STAR TYPES: A TYPE SYSTEM FOR PATTERN CALCULUS

Besik Dundua[1,2] Mikheil Rukhaia[2] Lali Tibua[2]

[1] FBT, International Black Sea University
Agmashenebeli Alley 13km., 0131 Tbilisi, Georgia
[2] VIAM, Ivane Javakhishvili Tbilisi State University
University str. 2, 0186 Tbilisi, Georgia

Abstract

The pattern calculus described in this paper integrates the functional mechanism of the lambda-calculus and the capabilities of pattern matching with star types. Such types specify finite sequences of terms and introduce non-determinism, caused by finitary matching. We parametrize the calculus with an abstract matching function and prove that for each concrete instance of the function with a finitary matching, the calculus enjoys subject reduction property.

1 Introduction

Pattern calculi extend λ-calculus by permitting to abstract over arbitrary terms, not only over variables. For instance, an expression $\lambda_{\{x\}}(f\,x).(x\,a)$ is a term in a pattern calculus, where $(f\,x)$ is called the pattern. Some of other instances of pattern calculi are λ-calculus with patterns [15], pure pattern calculus [12, 13], pattern-based calculi with finitary matching [1], ρ-calculus [5], λ-calculus with first-order constructor patterns [16].

Pattern calculi are expressive enough to encode term rewriting systems correspond to contracting symbols given in [18, 17]. Typed pattern based calculi are formalism for functional programming languages. Therefore, after studying properties of untyped pattern calculus parameterized with finitary matching [1], it is natural to integrate a type system in it and consider a typed version of the calculus. This paper presents star typed pattern calculus with finitary matching and introduces a corresponding subtyping relation. Star types allow to control non-determinism in pattern calculus influenced from finitary matching. We prove that star typed pattern calculus with finitary matching enjoys subject reduction property.

A distinctive feature of our pattern calculus is star types, extending simple types. For instance, we may have a type α^*, which, intuitively, represents a finite sequence of terms (aka a hedge) of type α; or a type

$\alpha_1 \to \alpha_2^* \to \alpha$, which represents a variadic function whose first argument should have the type α_1, followed by arbitrarily many arguments of type α_2. Star types naturally introduce subtyping, based on the monoidal structure of hedges. Fixed arity types (e.g., the binary type $(\theta_1^* \to \alpha_1) \to \theta_2 \to \alpha_2$) and their starred versions form the upper set in the preorder generated by the subtype ordering. We type variables only with types from this set, while constants can get arbitrary types, which may contain stars (e.g., $(\theta^* \to \alpha)^* \to \alpha$) but are not starred themselves (e.g., not $(\theta^* \to \alpha)^*$).

Typed pattern calculi have been studied in [14, 6, 3, 11], just to name a few. To the best of our knowledge, systems with star types have not been considered in this context. Star types (and, in general, regular expression types) proved to be useful for XML processing. XDuce [10], CDuce [4], XHaskell [19], XCentric [7], PρLog [9, 8] are some examples of such applications. Hence, our work also provides a bridge between pattern-calculi (which model pattern-matching in functional programming languages) and XML-processing languages.

2 A Non-deterministic Pattern Calculus

2.1 Untyped Terms

We start with defining the syntax of our untyped pattern calculus. The alphabet consists of the set $\mathcal{X}$ of variables and $\mathcal{F}$ of constants. They are disjoint and countably infinite. The symbols x and f range over $\mathcal{X}$ and $\mathcal{F}$, respectively. *Terms* are defined by the following grammar:

$$A, B ::= x \mid f \mid (A\,B) \mid \lambda_{\mathcal{V}} A.B$$

where $(A\,B)$ is an *application* and $\lambda_{\mathcal{V}} A.B$ is an *abstraction*. We call the term A in the abstraction a *pattern* and the finite set $\mathcal{V}$ of variables is supposed to specify which variables are bound by the abstraction. Application associates to the left, therefore we can write $(A\,B_1 \cdots B_n)$ for $((\cdots(A\,B_1)\cdots)B_n)$. When there is no ambiguity, the outermost parentheses are omitted as well. The letters A, B, C, D are used for terms and the set of terms is denoted by $\mathcal{T}$.

The *sets of free and bound variables* of a term D, denoted $\mathtt{fv}(D)$ and $\mathtt{bv}(D)$ respectively, are defined inductively as follows:

$$\begin{array}{ll}
\mathtt{fv}(x) = \{x\} \qquad \mathtt{fv}(f) = \varnothing & \mathtt{bv}(x) = \varnothing \qquad \mathtt{bv}(f) = \varnothing \\
\mathtt{fv}(A\,B) = \mathtt{fv}(A) \cup \mathtt{fv}(B) & \mathtt{bv}(A\,B) = \mathtt{bv}(A) \cup \mathtt{bv}(B) \\
\mathtt{fv}(\lambda_{\mathcal{V}} A.B) = (\mathtt{fv}(A) \cup \mathtt{fv}(B)) \backslash \mathcal{V} & \mathtt{bv}(\lambda_{\mathcal{V}} A.B) = \mathtt{bv}(A) \cup \mathtt{bv}(B) \cup \mathcal{V}
\end{array}$$

Note that, unlike the λ-calculus, we abstract not only on variables but on terms. Usually, terms are denoted by capital letters. We adopt Barendregt's variable name convention [2], i.e., free and bound variables have different names. This can be fulfilled by renaming bound variables. As usual, we identify terms modulo α-equivalence.

Hedges are finite (possible empty) sequences of terms. For readability, we put in brackets if they have more than one element, e.g., $\langle A, B\rangle$. For the empty hedge we use $\langle\rangle$. We use letter h to denote hedge.

The notions of free and bound variables are extend to hedges in a natural way: $\mathtt{fv}(\langle A_1, \ldots, A_n\rangle) = \cup_{i=1}^{n}\mathtt{fv}(A_i)$ and $\mathtt{bv}(\langle A_1, \ldots, A_n\rangle) = \cup_{i=1}^{n}\mathtt{bv}(A_i)$.

A *substitution* σ is a mapping from variables to hedges such that all but finitely many variables are mapped to themselves. If $x_1, \ldots, x_n$, $n \geqslant 0$ are all the variables for which $\sigma(x_i) \neq x_i$, then we write σ in the form of the finite set of pairs $\{x_1 \mapsto \sigma(x_1), \ldots, x_n \mapsto \sigma(x_n)\}$. The sets $Dom(\sigma) = \{x_1, \ldots, x_n\}$ and $Ran(\sigma) = \{\sigma(x_1), \ldots, \sigma(x_n)\}$ are called the *domain* and the *range* of σ, respectively. The set $Var(\sigma)$ is defined as $Var(\sigma) = Dom(\sigma) \cup \mathtt{fv}(Ran(\sigma))$. The Greek letters $\sigma, \rho, \varphi, \vartheta$ are used to denote substitutions.

The *restriction* of a substitution σ to a set of variables V, denoted $\sigma|_V$, is defined as $\sigma|_V(x) = \sigma(x)$ if $x \in V$, and $\sigma|_V(x) = x$ otherwise.

The *application* of a substitution φ to a term D replaces each *free* occurrence of a variable v in D with $\varphi(v)$. It is defined inductively:

$$
\begin{array}{ll}
x\sigma = \sigma(x), \text{ if } x \in Dom(\sigma). & (A\,B)\sigma = A\sigma\,B\sigma, \text{ if } B \notin \mathcal{X}. \\
x\sigma = x, \text{ if } x \notin Dom(\sigma). & (A\,x)\sigma = A\sigma\,B_1 \cdots B_n, \\
f\sigma = f. & \qquad \text{if } \sigma(x) = \langle B_1, \ldots, B_n\rangle, n \geqslant 0. \\
(\lambda_{\mathcal{V}} A.B)\sigma = \lambda_{\mathcal{V}} A\sigma.B\sigma. & (A\,x)\sigma = A\sigma\,x, \text{ if } x \notin Dom(\sigma).
\end{array}
$$

In the abstraction, it is assumed that $Var(\varphi) \cap \mathtt{bv}(\lambda_{\mathcal{V}} A.B) = \varnothing$. This can be achieved by properly renaming the bound variables. Hence, the equality here is α-equivalence.

The application of a substitution φ to a hedge $\langle s_1, \ldots, s_n\rangle$ is defined as $\langle s_1, \ldots, s_n\rangle\varphi = \langle s_1\varphi, \ldots, s_n\varphi\rangle$, where we write $\langle s_1\varphi, \ldots, s_{i-1}\varphi, t_1, \ldots, t_m, s_{i+1}\varphi, \ldots, s_n\varphi\rangle$ when $s_i\varphi = \langle t_1, \ldots, t_m\rangle$.

Evaluation in the pattern calculus is defined by a binary relation β_{p} on terms. It defines the way how pattern-abstractions are applied. The relation is parametrized by a *pattern matching function* $\mathcal{S}ol$, which takes as parameters two terms A, B and set of variables $\mathcal{V}$ and computes a finite set of substitutions. We denote it by $\mathcal{S}ol(A \lessdot_{\mathcal{V}} B)$. β_{p} is written in the form of a reduction rule:

$$\beta_{\mathsf{p}}: \quad (\lambda_{\mathcal{V}} A.B)\,C \to B\sigma, \text{ where } \sigma \in \mathcal{S}ol(A \lessdot_{\mathcal{V}} C) \text{ and } B\sigma \text{ is a term.}$$

The condition "$B\sigma$ is a term" is important to make sure that terms are reduced to terms, not to arbitrary hedges. A reducible expression, or *redex,* is any expression to which this rule applies. A binary relation of compatibility $\to_R$ on hedges is defined with the help of the following inference rules:

$$\frac{A \to_R A'}{AB \to_R A'B} \qquad \frac{A \to_R A'}{BA \to_R BA'} \qquad \frac{B \to_R B'}{\lambda_{\mathcal{V}} A.B \to_R \lambda_{\mathcal{V}} A.B'}$$

$$\frac{A \to_R A'}{\lambda_{\mathcal{V}} A.B \to_R \lambda_{\mathcal{V}} A'.B} \qquad \frac{A \to_R B}{\langle h_1, A, h_2\rangle \to_R \langle h_1, B, h_2\rangle}$$

In what follows, $\to_{\beta_C}$ denotes the compatible closure of the β_{p} relation and $\twoheadrightarrow_{\beta_C}$ denotes the reflexive and transitive closure of $\to_{\beta_C}$. The definition of $\twoheadrightarrow_{\beta_C}$ is extended to substitutions having the same domain by setting $\varphi \twoheadrightarrow_{\beta_C} \varphi'$ if for all $x \in Dom(\varphi) = Dom(\varphi')$, we have $x\varphi \twoheadrightarrow_{\beta_C} x\varphi'$.

2.1.1 Typed Terms.

Let $\mathbb{A}$ be a nonempty set of type atoms. The set of *types* over $\mathbb{A}$, denoted $\mathbb{T}_{\mathbb{A}}$ or simply $\mathbb{T}$, is defined inductively with the help of the type constructors $\to$ and $*$: $\alpha \in \mathbb{A} \Rightarrow \alpha \in \mathbb{T}$, $\theta_1, \theta_2 \in \mathbb{T} \Rightarrow (\theta_1 \to \theta_2) \in \mathbb{T}$, and $\theta_1, \theta_2 \in \mathbb{T} \Rightarrow (\theta_1^* \to \theta_2) \in \mathbb{T}$.

The set of *star types* (over $\mathbb{A}$), denoted $\mathbb{T}^{\star}_{\mathbb{A}}$ or simply $\mathbb{T}^*$, is defined inductively as $\theta \in \mathbb{T} \Rightarrow \theta \in \mathbb{T}^*$ and $\theta \in \mathbb{T} \Rightarrow \theta^* \in \mathbb{T}^*$. Note that if a type does not contain a star type, then it is a standard simple type.

We define yet another set of types, which we call *fixed arity types* and denote by $\mathbb{F}$. It is the smallest set with the properties $\alpha \in \mathbb{A} \Rightarrow \alpha \in \mathbb{F}$ and $\theta \in \mathbb{T}, \varphi \in \mathbb{F} \Rightarrow (\theta \to \varphi) \in \mathbb{F}$. Each type in $\mathbb{F}$ has the form $\theta_1 \to (\cdots \to (\theta_n \to \alpha)\cdots)$. The set of *starred fixed arity types* is denoted by $\mathbb{F}^*$. We have $\mathbb{F} \subset \mathbb{T}$ and $\mathbb{F}^* \subset \mathbb{T}^*$.

The letter α will be used to denote elements from $\mathbb{A}$, the letters θ, δ for elements of $\mathbb{T}$, Θ for elements of $\mathbb{T}^*$, φ for elements of $\mathbb{F}$, and Φ for elements of $\mathbb{F}^*$. As usual, $\Theta_1 \to \cdots \to \Theta_n \to \theta$ stands for $(\Theta_1 \to (\Theta_2 \to \cdots \to (\Theta_n \to \theta)\cdots))$.

The *subtyping relation* is the preorder generated by the relation $\leqslant$ defined as:

$$\theta_1^* \to \theta_2 \leqslant \theta_2 \qquad \theta_1^* \to \theta_2 \leqslant \theta_1^* \to \theta_1^* \to \theta_2 \qquad \theta \leqslant \theta^*$$

$$\theta_1^* \leqslant \theta_2^* \text{ if } \theta_1 \leqslant \theta_2 \qquad \Theta_1 \to \theta_1 \leqslant \Theta_2 \to \theta_2 \text{ if } \Theta_2 \leqslant \Theta_1 \text{ and } \theta_1 \leqslant \theta_2$$

We denote the subtyping relation with $\leqslant$ as well. The following lemma characterizes fixed arity types in $\mathbb{T}$ with respect to $\leqslant$:

Lemma 1. *For all types $\theta \in \mathbb{T}$, there exists $\varphi \in \mathbb{F}$ such that $\theta \leqslant \varphi$.*

Proof. By structural induction on θ. □

Corollary 1. *For all types* $\Theta \in \mathbb{T}^*$*, there exists* $\Phi \in \mathbb{F}^*$ *such that* $\Theta \preccurlyeq \Phi$*.*

The next lemma states that $\mathbb{F}$ is an upper set in the preorder $\mathbb{T}$:

Lemma 2. *For all types* $\varphi \in \mathbb{F}$ *and* $\theta \in \mathbb{T}$*, if* $\varphi \preccurlyeq \theta$*, then* $\theta \in \mathbb{F}$*.*

Proof. By structural induction on φ. □

Corollary 2. *For all types* $\Phi \in \mathbb{F}^*$ *and* $\Theta \in \mathbb{T}^*$*, if* $\Phi \preccurlyeq \Theta$*, then* $\Theta \in \mathbb{F}^*$*.*

We assume that each $f \in \mathcal{F}$ has the unique associated type, $type(f) \in \mathbb{T}$. A *type assignment statement* is an expression of the form $x : \Phi$ or $A : \Theta$ with $A \in \mathcal{T} \backslash \mathcal{X}$, $\Phi \in \mathbb{F}^*$, $\Theta \in \mathbb{T}^*$. The types Φ, Θ are the *predicate* and x, A are the *subject* of the statement. A *declaration* is a statement whose subject is a variable. A *basis* Γ is a set of declarations with distinct variables as subjects. By $Subject(\Gamma)$ we denote the set of variables that are subjects of the declarations in Γ: $Subject(\Gamma) = \{x \mid x : \Phi \in \Gamma\}$. Note that the variables are assigned fixed arity types or starred fixed arity types, while constants may have an arbitrary associated type from $\mathbb{T}$. A statement $A : \Theta$ is *derivable* from a basis Γ, written $\Gamma \vdash A : \Theta$, if $\Gamma \vdash A : \Theta$ can be produced by the following rules:

$$\frac{}{\Gamma, x : \Phi \vdash x : \Phi}(\text{VAR}) \qquad \frac{}{\Gamma \vdash f : type(f)}(\text{FUN})$$

$$\frac{\Gamma \vdash A : \Theta \to \theta \quad \Gamma \vdash B : \Theta}{\Gamma \vdash AB : \theta}(\text{APP}) \qquad \frac{\Gamma \vdash A : \Theta_1 \quad \Theta_1 \preccurlyeq \Theta_2}{\Gamma \vdash A : \Theta_2}(\text{SUB})$$

$$\frac{\Gamma, \Delta \vdash A : \theta_1 \qquad \Gamma, \Delta \vdash B : \theta_2 \qquad Subject(\Delta) = \mathcal{V}}{\Gamma \vdash \lambda_{\mathcal{V}} A.B : \theta_1 \to \theta_2}(\text{ABS})$$

The type assignment statement and the derivability relation extend to hedges:

$$\frac{\Gamma \vdash A_1 : \Theta_1, \ldots, \Gamma \vdash A_n : \Theta_n \qquad \Theta_1 \preccurlyeq \Theta, \ldots, \Theta_n \preccurlyeq \Theta}{\Gamma \vdash \langle A_1, \ldots, A_n \rangle : \Theta}(\text{HED})$$

From this definition, by Corollary 2 we have the lemma:

Lemma 3. *If* $\Gamma \vdash x : \Theta$*, then* $\Theta \in \mathbb{F}^*$*.*

Example 1. Let $type(f) = \alpha_1 \to \alpha_1$, $type(g) = \alpha_2^* \to \alpha_1^* \to \alpha_2$, $type(a) = \alpha_1$, $type(b) = \alpha_2$, and $\Gamma = \{x : \alpha_1, y : \alpha_2^*, z : \alpha_1 \to \alpha_2\}$. Then some examples of derivable statements are

- $\Gamma \vdash x : \alpha_1$, $\Gamma \vdash x : \alpha_1^*$, and $\Gamma \vdash y : \alpha_2^*$.

- $\Gamma \vdash z : (\alpha_2^* \to \alpha_1) \to \alpha_2$ and $\Gamma \vdash z : (\alpha_2^* \to \alpha_2^* \to \alpha_1) \to \alpha_2$.
- $\Gamma \vdash (g\, b) : \alpha_2^* \to \alpha_1^* \to \alpha_2$, $\Gamma \vdash (g\, b) : \alpha_1^* \to \alpha_2$, and $\Gamma \vdash (g\, b) : \alpha_2$.
- $\Gamma \vdash (g\, (f\, a)) : \alpha_1^* \to \alpha_2$ and $\Gamma \vdash (g\, (f\, a)) : \alpha_2$.
- $\Gamma \vdash \langle y, b, (g\, b), (g\, (f\, a))\rangle : \alpha_2^*$.

We say that $\langle A_1, \ldots, A_n\rangle$ is a Γ-*typed hedge* iff there exists $\Theta \in \mathbb{T}^*$ such that $\Gamma \vdash \langle A_1, \ldots, A_n\rangle : \Theta$. Respectively, A is a Γ-*typed term* iff there exists $\theta \in \mathbb{T}$ such that $\Gamma \vdash A : \theta$. Under this definition, every Γ-typed term is also a Γ-typed hedge, but not vice versa. For instance, if $x : \varphi^\star \in \Gamma$, then x is a Γ-typed hedge, but not a Γ-typed term.

A *typed hedge* (resp. *typed term*) is a Γ-typed hedge (term) for some Γ. We use the letter h for typed hedges and the letters M, N, P, Q, W for typed terms.

Given a basis Γ, we define a Γ-*typed substitution* σ_Γ as a substitution from Γ-typed variables to Γ-typed hedges such that types are preserved. Type preservation means that for each variable x there exist types Φ and Θ with $\Theta \leqslant \Phi$ such that $x : \Phi \in \Gamma$ and $\Gamma \vdash \sigma_\Gamma(x) : \Theta$. The subscript Γ from σ_Γ is sometimes omitted, if it is clear from the context.

Note that the application of a Γ-substitution σ to a Γ-typed hedge maps Γ-typed hedges to Γ-typed hedges. Also, Γ-typed terms are mapped to Γ-typed terms (and not to arbitrary Γ-typed hedges).

Example 2. Let f, g, a, b, and Γ be defined as in Example 1. Let also $M = \lambda_{\{x\}}(f\, x).(g\, y\, x)$ and $\sigma = \{x \mapsto (f\, a), y \mapsto \langle b, (g\, b), (g\, (f\, a))\rangle\}$. Then we have $M\sigma = \lambda_{\{x\}}(fx).(g\, b\, (g\, b)\, (g\, (f\, a))\, x)$.

2.1.2 Reduction.

For a given Γ, a Γ-typed version of the pattern matching function $\mathcal{S}ol$ takes as parameters two Γ-typed terms P, Q and Γ-typed set of variables $\mathcal{V}$ and computes a finite set of Γ-typed substitutions. We denote it by $\mathcal{S}ol(P \lessdot^\Gamma_\mathcal{V} Q)$, dropping Γ when it does not cause a confusion. Then for Γ-typed terms P, N, Q and a Γ-typed substitution σ, β_p can be written as

$$\beta_\mathsf{p}: \quad (\lambda_\mathcal{V} P.N)\, Q \to N\sigma, \text{ where } \sigma \in \mathcal{S}ol(P \lessdot_\mathcal{V} Q).$$

Note that there is no need to require $N\sigma$ to be a term explicitly, because this property always holds due to the fact that the application of a Γ-typed substitution to a Γ-typed term gives a Γ-typed term.

3 Subject Reduction

The first interesting property of our calculus is subject reduction (SR), which essentially says that the $\twoheadrightarrow_{\beta_C}$ relation preserves types. SR is based on two lemmas: the Generation Lemma and the Substitution Lemma.

Lemma 4 (Generation Lemma)**.** *Let* Γ *be a basis.*

- *If* $\Gamma \vdash x : \Phi$*, then there exists* $\Phi' \leqslant \Phi$ *such that* $(x : \Phi') \in \Gamma$*.*
- *If* $\Gamma \vdash MN : \Theta$*, then there exist* Θ' *and* $\theta \leqslant \Theta$ *such that* $\Gamma \vdash M : \Theta' \to \theta$ *and* $\Gamma \vdash N : \Theta'$*.*
- *If* $\Gamma \vdash \lambda_{\mathcal{V}} P.N : \Theta$*, then there exist* θ_1, θ_2*, and* Δ *such that* $\theta_1 \to \theta_2 \leqslant \Theta$*,* $\mathit{Subject}(\Delta) = \mathcal{V}$*,* $\Gamma, \Delta \vdash P : \theta_1$ *and* $\Gamma, \Delta \vdash N : \theta_2$*.*
- *If* $\Gamma \vdash \langle M_1, \ldots, M_n \rangle : \Theta$*, then there exist* $\Theta_1 \leqslant \Theta, \ldots, \Theta_n \leqslant \Theta$ *such that* $\Gamma \vdash M_1 : \Theta_1, \ldots, \Gamma \vdash M_n : \Theta_n$*.*

Proof. By induction on the length of the type derivation. □

Lemma 5 (Substitution Lemma)**.** *Let* Γ *be a basis and* σ *be a* Γ*-typed substitution. If* $\Gamma \vdash M : \theta$*, then there exists* $\delta \leqslant \theta$ *such that* $\Gamma \vdash M\sigma : \delta$*.*

Proof. By structural induction on M. When $M = x$, either $x\sigma = x$ and $\delta = \theta$, or $x\sigma \neq x$ and by Lemma 4, there exists $\Phi \leqslant \theta$ such that $x : \Phi \in \Gamma$. Since σ is Γ-typed, there exists $\Phi' \leqslant \Phi$ such that $\Gamma \vdash x\sigma : \Phi'$ and we can take $\delta = \Phi' \leqslant \theta$.

The proof is easy for $M = f$ and $M = \lambda_{\mathcal{V}} P.N$. We only consider the case $M = M_1 x$. For $M = M_1 M_2$ with $M_2 \notin \mathcal{X}$ the proof is similar.

Let $M = M_1 x$. by Lemma 4, there exist Θ, θ' such that $\Gamma \vdash M_1 : \Theta \to \theta'$, $\Gamma \vdash x : \Theta$, and $\theta' \leqslant \theta$. Then $\Theta \in \mathbb{F}^*$. First, assume $\Theta = \varphi^*$ for some φ. Let $x\sigma = \langle N_1, \ldots, N_n \rangle$, $n \geqslant 0$. Then $(M_1 x)\sigma = ((M_1\sigma)N_1 \cdots N_n)$. By the IH there exists $\delta' \leqslant \varphi^* \to \theta'$ such that $\Gamma \vdash M_1\sigma : \delta'$. If $n = 0$, this already proves the lemma, because from $\delta' \leqslant \varphi^* \to \theta' \leqslant \theta'$, the SUB rule gives $\Gamma \vdash M_1\sigma : \theta'$, and we can take $\delta = \theta'$. If $n > 0$, by the SUB rule, we have $\Gamma \vdash M_1\sigma : \varphi^* \to \theta'$ and, eventually, $\Gamma \vdash M_1\sigma : \varphi^* \to \cdots \to \varphi^* \to \theta'$ for n-fold application. Since $\Gamma \vdash \langle N_1, \ldots, N_n \rangle : \varphi^*$, by HED, there exist $\Theta_1, \ldots, \Theta_n$ such that $\Gamma \vdash N_i : \Theta_i \leqslant \varphi^*$ for all $1 \leqslant i \leqslant n$. Then by SUB we have $\Gamma \vdash N_i : \varphi^*$ for all $1 \leqslant i \leqslant n$. Applying APP n-times, we get $\Gamma \vdash ((M_1\sigma)N_1 \ldots N_n) : \theta'$ and we can take $\delta = \theta' \leqslant \theta$.

Now assume $\Theta = \varphi$ for some φ. Then $\Gamma \vdash x\sigma : \varphi$. By the IH, $\Gamma \vdash M_1\sigma : \delta'$ and $\delta' \leqslant \varphi \to \theta'$. By SUB, $\Gamma \vdash M_1\sigma : \varphi \to \theta'$. Then APP gives $\Gamma \vdash (M_1\sigma)x\sigma : \theta'$ and we take $\delta = \theta' \leqslant \theta$. □

□

Theorem 1 (Subject Reduction)**.** *If* $M_1 \twoheadrightarrow_{\beta_C} M_2$ *and* $\Gamma \vdash M_1 : \Theta$*, then* $\Gamma \vdash M_2 : \Theta$*.*

Proof. We prove $\Gamma \vdash M_2 : \Theta$ from $\Gamma \vdash M_1 : \Theta$ and $M_1 \to_{\beta_C} M_2$. Then the theorem follows by induction on the length of the reduction sequence $M_1 \twoheadrightarrow_{\beta_C} M_2$.

We proceed by induction on the derivation of $\Gamma \vdash M_1 : \Theta$. When $\Gamma \vdash M_1 : \Theta$ is an axiom, then M_1 is either x or f and it can not be reduced by $\to_{\beta_C}$. Hence, the theorem follows trivially, since $M_1 \to_{\beta_C} M_2$ is not possible. When $\Gamma \vdash M_1 : \Theta$ is $\Gamma \vdash N_1 N_2 : \Theta$, then by Lemma 4 there exist Θ' and $\theta \preccurlyeq \Theta$ such that $\Gamma \vdash N_1 : \Theta' \to \theta$ and $\Gamma \vdash N_2 : \Theta'$. We have the following cases:

- $M_2 = N_1' N_2$ with $N_1 \to_{\beta_C} N_1'$, or $M_2 = N_1 N_2'$ with $N_2 \to_{\beta_C} N_2'$. In these cases we apply the IH to get $\Gamma \vdash M_2 : \theta$. Then SUB gives $\Gamma \vdash M_2 : \Theta$.

- $N_1 = \lambda_{\mathcal{V}} P.Q$ and $M_1 = (\lambda_{\mathcal{V}} P.Q) N_2 \to_{\beta_C} M_2 = Q\sigma$ where $\sigma \in \mathcal{S}ol(P \lessdot_{\mathcal{V}} N_2)$. σ is a Γ, Δ-based substitution, where $Subject(\Delta) = \mathcal{V}$. By Lemma 4, there exist Θ' and $\theta \preccurlyeq \Theta$ such that $\Gamma \vdash (\lambda_{\mathcal{V}} P.Q) : \Theta' \to \theta$ and $\Gamma \vdash N_2 : \Theta'$. Again, by Lemma 4 there exist θ_1, θ_2 such that $\theta_1 \to \theta_2 \preccurlyeq \Theta' \to \theta$, $\Gamma, \Delta \vdash P : \theta_1$, and $\Gamma, \Delta \vdash Q : \theta_2$. By the subtyping rules, we get $\Theta' \preccurlyeq \theta_1$ and $\theta_2 \preccurlyeq \theta$. By Lemma 5, we have $\Gamma, \Delta \vdash Q\sigma : \delta$ with $\delta \preccurlyeq \theta_2$. By $\mathbf{C}_0$, on the one hand we have $Dom(\sigma) = \mathcal{V}$ and on the other hand we have $Ran(\sigma) \cap \mathcal{V} = \varnothing$, hence we conclude $\mathtt{fv}(Q\sigma) \cap \mathcal{V} = \varnothing$. Since $\mathcal{V} = Subject(\Delta)$, from $\Gamma, \Delta \vdash Q\sigma : \delta$ we get $\Gamma \vdash Q\sigma : \delta$ with $\delta \preccurlyeq \theta_2$. We also know $\theta_2 \preccurlyeq \theta \preccurlyeq \Theta$. Hence, by SUB we conclude $\Gamma \vdash Q\sigma : \Theta$.

When $\Gamma \vdash M_1 : \Theta$ is $\Gamma \vdash \lambda_{\mathcal{V}} P.Q : \Theta$, then by Lemma 4 there exist $\theta_1 \to \theta_2 \preccurlyeq \Theta$ and Δ such that $Subject(\Delta) = \mathcal{V}$, $\Gamma, \Delta \vdash P : \theta_1$, and $\Gamma, \Delta \vdash Q : \theta_2$. If $M_2 = \lambda_{\mathcal{V}} P'.Q$ with $P \to_{\beta_C} P'$ or $M_2 = \lambda_{\mathcal{V}} P.Q'$ with $Q \to_{\beta_C} Q'$, then by the IH and ABS we get $\Gamma \vdash M_2 : \theta_1 \to \theta_2$. Finally, SUB gives $\Gamma \vdash M_2 : \Theta$. □ □

Example 3. If variables were permitted to have arbitrary types instead of fixed arity types or starred fixed arity types, then SR would not hold: Assume $(x : \alpha^* \to \alpha) \in \Gamma$ and $type(a) = \alpha$. Then we have $\Gamma \vdash x : \alpha \to \alpha$, $\Gamma \vdash x : \alpha$, $\Gamma \vdash \lambda_{\{x\}} x.(x\,x) : \alpha \to \alpha$ and, finally, $\Gamma \vdash (\lambda_{\{x\}} x.(x\,x) a) : \alpha$. However, $(a\,a)$, that is obtained by reducing $(\lambda_{\{x\}} x.(x\,x)\,a)$, is not typeable anymore.

Acknowledgments

This research has been partially supported by the Shota Rustaveli National Science Foundation of Georgia under the grant FR-19-18557, and by the Shota Rustaveli National Science Foundation of Georgia and Turkish Scientic and Technological Research Council joint grant 04/03.

References

1. Alves, S., Dundua, B., Florido, M., and Kutsia, T. Pattern-based calculi with finitary matching. *Log. J. IGPL 26*, 2 (2018), 203–243.

2. Barendregt, H. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, 1984. Revised edition.

3. Barthe, G., Cirstea, H., Kirchner, C., and Liquori, L. Pure patterns type systems. In *POPL* (2003), A. Aiken and G. Morrisett, Eds., ACM, pp. 250–261.

4. Benzaken, V., Castagna, G., and Frisch, A. CDuce: an XML-centric general-purpose language. In *ICFP* (2003), C. Runciman and O. Shivers, Eds., ACM, pp. 51–63.

5. Cirstea, H., and Kirchner, C. The rewriting calculus - parts I and II. *Logic Journal of the IGPL 9*, 3 (2001).

6. Cirstea, H., Kirchner, C., and Liquori, L. Rewriting calculus with(out) types. *Electr. Notes Theor. Comput. Sci. 71* (2002), 3–19.

7. Coelho, J., and Florido, M. Xcentric: A logic-programming language for xml processing. In *PLAN-X* (2007), pp. 93–94.

8. Dundua, B., Kutsia, T., and Marin, M. Strategies in Pρlog. In *WRS* (2009), M. Fernández, Ed., vol. 15 of *EPTCS*, pp. 32–43.

9. Dundua, B., Kutsia, T., and Reisenberger-Hagmayer, K. An overview of pρlog. In *Practical Aspects of Declarative Languages - 19th International Symposium, PADL 2017, Paris, France, January 16-17, 2017, Proceedings* (2017), Y. Lierler and W. Taha, Eds., vol. 10137 of *Lecture Notes in Computer Science*, Springer, pp. 34–49.

10. Hosoya, H., and Pierce, B. C. Xduce: A statically typed xml processing language. *ACM Trans. Internet Techn. 3*, 2 (2003), 117–148.

11. Jay, B. *Pattern Calculus*. Springer, 2009.

12. Jay, C. B., and Kesner, D. Pure pattern calculus. In *ESOP* (2006), P. Sestoft, Ed., vol. 3924 of *LNCS*, Springer, pp. 100–114.

13. Jay, C. B., and Kesner, D. First-class patterns. *J. Funct. Program. 19*, 2 (2009), 191–225.

14. Kesner, D., Puel, L., and Tannen, V. A typed pattern calculus. *Inf. Comput. 124*, 1 (1996), 32–61.

15. Klop, J. W., van Oostrom, V., and de Vrijer, R. C. Lambda calculus with patterns. *Theor. Comput. Sci. 398*, 1-3 (2008), 16–31.

16. Peyton Jones, S. L., and Wadler, P. *The Implementation of Functional Programming Languages*. Prentice Hall, 1987, ch. 4: Structured Types and the Semantics of Pattern Matching.

17. Pkhakadze, S. A n. bourbaki type general theory and the properties of contracting symbols and corresponding contracted forms. *Georgian Mathematical Journal 6*, 2 (1999), 179–190.

18. Sh, P. Some problems of the notation theory. In *Proceedings of I. Vekua Institute of Applied Mathematics of Tbilisi State University, Tbilisi* (1977).

19. Sulzmann, M., and Lu, K. Z. M. XHaskell - adding regular expression types to Haskell. In *IFL* (2007), O. Chitil, Z. Horváth, and V. Zsók, Eds., vol. 5083 of *LNCS*, Springer, pp. 75–92.