCons$) |
| $S \circ id$ | $\longrightarrow$ | $S$ | ($IdR$) | $\lambda.(M\ \underline{1})$ | $\longrightarrow$ | $N$ 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 by $\sigma$, 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*** $\tau ::= \alpha \mid \tau \to \tau$ ***Contexts*** $A ::= nil \mid \tau.A$

*where* $\alpha$ *ranges over* **type variables**.

A **type assignment system** $\mathcal{S}$ is a set of rules, allowing some terms of a given system to 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 $\lambda$-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 $\tau$ in context $A$, and $\langle A, \tau\rangle$ is called a **typing of** $M$. If $\Theta = \langle A, \tau\rangle$ is a typing in $\mathcal{S}$ then $\mathcal{S} \Vdash M : \Theta$ denotes that $\Theta$ is a typing of $M$ in $\mathcal{S}$.

The contexts for $\lambda$-terms in de Bruijn notation are sequences of types. If $A$ is 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 $\tau$ 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 $\Theta$ 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* $\Theta$ *in system* $\mathcal{S}$ *is principal for some term* $M$ *if* $\mathcal{S} \Vdash M : \Theta$ *and for any* $\Theta'$ *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 λ-calculus in de Bruijn notation $TA_{\lambda dB}$

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

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

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

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

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

Lemma 1 above is proved via the statement of a more general property of $TA_{\lambda dB}$, which justifies why the weakening for this type system has to be done only adding types at the end of contexts.

Using ($\lambda dB$-weak) and type substitution, we follow the definition of [We2002] for Hindley's Principal Typing to define principal typing for the $\lambda dB$-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 $\lambda$-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 $\lambda dB$ may have 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 $\lambda$-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 $\Theta$ is principal in $TA_{\lambda dB}$ according to Definition 11 iff $\Theta$ is principal in $TA_{\lambda dB}$ according to Definition 9.*

The proof is similar to the one in [We2002]. The 'sufficient' condition uses a substitution lemma as in [Hi97] 3A2.1(ii) and the weakening from Lemma 1. The 'necessary' condition is constructive by contraposition building a counter example: given a term $M$ with PT $\Theta$ one supposes a typing $\Theta'$ that is not PT of $M$ according to definition 11. From $M$ and the relation between $\Theta$ and $\Theta'$ given by definition 11, one builds a new term $N$ for which $\Theta'$ is a typing, but $\Theta$ is not. The main difference between the proof in [We2002] and this one is the recursive function used to give $N$ a structure exploring some specific $\Theta'$ feature, which has to be split, according to the order in which the term is bound during the recursive construction of the counter example.

We now present a type inference algorithm for $\lambda dB$-terms, similarly to the one in [AyMu2000] for $\lambda s_e$, 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 by $M'$. 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.

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

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

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.

*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:

$\langle\!\langle R_0, \emptyset\rangle\!\rangle \rightarrow_{\text{Varn}}$

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

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

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

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

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 $\{\tau_1 = \tau_2 \rightarrow \tau_3, \tau_4 = \tau_1^* \rightarrow \tau_3, A_1 = \tau'_1.\tau_1.A'_1, A_1 = \tau_2.A'_2, A_1 = \tau_1^*.A_4\}$. Thus, simplifying one gets $\{\tau_1 = \tau_2 \rightarrow \tau_3, \tau_4 = \tau_1^* \rightarrow \tau_3, \tau'_1.\tau_1.A'_1 = \tau_2.A'_2 = \tau_1^*.A_4\}$. From these equations one gets the most general unifier (mgu for short) $\tau_4 = \tau_2 \rightarrow \tau_3$ and $A_4 = (\tau_2 \rightarrow \tau_3).A'_1$, for the variables of interest. Since the context must be the shortest one, $A'_1 = nil$ and $\langle(\tau_2 \rightarrow \tau_3).nil, \tau_2 \rightarrow \tau_3\rangle$ 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 $\lambda s_e$

The typed version of $\lambda s_e$ presented is in Curry style, which we have verified to have the same 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 $\lambda s_e$-terms and the rules are the same as the untyped version.

Since the syntax of $\lambda s_e$ remains close to the $\lambda dB$-calculus, to have a type assignment system for the $\lambda s_e$-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.*

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

*where in (Sigma) $|A| \geq i-1$ and in (Phi) $|A| \geq k+i-1$.*

Weakening for $\lambda s_e$ is done in the same way as for $\lambda dB$, adding types at the end of a context, giving the following lemma.

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

Consequently, the definition of principal typings in $\lambda s_e$ 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 $\Theta$ is principal in $TA_{\lambda s_e}$ according to Definition 13 iff $\Theta$ is principal in $TA_{\lambda s_e}$ according to Definition 9.*

The proof of Theorem 3 is a straightforward extension of that of Theorem 1.

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

| | |
|---|---|
| (Sigma) | $\langle\!\langle R \cup \{(M^{A_1}_{\tau_1}\sigma^i N^{A_2}_{\tau_2})^{A_3}_{\tau_3}\}, E\rangle\!\rangle \rightarrow$ $\langle\!\langle R, E \cup \{\tau_1{=}\tau_3, A_1{=}\tau'_1.\cdots.\tau'_{i-1}.\tau_2.A_2, A_3{=}\tau'_1.\cdots.\tau'_{i-1}.A_2\}\rangle\!\rangle$, where $\tau'_1,\ldots,\tau'_{i-1}$ are new type variables and the sequence is empty if $i=1$; |
| (Phi) | $\langle\!\langle R \cup \{(\varphi^i_k M^{A_1}_{\tau_1})^{A_2}_{\tau_2}\}, E\rangle\!\rangle \rightarrow$ $\langle\!\langle R, E \cup \{\tau_1 = \tau_2, A_2 = \tau'_1.\cdots.\tau'_{k+i-1}.A', A_1 = \tau'_1.\cdots.\tau'_k.A'\}\rangle\!\rangle$, where $A'$ and $\tau'_1,\ldots,\tau'_{k+i-1}$ are new variables of context and type. If $k+i-1=0$ or $k=0$, 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 $\lambda s_e$-Calculus

Similarly to the previous algorithm, the rules of Table 4 were developed according to the rules of Definition 12. The decorated term associated with $M$, denoted by $M'$, has a syntax close to that of decorated $\lambda dB$-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 $M = \lambda.((\underline{1}\,\sigma^2\underline{2}\,)\,(\varphi^2_0\,\underline{2}\,))$, one obtains the corresponding $R_0$ from $M' = (\lambda.((\underline{1}^{A_1}_{\tau_1}\sigma^2\underline{2}^{A_2}_{\tau_2})^{A_3}_{\tau_3}\,(\varphi^2_0\,\underline{2}^{A_4}_{\tau_4})^{A_5}_{\tau_5})^{A_6}_{\tau_6})^{A_7}_{\tau_7}$. 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 $E_f$, in a similar fashion to example 1, one obtains the system of equations which lead to the mgu $\tau_7 = (\tau_2 \rightarrow \tau_6) \rightarrow \tau_6$ and $A_7 = \tau'_1.\tau_2.A'_2$ for the 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. $S \rhd A$ 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.*

$$\textit{(var)}\quad \tau.A \vdash \underline{1} : \tau \qquad \textit{(lambda)}\quad \frac{\sigma.A \vdash M : \tau}{A \vdash \lambda.M : \sigma \to \tau}$$

$$\textit{(app)}\quad \frac{A \vdash M : \sigma \to \tau \quad A \vdash N : \sigma}{A \vdash (M\ N) : \tau} \qquad \textit{(clos)}\quad \frac{A \vdash S \rhd A' \qquad A' \vdash M : \tau}{A \vdash M[S] : \tau}$$

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

$$\textit{(cons)}\quad \frac{A \vdash M : \tau \qquad A \vdash S \rhd A'}{A \vdash M.S \rhd \tau.A'} \qquad \textit{(comp)}\quad \frac{A \vdash S \rhd A'' \quad A'' \vdash S' \rhd A'}{A \vdash S' \circ S \rhd A'}$$

Observe that the name of the typing rules begin with lower-case letters, while the rewriting rules with upper-case letters. As for $\lambda s_e$, 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].

The notion of typing for $TA_{\lambda\sigma}$ has to be adapted because 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 $\lambda$-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 $\sigma$. Similarly, if $A \vdash S \rhd A'$, then $A.\sigma \vdash S \rhd A'.\sigma$. Hence, the rules ($\lambda\sigma$-tweak) and ($\lambda\sigma$-sweak) hold in System $TA_{\lambda\sigma}$, where*

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

Lemma 3 and type substitutions 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 $\Theta$ is principal in $TA_{\lambda\sigma}$ according to Definition 15 iff $\Theta$ is principal in $TA_{\lambda\sigma}$ according to Definition 9.*

Despite the fact that the notion of typing is extended to include the sort substitution, the techniques used to prove Theorem 5 are the same applied to prove Theorems 1 and 3.

| | | |
|---|---|---|
| (Var) | $\langle\!\langle R \cup \{\underline{1}^{A}_{\tau}\}, E\rangle\!\rangle$ | $\rightarrow \langle\!\langle R, E \cup \{A = \tau.A'\}\rangle\!\rangle$,where $A'$ is a fresh context variable; |
| (Lambda) | $\langle\!\langle R \cup \{(\lambda.M^{A_1}_{\tau_1})^{A_2}_{\tau_2}\}, E\rangle\!\rangle$ | $\rightarrow \langle\!\langle R, E \cup \{\tau_2 = \tau^* \rightarrow \tau_1, A_1 = \tau^*.A_2\}\rangle\!\rangle$, where $\tau^*$ is a fresh type variable; |
| (App) | $\langle\!\langle R \cup \{(M^{A_1}_{\tau_1}\ N^{A_2}_{\tau_2})^{A_3}_{\tau_3}\}, E\rangle\!\rangle$ | $\rightarrow \langle\!\langle R, E \cup \{A_1 = A_2, A_2 = A_3, \tau_1 = \tau_2 \rightarrow \tau_3\}\rangle\!\rangle$ |
| (Clos) | $\langle\!\langle R \cup \{(M^{A_1}_{\tau_1}[S^{A_2}_{A_3}])^{A_4}_{\tau_2}\}, E\rangle\!\rangle$ | $\rightarrow \langle\!\langle R, E \cup \{A_1 = A_3, A_2 = A_4, \tau_1 = \tau_2\}\rangle\!\rangle$ |
| (Id) | $\langle\!\langle R \cup \{id^{A_1}_{A_2}\}, E\rangle\!\rangle$ | $\rightarrow \langle\!\langle R, E \cup \{A_1 = A_2\}\rangle\!\rangle$ |
| (Shift) | $\langle\!\langle R \cup \{\uparrow^{A_1}_{A_2}\}, E\rangle\!\rangle$ | $\rightarrow \langle\!\langle R, E \cup \{A_1 = \tau'.A_2\}\rangle\!\rangle$,where $\tau'$ is a fresh type variable; |
| (Cons) | $\langle\!\langle R \cup \{(M^{A_1}_{\tau_1}.S^{A_2}_{A_3})^{A_4}_{A_5}\}, E\rangle\!\rangle$ | $\rightarrow \langle\!\langle R, E \cup \{A_1 = A_2, A_2 = A_4, A_5 = \tau_1.A_3\}\rangle\!\rangle$ |
| (Comp) | $\langle\!\langle R \cup \{(S^{A_1}_{A_2} \circ T^{A_3}_{A_4})^{A_5}_{A_6}\}, E\rangle\!\rangle$ | $\rightarrow \langle\!\langle R, E \cup \{A_1 = A_4, A_2 = A_6, A_3 = A_5\}\rangle\!\rangle$ |

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

We now present an algorithm for type inference, to verify if $\mathrm{TA}_{\lambda\sigma}$ 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].

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 $M' = (((\underline{1}^{A_1}_{\tau_1}[\uparrow^{A_2}_{A_3}])^{A_4}_{\tau_2}.id^{A_5}_{A_6})^{A_7}_{A_8} \circ \uparrow^{A_9}_{A_{10}})^{A_{11}}_{A_{12}}$. Then $R_0 = \{(\underline{1}^{A_1}_{\tau_1}[\uparrow^{A_2}_{A_3}])^{A_4}_{\tau_2}, ((\underline{1}^{A_1}_{\tau_1}[\uparrow^{A_2}_{A_3}])^{A_4}_{\tau_2}.id^{A_5}_{A_6})^{A_7}_{A_8}, (((\underline{1}^{A_1}_{\tau_1}[\uparrow^{A_2}_{A_3}])^{A_4}_{\tau_2}.id^{A_5}_{A_6})^{A_7}_{A_8} \circ \uparrow^{A_9}_{A_{10}})^{A_{11}}_{A_{12}}$, $\underline{1}^{A_1}_{\tau_1}, \uparrow^{A_2}_{A_3}, id^{A_5}_{A_6}, \uparrow^{A_9}_{A_{10}}\}$. Applying the rules from Table 5 to the pair $\langle\!\langle R_0, \emptyset\rangle\!\rangle$ until the pair $\langle\!\langle \emptyset, E_f\rangle\!\rangle$ is reached, 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 $\lambda s_e$ 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 $\lambda$-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 $\lambda s_e$ 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 $\lambda s_e$-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.

[Jim96] , T. Jim. What are principal typings and what are they good for?. In *Proc. of POPL'95: Symp. on Principles of Programming Languages*, 42–53, ACM, 1996.

[KR97] F. Kamareddine and A. Ríos. Extending a $\lambda$-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 $\lambda$-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.