Lambda Calculus with Regular Types

Besik Dundua
Institute of Applied Mathematics
Tbilisi State University
bdundua@gmail.com

Mário Florido
Department of Computer Science
University of Porto
amf@dcc.fc.up.pt

Temur Kutsia
Research Institute for Symbolic Computation
Johannes Kepler University Linz
kutsia@risc.jku.at

Abstract—**In this paper we define λ_{R}: a foundational calculus for sequence processing with regular expression types. Its term language is the lambda calculus extended with sequences of terms and its types are regular expressions over simple types. We provide a flexible notion of subtyping based on the semantic notion of nominal interpretation of a type. Then we prove that types are preserved by reduction (subject reduction), and that there exist no infinite reduction sequences starting at typed terms (strong normalization).**

I. Introduction

Lambda calculus is regarded as a core formalism to model programming languages. It provides a framework for both computation and deduction. Since the pioneering works of Landin [13]–[16] and McCarthy [17], [18], lambda calculus plays an important role in specification and design of programming languages, for some compiler construction techniques, and in studying type systems.

The original version of lambda calculus, introduced by Church in 1930s, was untyped and permitted any expression (function) to be applied to every other expression (argument). Any computable function can be represented as a term of this calculus, which gave rise to the functional style of programming.

Types were introduced to restrict arbitrary function applications. Simply typed lambda calculus is the most basic typed lambda calculus, with the only type constructor $\to$ for function types. One of the important properties it possesses is so called strong normalization: typed terms always have a normal form. Taking into account undecidability of halting problem, it means that not all computable functions can be expressed as a typed term in this calculus. Hence, simply typed lambda calculus is not Turing complete. However, as Barendregt [3] notes, it is not as bad as it sounds, since computable functions that can not be represented as typed terms are not easy to construct.

To close the gap between simply typed lambda calculus and programming languages, extensions of the calculus have been proposed. Pierce's book [19] gives a detailed overview of various such extending features, starting from simple ones such as, e.g., tuples, records, sums, recursion, lists, and going up to subtyping and polymorphism, even for higher-order systems.

The extension that we propose in this paper brings regular expressions in types. By this, it also includes term sequences of arbitrary finite length into the language. A characteristic feature of sequences is that they are flat and can not be nested, and a singleton sequence is identified with its sole element.

The obtained calculus, which we denote by λ_{R}, is quite flexible. It can easily model, for instance, so called unranked functions (these are functions that can be applied to arbitrary number of arguments) and their curried applications, e.g., the term of the form $f[a][f[b, \lambda x.g[x]], d]$ is typable. It can be done if f has the type, say, $\alpha \cdot (\beta^* \to \beta)^* \to \alpha^* \to \alpha$, the terms a, b, and d have the type α, g has the type $\beta^* \to \beta$, and x is of type β^*, where the dot '$\cdot$' and the star '$*$' are the regular operators for concatenation and repetition (Kleene star), respectively.

Regular types naturally introduce subtyping, since α is a subtype of α^* and of $\alpha+\beta$. Besides, we consider a partial order on basic types and define a syntax-oriented subtyping relation. The two main results that we prove are those that one usually tries to show for typed lambda calculi: subject reduction (evaluation is type-safe) and strong normalization (every term has a normal form).

λ_{R} generalizes simply typed lambda calculus: If we forbid the regular operators in types and forget subtyping over basic types, we get simply typed lambda calculus.

The reader familiar with the programming language of the symbolic computation system Mathematica [21] (also known as the Wolfram language) might notice that our calculus can be quite convenient to model a part of this language. Mathematica expressions are essentially untyped first- or higher-order unranked terms with a pretty liberal application rule, see, e.g. [5]. The strong normalization property of our calculus does not permit to express the whole language.

The paper is organized as follows. In the next section we give an overview of related work. Section III describes the term language. In Section IV we define the type language and the type system. We then have two sections with the two fundamental properties of the type system: subject seduction in Section V and strong normalization in Section VI. Section VII concludes.

II. Related Work

Previous work on regular expression types was mainly motivated by XML-related applications. Schema languages for XML introduce regular expression notation to describe XML documents. Languages such as XDuce [11], XHaskell [20], CDuce [4], XCentric [7], designed for XML processing, use regular expressions and regular tree languages as types and in pattern matching. These works deal with programming languages for tree manipulation (thus are not strongly normalizing) and with constructors in their syntax (thus the need for considering tree languages instead of simple regular languages). In our work we define a core functional calculus and focus on the functional behavior of sequence processing, thus we choose not to consider constructors in the term syntax. A generalization of our work to a term language with constructors would need to extend types with tree languages and could provide a model of these languages, but, although interesting, this was not the goal of this work.

In [12], unification and matching with regular expression sorts has been studied. The sort system there is pretty similar to our types, with the difference that the language in [12] is first-order.

Recently, we introduced CLP(H): a constraint logic programming language for hedges (finite sequences of unranked terms) [8], [9]. In that work, regular expressions fit naturally as membership constraints modeling dynamic type checking. In contrast, the calculus λ_{R} is motivated by a static typing approach, where types are checked at compile time. Our subtyping relation is semantic. However, dealing with regular languages for which the subset relation is decidable, makes it decidable. Semantic subtyping [10] gave a model to deal with set-theoretic operations on types (such as union, intersection and negation). We define our subtyping relation as directly relying on the decidability of the subset relation for regular languages, but it could be interesting in the future to consider how our simple and decidable subtyping definition would fit in the semantic subtyping context.

III. Terms

The alphabet of the language of λ_{R} consists of the set $\mathcal{X}$ of variables and $\mathcal{F}$ of function constants. They are disjoint and countably infinite. The symbols x and f range over $\mathcal{X}$ and $\mathcal{F}$, respectively.

Untyped terms are defined by the following grammar:

$$A ::= x \mid \ f \mid A[A_1,\dots,A_n] \mid \lambda x.A$$

where $A[A_1,\dots,A_n]$ is an *application* and $\lambda x.A$ is an *abstraction*. When $n = 1$ the grammar generates the standard untyped λ-terms [2].

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

$$\mathtt{fv}(x) = \{x\} \quad \mathtt{fv}(f) = \emptyset$$
$$\mathtt{fv}(\lambda x.A) = \mathtt{fv}(A) \setminus \{x\}$$
$$\mathtt{fv}(A[A_1,\dots,A_n]) = \mathtt{fv}(A) \cup (\cup_{i=1}^n \mathtt{fv}(A_i))$$

$$\mathtt{bv}(x) = \emptyset \quad \mathtt{bv}(f) = \emptyset$$
$$\mathtt{bv}(\lambda x.A) = \mathtt{bv}(A) \cup \{x\}$$
$$\mathtt{bv}(A[A_1,\dots,A_n]) = \mathtt{bv}(A) \cup (\cup_{i=1}^n \mathtt{bv}(A_i))$$

The sets of free and bound variables of a term sequence $[A_1,\dots,A_n]$ are defined as $\mathtt{fv}([A_1,\dots,A_n]) = \cup_{i=1}^n \mathtt{fv}(A_i)$ and $\mathtt{bv}([A_1,\dots,A_n]) = \cup_{i=1}^n \mathtt{bv}(A_i)$ respectively. Below we work modulo α-equivalence and use Barendregt's hygiene convention [2] to distinguish the names of free and bound variables.

IV. Types

We consider a finite set $\mathcal{B}$ of basic types, partially ordered with the relation $\leqq$. Its elements are denoted by α and β. *Types* are defined by the grammar

$$\tau, \rho ::= \varphi \mid \epsilon \mid \tau \cdot \rho \mid \tau{+}\rho \mid \tau^*, \qquad \varphi ::= \alpha \mid \tau \to \varphi,$$

where $\cdot$ (concatenation), $+$ (choice), and * (Kleene star) are regular operators.

Note that our arrow types have the form $\tau_1 \to \dots \to \tau_n \to \alpha$ for $n \geq 1$. We often will use this more explicit notation.

For a given set $\mathcal{S}$, $[s_1,\dots,s_n]$ denotes a finite *sequence* of elements $s_1,\dots,s_n$ of $\mathcal{S}$. In particular, the empty sequence is written as $[\,]$. For singleton sequences, we usually omit the parentheses and write s instead of $[s]$, when it does not cause a confusion. The result of joining two sequences $[s_1,\dots,s_n]$ and $[s'_1,\dots,s'_m]$, written $[s_1,\dots,s_n] \asymp [s'_1,\dots,s'_m]$, is the sequence $[s_1,\dots,s_n,s'_1,\dots,s'_m]$. As usual, $\asymp$ is associative and the empty sequence plays the role of its unit element.

We define the operations of concatenation and Kleene closure on sets of sequences: Given two such sets, S and R, their concatenation $S{\cdot}R$ is the set of sequences $\{[s_1,\dots,s_n] \asymp [r_1,\dots,r_m] \mid [s_1,\dots,s_n] \in S$ and $[r_1,\dots,r_m] \in R\}$. The Kleene closure of S, denoted S^*, is defined as $S^* = \cup_{i\geq 0} S^i$, where $S^0 = \{[\,]\}$ and $S^n = S^{n-1} \cdot S$ for $n > 0$.

The *nominal interpretation* $(\!|.|\!)$ *of a type* is defined as a set of finite sequences, where each element of the sequence is either a basic type or a tuple of nominal interpretations. We call those elements *nominal atoms*. The definition of nominal type interpretations is as follows:

- $(\!|\alpha|\!) = \{[\alpha]\}$, i.e., the set that contains a singleton sequence $[\alpha]$.
- $(\!|\epsilon|\!) = \{[\,]\}$.
- $(\!|\tau \cdot \rho|\!) = (\!|\tau|\!) \cdot (\!|\rho|\!)$.
- $(\!|\tau{+}\rho|\!) = (\!|\tau|\!) \cup (\!|\rho|\!)$.
- $(\!|\tau^*|\!) = (\!|\tau|\!)^*$.
- $(\!|\tau_1 \to \dots \to \tau_n \to \alpha|\!) = \{[\langle (\!|\tau_1|\!),\dots,(\!|\tau_n|\!),(\!|\alpha|\!)\rangle]\}$, $n > 0$, i.e., the set that contains a singleton sequence consisting of the tuple of nominal interpretations $\langle (\!|\tau_1|\!),\dots,(\!|\tau_n|\!),(\!|\alpha|\!)\rangle$.

Since any type $\tau \to \varphi$ has the form $\tau_1 \to \dots \to \tau_n \to \alpha$ for some $\tau_1,\dots,\tau_n,\alpha$, nominal type interpretation is defined for all types.

Examples:

$$(\!|\alpha^*|\!) = \{[\,],[\alpha],[\alpha,\alpha],[\alpha,\alpha,\alpha],\dots\}.$$
$$(\!|\alpha^* \to \beta|\!) = \{[\langle\{[\,],[\alpha],[\alpha,\alpha],[\alpha,\alpha,\alpha],\dots\},\{[\beta]\}\rangle]\}.$$
$$(\!|\alpha^* \to (\beta\cdot\beta) \to \beta|\!) =$$
$$\{[\langle\{[\,],[\alpha],[\alpha,\alpha],[\alpha,\alpha,\alpha],\dots\},\{[\beta,\beta]\},\{[\beta]\}\rangle]\}.$$
$$(\!|(\alpha^* \to \beta)\cdot\alpha^*|\!) =$$
$$\{[\langle\{[\,],[\alpha],[\alpha,\alpha],[\alpha,\alpha,\alpha],\dots\},\{[\beta]\}\rangle],$$

$$
\begin{aligned}
&\quad [\langle\{[\,],[\alpha],[\alpha,\alpha],[\alpha,\alpha,\alpha],\ldots\},\{[\beta]\}\rangle,\alpha],\\
&\quad [\langle\{[\,],[\alpha],[\alpha,\alpha],[\alpha,\alpha,\alpha],\ldots\},\{[\beta]\}\rangle,\alpha,\alpha],\ldots\}.\\
&(\!|(\alpha^* \to \beta)\cdot\alpha^* \to \beta|\!) =\\
&\quad \{[\langle\{[\langle\{[\,],[\alpha],[\alpha,\alpha],[\alpha,\alpha,\alpha],\ldots\},\{[\beta]\}\rangle],\\
&\qquad [\langle\{[\,],[\alpha],[\alpha,\alpha],[\alpha,\alpha,\alpha],\ldots\},\{[\beta]\}\rangle,\alpha],\\
&\qquad [\langle\{[\,],[\alpha],[\alpha,\alpha],[\alpha,\alpha,\alpha],\ldots\},\{[\beta]\}\rangle,\alpha,\alpha],\ldots\},\\
&\quad \{[\beta]\}\rangle]\}.
\end{aligned}
$$

Definition 1 ($\preceq$ ordering, subtyping)**.** *We define the* ordering $\preceq$ *for nominal atoms and their sequences recursively:*

- $\alpha \preceq \beta$ *iff* $\alpha \leqq \beta$.
- $\langle D_1,\ldots,D_n,R\rangle \preceq \langle D'_1,\ldots,D'_n,R'\rangle$ *iff* $D'_i \preceq D_i$ *for all* $1 \le i \le n$ *and* $R \preceq R'$.
- *For two equal-length sequences of nominal atoms, we have* $[a_1,\ldots,a_n] \preceq [b_1,\ldots,b_n]$ *iff* $a_i \preceq b_i$ *for each* $1 \le i \le n$.

This ordering is extended to nominal type interpretations:

- *For nominal type interpretations* I_1 *and* I_2*, we have* $I_1 \preceq I_2$ *iff for each sequence of nominal atoms* $\tilde{s}_1 \in I_1$ *there is a sequence of nominal atoms* $\tilde{s}_2 \in I_2$ *such that* $\tilde{s}_1 \preceq \tilde{s}_2$.

The subtyping relation *between types is defined as* $\tau_1 \preccurlyeq \tau_2$ *iff* $(\!|\tau_1|\!) \preceq (\!|\tau_2|\!)$.

Under this definition, for example, if $\alpha \leqq \beta$, we have $\alpha\cdot\beta\cdot\beta^* \preccurlyeq \beta^*$, $\alpha+\beta \preccurlyeq \beta$, $\beta^* \to \alpha \preccurlyeq \alpha^* \to \beta$ and $(\alpha^* \to \beta)\cdot\alpha^* \to \beta \preccurlyeq (\beta^* \to \alpha) \to \beta$.

The subtyping relation is decidable. We do not spell the details of the decision algorithm here, but give some intuition behind it. First, note that to decide whether $\tau_1 \to \varphi_1 \preccurlyeq \tau_2 \to \varphi_2$, we need to decide whether $\tau_2 \preccurlyeq \tau_1$ and $\varphi_1 \preccurlyeq \varphi_2$. That means, eventually the subtyping decision problem reduces to the subtyping decision problem between two types that do not contain the arrow, i.e., between regular expressions over basic types. If we did not have subtyping between basic types, this problem can be effectively decided by any algorithm that checks inclusion between regular languages, e.g., by Antimirov's algorithm [1] that performs this test on regular expressions without translating them to finite automata. The presence of subtyping between basic types requires a slight preprocessing of types (see, e.g., [12]): Replace each basic type α with the sum $\beta_1+\cdots+\beta_n$, $n > 0$, where β_i, $1 \le i \le n$, are all the basic types for which $\beta_i \preccurlyeq \alpha$ holds. Since $\mathcal{B}$ is finite, the sum is finite. Correctness of such a translation for the subtyping decision algorithm has been proved in [12]. The obtained regular expressions can then be compared by Antimirov's algorithm.

A. Type System

We assume that each $f \in \mathcal{F}$ has the unique associated type, denoted by $type(f)$, which is a type φ, i.e., either α or $\tau_1 \to \cdots \to \tau_n \to \alpha$. A *type assignment statement* is an expression of the form $A : \tau$ where the term A is called the *subject* and the type τ is the *predicate* 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: $Subject(\Gamma) = \{x \mid x : \tau \in \Gamma \text{ for some } \tau\}$.

A statement $A : \tau$ is *derivable* from a basis Γ, written $\Gamma \vdash A : \tau$, if $\Gamma \vdash A : \tau$ can be produced by the following rules:

$$
\frac{}{\Gamma, x:\tau \vdash x:\tau}\,(\textsc{var}) \qquad \frac{}{\Gamma \vdash f : type(f)}\,(\textsc{fun})
$$

$$
\frac{\begin{array}{c}\Gamma \vdash A_1 : \tau_1 \ \cdots\ \Gamma \vdash A_n : \tau_n \qquad n > 0\\ \Gamma \vdash A : \tau \to \varphi \qquad \tau_1\cdot\ldots\cdot\tau_n \preccurlyeq \tau\end{array}}{\Gamma \vdash A[A_1,\ldots,A_n] : \varphi}\,(\textsc{app1})
$$

$$
\frac{\Gamma \vdash A : \tau \to \varphi \qquad \epsilon \preccurlyeq \tau}{\Gamma \vdash A[\,] : \varphi}\,(\textsc{app2})
$$

$$
\frac{\Gamma, x:\tau \vdash A : \varphi}{\Gamma \vdash \lambda x.A : \tau \to \varphi}\,(\textsc{abs})
$$

Note the absence of the subsumption rule and, instead, the usage of subtyping in the APP1 and APP2 rules. By this, the rules are syntax directed (cf., e.g., the algorithmic typing relations in [19], or typing rules in [6].)

Given a basis Γ, we say that a term A is a Γ*-typed term of type* τ, if $\Gamma \vdash A : \tau$ holds.

A Γ*-typed substitution* σ_Γ is a mapping from Γ-typed variables to sequences of Γ-typed terms such that for each variable x of type τ, we have $\sigma_\Gamma(x) = [A_1,\ldots,A_n]$, where

- if $n = 0$, then $\epsilon \preccurlyeq \tau$,
- if $n > 0$ and A_i is of type τ_i for each $1 \le i \le n$, then $\tau_1\cdot\cdots\cdot\tau_n \preccurlyeq \tau$.

Each substitution σ is represented as a finite set of pairs $\{x_1 \mapsto \sigma(x_1),\ldots,\, x_n \mapsto \sigma(x_n)\}$ where the x's are variables for which $\sigma(x_i) \ne x_i$. 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 $\mathtt{fv}(\sigma)$ is defined as $\mathtt{fv}(\sigma) = Dom(\sigma) \cup \mathtt{fv}(Ran(\sigma))$ where $\mathtt{fv}(Ran(\sigma)) = \cup_{i=1}^{n}\mathtt{fv}(\sigma(x_i))$

The *application* of a substitution σ to A replaces each *free* occurrence of a variable x in A with $\sigma(x)$. It is defined inductively:

$$
\begin{aligned}
&x\sigma = \sigma(x), \text{ if } x \in Dom(\sigma).\\
&x\sigma = x, \text{ if } x \notin Dom(\sigma).\\
&f\sigma = f.\\
&(A[A_1,\ldots,A_n])\sigma = A\sigma[A_1,\ldots,A_n]\sigma.\\
&(\lambda x.A)\sigma = \lambda x.A\sigma, \text{ if } x \notin \mathtt{fv}(\sigma).\\
&(\lambda x.A)\sigma = \lambda y.A\{x \mapsto y\}\sigma, \quad \text{if } x \in \mathtt{fv}(\sigma) \text{ and } y \text{ is fresh.}
\end{aligned}
$$

Application of a substitution σ to a sequence is defined as $[\,]\sigma = [\,]$ and $[A_1,\ldots,A_n]\sigma = A_1\sigma \asymp \cdots \asymp A_n\sigma$ for $n > 0$.

Note that for a non-variable term A, $A\sigma_\Gamma$ is always a single term (a singleton sequence). For a variable x, it is a single term if $\Gamma \vdash x : \varphi+\cdots+\varphi$.

B. Reduction

A binary relation $\to_R$ is compatible if it is defined by the inference rules below.

$$\frac{A \to_R A'}{A[A_1,\dots,A_n] \to_R A'[A_1,\dots,A_n]} \qquad \frac{A \to_R A'}{\lambda x.A \to_R \lambda x.A'}$$

$$\frac{A_i \to_R A_i'}{A[A_1,\dots,A_i,\dots,A_n] \to_R A[A_1,\dots,A_i',\dots,A_n]}$$

Reduction is defined by the following rule:

$$\beta: \quad (\lambda x.A)\,[A_1,\dots,A_n] \to A\sigma, \quad \text{where } \sigma = \{x \mapsto [A_1,\dots,A_n]\}.$$

Example 1. *Some examples of reduction:*

- $\big(\lambda x.f[x][\lambda y.c[y],x]\big)[a,b] \to f[a,b][\lambda y.c[y],a,b]$.
- $\big(\lambda x.f[x][\lambda y.c[y],x]\big)[] \to f[][\lambda y.c[y]]$.
- $\big(\lambda x.x[a,b]\big)[\lambda y.f[y,y][y]] \to \big(\lambda y.f[y,y][y]\big)[a,b] \to f[a,b,a,b][a,b]$.

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

Lemma 1. *For all terms A, A', and all substitutions σ, σ' with $\mathit{Dom}(\sigma) = \mathit{Dom}(\sigma')$, if $A \twoheadrightarrow_\beta A'$ and $\sigma \twoheadrightarrow_\beta \sigma'$, then $A\sigma \twoheadrightarrow_\beta A'\sigma'$.*

Proof. A straightforward proof, by induction on $\twoheadrightarrow_\beta$. □

V. Generalized Subject Reduction

The subject reduction theorem states that the reduction relation preserves types. Since typing in our calculus is syntax-directed, without the subsumption rule, we will have a generalized version of the subject reduction theorem (GSR), which essentially states that the reduction relation does not increase types.

GSR is based on two lemmas: the Generation Lemma and the Substitution Lemma.

Lemma 2 (Generation Lemma). *Let Γ be a basis.*

- *If $\Gamma \vdash x : \tau$, then $(x : \tau) \in \Gamma$.*
- *If $\Gamma \vdash f : \tau$, then $\tau = \mathit{type}(f)$.*
- *If $\Gamma \vdash A[A_1,\dots,A_n] : \varphi$, then there exist $\tau, \tau_1, \dots, \tau_n$, $n > 0$ such that $\Gamma \vdash A : \tau \to \varphi$, $\Gamma \vdash A_1 : \tau_1$,..., $\Gamma \vdash A_n : \tau_n$, and $\tau_1 \cdot \dots \cdot \tau_n \preccurlyeq \tau$.*
- *If $\Gamma \vdash A[] : \varphi$, then there exists τ such that $\Gamma \vdash A : \tau \to \varphi$ and $\epsilon \preccurlyeq \tau$.*
- *If $\Gamma \vdash \lambda x.A : \tau$, then there exist ρ and φ such that $\tau = \rho \to \varphi$ and $\Gamma, x : \rho \vdash A : \varphi$.*

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

Lemma 3 (Substitution Lemma). *Let Γ be a basis and σ be a Γ-typed substitution. If $\Gamma \vdash A : \tau_1$, then $\Gamma \vdash A\sigma : \tau_2$, where $\tau_2 \preccurlyeq \tau_1$.*

Proof. By structural induction on A.

- When $A = x$, either $x\sigma = x$ and the lemma trivially holds, or $x\sigma \neq x$ and by Lemma 2 we have $(x : \tau_1) \in \Gamma$. Since σ is Γ-typed, there exists τ_2 such that $\tau_2 \preccurlyeq \tau_1$ and $\Gamma \vdash x\sigma : \tau_2$.
- When $A = f$, the proof is obvious.
- When $A = A[A_1,\dots,A_n]$, then by Lemma 2 there exist $\rho, \rho_1, \dots, \rho_n$, $n > 0$ such that $\Gamma \vdash A : \rho \to \tau_1$, $\Gamma \vdash A_1 : \rho_1$,..., $\Gamma \vdash A_n : \rho_n$, and $\rho_1 \cdot \dots \cdot \rho_n \preccurlyeq \rho$. By the induction hypothesis there exist $\rho', \varphi', \rho_1', \dots, \rho_n'$ such that $\rho' \to \varphi' \preccurlyeq \rho \to \tau_1$, $\rho_1' \preccurlyeq \rho_1$, ... ,$\rho_n' \preccurlyeq \rho_n$ and $\Gamma \vdash A\sigma : \rho' \to \varphi'$, $\Gamma \vdash A_1\sigma : \rho_1'$,..., $\Gamma \vdash A_n\sigma : \rho_n'$. From $\rho' \to \varphi' \preccurlyeq \rho \to \tau_1$ we know $\rho \preccurlyeq \rho'$, $\varphi' \preccurlyeq \tau_1$. Also, $\rho_1' \preccurlyeq \rho_1$, ... ,$\rho_n' \preccurlyeq \rho_n$ implies $\rho_1' \cdot \dots \cdot \rho_n' \preccurlyeq \rho_1 \cdot \dots \cdot \rho_n$. Therefore, we have $\rho_1' \cdot \dots \cdot \rho_n' \preccurlyeq \rho_1 \cdot \dots \cdot \rho_n \preccurlyeq \rho \preccurlyeq \rho'$ and, by the rule APP1, we can conclude that $\Gamma \vdash A[A_1,\dots,A_n]\sigma : \varphi'$. We take $\tau_2 = \varphi' \preccurlyeq \tau_1$, which proves this case.
- When $A = A[\,]$, then by Lemma 2, there exists ρ such that $\epsilon \preccurlyeq \rho$ and $\Gamma \vdash A : \rho \to \tau_1$. By the induction hypothesis there exist ρ' and φ such that $\rho' \to \varphi \preccurlyeq \rho \to \tau_1$ and $\Gamma \vdash A\sigma : \rho' \to \varphi$. Since $\rho' \to \varphi \preccurlyeq \rho \to \tau_1$ we have $\rho \preccurlyeq \rho'$, $\varphi \preccurlyeq \tau_1$ and hence $\epsilon \preccurlyeq \rho \preccurlyeq \rho'$. By the rule APP2, we can conclude $\Gamma \vdash A[\,]\sigma : \varphi$. We take $\tau_2 = \varphi \preccurlyeq \tau_1$, which proves this case.
- When $A = \lambda x.A$, then by Lemma 2 there exist ρ, φ such that $\tau_1 = \rho \to \varphi$ and $\Gamma, x : \rho \vdash A : \varphi$. By the induction hypothesis there exists φ' such that $\varphi' \preccurlyeq \varphi$ and $\Gamma, x : \rho \vdash A\sigma : \varphi'$. By the rule ABS, we can conclude that $\Gamma \vdash \lambda x.A : \rho \to \varphi'$. We take $\tau_2 = \rho \to \varphi' \preccurlyeq \rho \to \varphi = \tau_1$ to finish the proof.

□

Theorem 1 (Generalized Subject Reduction). *If $A_1 \twoheadrightarrow_\beta A_2$ and $\Gamma \vdash A_1 : \tau_1$, then $\Gamma \vdash A_2 : \tau_2$ and $\tau_2 \preccurlyeq \tau_1$.*

Proof. We prove $\Gamma \vdash A_2 : \tau_2$ from $\Gamma \vdash A_1 : \tau_1$ and $A_1 \to_\beta A_2$. Then the theorem follows by induction on the length of the reduction sequence $A_1 \twoheadrightarrow_\beta A_2$.

We proceed by induction on the derivation of $\Gamma \vdash A_1 : \tau_1$.

- When A_1 is either x or f, then it can not be reduced by $\to_\beta$. Hence, the theorem follows trivially, since $A_1 \to_\beta A_2$ is not possible.
- When $A_1 = A'[A_1',\dots,A_n']$ and $n > 0$, then by Lemma 2 there exist $\rho', \rho_1', \dots, \rho_n'$ such that $\rho_1' \cdot \dots \cdot \rho_n' \preccurlyeq \rho'$ and $\Gamma \vdash A' : \rho' \to \tau_1$, $\Gamma \vdash A_1' : \rho_1'$,..., $\Gamma \vdash A_n' : \rho_n'$. We have the following cases:
  - $A_2 = A''[A_1',\dots,A_n']$ with $A' \to_\beta A''$. By the induction hypotheses there exist ρ'', τ'' such that $\rho'' \to \tau'' \preccurlyeq \rho' \to \tau_1$ and $\Gamma \vdash A'' : \rho'' \to \tau''$. Since $\rho'' \to \tau'' \preccurlyeq \rho' \to \tau_1$ we have $\rho' \preccurlyeq \rho''$ and $\tau'' \preccurlyeq \tau_1$, hence $\rho_1' \cdot \dots \cdot \rho_n' \preccurlyeq \rho' \preccurlyeq \rho''$ and by the typing rule APP1 we get $\Gamma \vdash A_2 = A''[A_1',\dots,A_n'] : \tau''$. Then we can just take $\tau_2 = \tau''$.
  - $A_2 = A'[A_1',\dots,A_i'',\dots A_n']$ with $A_i' \to_\beta A_i''$. By the induction hypotheses there exists ρ_i'' such that $\rho_i'' \preccurlyeq \rho_i'$

and $\Gamma \vdash A''_i : \rho''_i$. Since $\rho'_1 \cdot \cdots \cdot \rho'_i \cdot \cdots \cdot \rho'_n \preccurlyeq \rho'$ we have $\rho'_1 \cdot \cdots \cdot \rho''_i \cdot \cdots \cdot \rho'_n \preccurlyeq \rho'$ and by the typing rule APP1 we get $\Gamma \vdash A'[A'_1, \ldots, A''_i, \ldots A'_n] : \tau_1$. Then we can just take $\tau_2 = \tau_1$.

– $A' = \lambda x.A$, i.e., $A_1 = (\lambda x.A)[A'_1, \ldots, A'_n]$. By Lemma 2 there exist $\tau', \tau'_1, \ldots, \tau'_n$, $n > 0$ such that $\Gamma \vdash \lambda x.A : \tau' \to \tau_1$, $\Gamma \vdash A_1 : \tau'_1, \ldots, \Gamma \vdash A_n : \tau'_n$, and $\tau'_1 \cdot \cdots \cdot \tau'_n \preccurlyeq \tau'$. Since we have $\Gamma \vdash \lambda x.A : \tau' \to \tau_1$, Lemma 2 implies that $\Gamma, x : \tau' \vdash A : \tau_1$. The reduction reduces the term $A_1 = (\lambda x.A)[A'_1, \ldots, A'_n]$ to $A\sigma = A_2$, where $\sigma = \{x \mapsto [A'_1, \ldots, A'_n]\}$ is a Γ-typed substitution. By Lemma 3, since $\Gamma \vdash A : \tau_1$, we have $\Gamma \vdash A\sigma : \tau_2$ where $\tau_2 \preccurlyeq \tau_1$. Hence, this case also holds.

- When $A_1 = A'[\,]$, then by Lemma 2 there exists ρ such that $\epsilon \preccurlyeq \rho$ and $\Gamma \vdash A' : \rho \to \tau_1$. The proof is similar to the previous case.
- If $A_1 = \lambda x.A'$, then by Lemma 2 there exist τ', φ' such that $\tau_1 = \tau' \to \varphi'$ and $\Gamma, x : \tau' \vdash A' : \varphi'$. By the induction hypothesis, there exist $\varphi'' \preccurlyeq \varphi'$ and $A' \twoheadrightarrow_\beta A''$ such that $\Gamma \vdash A'' : \varphi''$. By the typing rule ABS, we get $\Gamma \vdash \lambda x.A' : \tau' \to \varphi''$. Since $\tau' \to \varphi'' \preccurlyeq \tau' \to \varphi'$, this case also holds.

□

Example 2. *Now we give an example where the type of a term strictly decreases after reduction. Let* $\Gamma \vdash x : \mathsf{Real}$, $\Gamma \vdash z : \mathsf{Int}^*$, $\Gamma \vdash f : \mathsf{Int}^* \to \mathsf{Int}$ *and* $\Gamma \vdash A_i : \mathsf{Nat}$, $0 \le i \le n$. *We also have the following ordering on basic types:* $\mathsf{Nat} \preceq \mathsf{Int} \preceq \mathsf{Real}$. *Then the type of the term* $(\lambda z.(\lambda x.x)(f[z]))[A_1, \ldots A_n]$ *under* Γ *is* Real. *On the other hand, this term reduces as*

$$(\lambda z.(\lambda x.x)(f[z]))[A_1, \ldots A_n] \to_\beta$$
$$(\lambda x.x)(f[A_1, \ldots A_n]) \to_\beta f[A_1, \ldots A_n]$$

and the type of the term $f[A_1, \ldots A_n]$ *under* Γ *is* Int, *which is a subtype of* Real.

VI. Strong Normalization

An untyped term is called *strongly normalizing* if there is no infinite reduction starting from it. We denote the *set of strongly normalizing untyped terms* by SN.

We start with defining the semantics of types with respect to SN. The definition is formulated modulo equality of a single term and the singleton sequence containing that term, i.e., A and $[A]$ are identified. If this relation is $=_s$, we, in particular, have that $\mathsf{SN} =_s \{[A] \mid A \in \mathsf{SN}\}$.

Now the *semantics* $[\![.]\!]$ of a type with respect to SN is defined as follows:

- $[\![\alpha]\!] = \mathsf{SN}$.
- $[\![\epsilon]\!] = \{[\,]\}$.
- $[\![\tau \cdot \rho]\!] = [\![\tau]\!] \cdot [\![\rho]\!]$.
- $[\![\tau + \rho]\!] = [\![\tau]\!] \cup [\![\rho]\!]$.
- $[\![\tau^*]\!] = [\![\tau]\!]^*$.
- $[\![\tau_1 \to \cdots \to \tau_n \to \alpha]\!] =$
$$\{A \mid A[A^1_1, \ldots, A^1_{m_1}] \cdots [A^n_1, \ldots, A^n_{m_n}] \in [\![\alpha]\!]$$
$$\text{for all } [A^i_1, \ldots, A^i_{m_i}] \in [\![\tau_i]\!],\ 1 \le i \le n\}.$$

Definition 2. *A* saturated set S *is a subset of* SN, *satisfying the conditions below. Let* $A'_1, \ldots A'_m, A^1_1, \ldots, A^1_{n_1}, \ldots, A^k_1, \ldots, A^k_{n_k} \in \mathsf{SN}$, $m, k, n_1, \ldots n_k \ge 0$. *Then*

(1) *For all* x *and* f,
- $x[A^1_1, \ldots, A^1_{n_1}] \cdots [A^k_1, \ldots, A^k_{n_k}] \in S$ *and*
- $f[A^1_1, \ldots, A^1_{n_1}] \cdots [A^k_1, \ldots, A^k_{n_k}] \in S$.

(2) *If* $A\sigma[A^1_1, \ldots, A^1_{n_1}] \cdots [A^k_1, \ldots, A^k_{n_k}] \in S$ *with the substitution* $\sigma = \{x \mapsto [A'_1, \ldots A'_m]\}$, *then we have* $(\lambda x.A)[A'_1, \ldots A'_m][A^1_1, \ldots, A^1_{n_1}] \cdots [A^k_1, \ldots, A^k_{n_k}] \in S$.

The *set of all saturated sets* is denoted by SAT.

Lemma 4. *1)* $\mathsf{SN} \in \mathsf{SAT}$.
2) If $D \in \mathsf{SAT}^*$ *and* $R \in \mathsf{SAT}$, *then* $F \in \mathsf{SAT}$, *where* F *is defined as* $F := \{A \mid A[A_1, \ldots, A_n] \in R$ *for all* $[A_1, \ldots, A_n] \in D\}$.
3) $[\![\tau]\!] \in \mathsf{SAT}^*$ *for all types* τ.

Proof. As a shortcut, in some proofs below we write $\overline{A'}$ for $A'_1, \ldots A'_m$, $\overline{B'}$ for $B'_1, \ldots B'_m$, $\overline{A^i}$ for $A^i_1, \ldots, A^i_{n_i}$, and $\overline{B^i}$ for $B^i_1, \ldots, B^i_{n_i}$.

1) We show that the set SN is saturated. Assume $A^1_1, \ldots, A^1_{n_1}, \ldots, A^k_1, \ldots, A^k_{n_k} \in \mathsf{SN}$, $k, n_1, \ldots n_k \ge 0$. Since $x[\overline{A^1}] \cdots [\overline{A^k}] \in \mathsf{SN}$ and $f[\overline{A^1}] \cdots [\overline{A^k}] \in \mathsf{SN}$, the condition **(1)** of the definition of saturated sets is satisfied. To show the condition **(2)**, assume $A'_1, \ldots A'_m \in \mathsf{SN}$ and $A\sigma[\overline{A^1}] \cdots [\overline{A^k}] \in \mathsf{SN}$ where $\sigma = \{x \mapsto [\overline{A'}]\}$. Since $A\sigma$ is a subterm of a term in SN, we have $A\sigma \in \mathsf{SN}$ and, hence, $A \in \mathsf{SN}$. Therefore, any reduction inside $(\lambda x.A)[\overline{A'}][\overline{A^1}] \cdots [\overline{A^k}]$ must terminate. Let $(\lambda x.B)[\overline{B'}][\overline{B^1}] \cdots [\overline{B^k}]$ be the result of such a reduction. Its contraction gives $B\sigma'[\overline{B^1}] \cdots [\overline{B^k}]$ where $\sigma' = \{x \mapsto [\overline{B'}]\}$. On the other hand, since B is a reduct of A, $\overline{B'}$ is a reduct of $\overline{A'}$, and $\overline{B^i}$ is a reduct of $\overline{A^i}$ for all $1 \le i \le k$, by Lemma 1 we get that $B\sigma'[\overline{B^1}] \cdots [\overline{B^k}]$ is a reduct of $A\sigma[\overline{A^1}] \cdots [\overline{A^k}]$. Since the latter is in SN, we conclude that $B\sigma'[\overline{B^1}] \cdots [\overline{B^k}] \in \mathsf{SN}$, which implies that $(\lambda x.B)[\overline{B'}][\overline{B^1}] \cdots [\overline{B^k}] \in \mathsf{SN}$ and, hence, $(\lambda x.A)[\overline{A'}][\overline{A^1}] \cdots [\overline{A^k}] \in \mathsf{SN}$.
2) Assume the sets $D \in \mathsf{SAT}^*$ and $R \in \mathsf{SAT}$, and show $F \in \mathsf{SAT}$. Since $D \in \mathsf{SAT}^*$, its elements are sequences of elements from SN, i.e., for all $[A_1, \ldots A_n] \in D$, $n \ge 0$, we have $[A_1, \ldots A_n] \in \mathsf{SN}^*$. In particular, we get that for all A and $A_1, \ldots, A_n \in \mathsf{SN}$, if $A[A_1, \ldots, A_n] \in R$, then $A[A_1, \ldots, A_n] \in \mathsf{SN}$, which implies that all such A's are from SN. That means, $F \subseteq \mathsf{SN}$.
To show $F \in \mathsf{SAT}$, we fix $A'_1, \ldots A'_m, A^1_1, \ldots, A^1_{n_1}, \ldots, A^k_1, \ldots, A^k_{n_k} \in \mathsf{SN}$, $m, k, n_1, \ldots n_k \ge 0$, and show that the conditions **(1)** and **(2)** of Definition 2 are satisfied. Since $R \in \mathsf{SAT}$, we have
- $x[\overline{A^1}] \cdots [\overline{A^k}][A_1, \ldots, A_n] \in R$ and
- $f[\overline{A^1}] \cdots [\overline{A^k}][A_1, \ldots, A_n] \in R$

for all x, f, and $[A_1, \ldots, A_n] \in D$. But then, by the definition of F, we get that

- $x[\overline{A^1}] \cdots [\overline{A^k}] \in F$ and
- $f[\overline{A^1}] \cdots [\overline{A^k}] \in F$,

which implies that **(1)** is satisfied.

To show **(2)**, we take $\sigma = \{x \mapsto [\overline{A'}]\}$, assume $A'\sigma[\overline{A^1}] \cdots [\overline{A^k}] \in F$ for some fixed A', and show $(\lambda x.A')[\overline{A'}][\overline{A^1}] \cdots [\overline{A^k}] \in F$. From the assumption $A'\sigma[\overline{A^1}] \cdots [\overline{A^k}] \in F$, by the definition of the set F, we get that $A'\sigma[\overline{A^1}] \cdots [\overline{A^k}][A_1, \ldots, A_n] \in R$ for all $[A_1, \ldots, A_n] \in D$, $n \geq 0$. Since $R \in \mathsf{SAT}$, from $A'\sigma[\overline{A^1}] \cdots [\overline{A^k}][A_1, \ldots, A_n] \in R$, by Definition 2, we get $(\lambda x.A')[\overline{A'}][\overline{A^1}] \cdots [\overline{A^k}][A_1, \ldots, A_n] \in R$ for all $[A_1, \ldots, A_n] \in D$, $n \geq 0$. Using the definition of F once again, we conclude that the membership $(\lambda x.A')[\overline{A'}][\overline{A^1}] \cdots [\overline{A^k}] \in F$ holds.

3) We prove it by induction on the generation of τ.
   a) When $\tau = \alpha$, by the definition of $[\![.]\!]$ we have $[\![\tau]\!] = \mathsf{SN}$. By the first item of this lemma, we then get $[\![\tau]\!] \in \mathsf{SAT}$ and, hence, $[\![\tau]\!] \in \mathsf{SAT}^*$.
   b) The cases when τ is ϵ, $\rho_1 \cdot \rho_2$, $\rho_1 + \rho_2$, and ρ^* follow from the induction hypothesis and the definition of regular sets.
   c) Let $\tau = \rho \to \varphi$. By the definition of type semantics we have $[\![\rho \to \varphi]\!] = \{A \mid A[A_1, \ldots, A_n] \in [\![\varphi]\!]$ for all $[A_1, \ldots, A_n] \in [\![\rho]\!]\}$. By the induction hypothesis, we have $[\![\varphi]\!] \in \mathsf{SAT}^*$ and $[\![\rho]\!] \in \mathsf{SAT}^*$. From $[\![\varphi]\!] \in \mathsf{SAT}^*$, by the definition of $[\![.]\!]$, we can easily conclude that $[\![\varphi]\!] \in \mathsf{SAT}$. By case 2 of Lemma 4 we get $[\![\rho \to \varphi]\!] \in \mathsf{SAT}$. Hence, also in this case $[\![\tau]\!] \in \mathsf{SAT}^*$.

□

In the technical lemmas below, we need a mapping that relates nominal type interpretation with type semantics.

Definition 3. *We define the mapping $\mathcal{M}$ on nominal atoms, sequences of nominal atoms, and sets of sequences of nominal atoms, in the following way:*

$$\mathcal{M}(\alpha) := [\![\alpha]\!].$$
$$\mathcal{M}(\langle D_1, \ldots, D_n, R\rangle) := \{A \mid A[A^1_1, \ldots, A^1_{m_1}] \cdots [A^n_1, \ldots, A^n_{m_n}] \in \mathcal{M}(R) \text{ for all } [A^i_1, \ldots, A^i_{m_i}] \in \mathcal{M}(D_i),\ 1 \leq i \leq n\}.$$
$$\mathcal{M}([\,]) := \{[\,]\}.$$
$$\mathcal{M}([a_1, \ldots, a_n]) := \mathcal{M}(a_1) \cdot \cdots \cdot \mathcal{M}(a_n),\ n > 0.$$
$$\mathcal{M}(S) := \cup_{\tilde{s} \in S} \mathcal{M}(\tilde{s}).$$

Lemma 5. *For all τ, $\mathcal{M}((\!|\tau|\!)) = [\![\tau]\!]$.*

Proof. By structural induction on τ.

- $\tau = \alpha$. Then $\mathcal{M}((\!|\alpha|\!)) = \mathcal{M}(\{[\alpha]\}) = \mathcal{M}([\alpha]) = \mathcal{M}(\alpha) = [\![\alpha]\!]$.
- $\tau = \epsilon$. Then $\mathcal{M}((\!|\epsilon|\!)) = \mathcal{M}(\{[\,]\}) = \mathcal{M}([\,]) = \{[\,]\} = [\![\epsilon]\!]$.
- $\rho = \rho_1 \cdot \rho_2$. Then $\mathcal{M}((\!|\rho_1 \cdot \rho_2|\!)) = \mathcal{M}((\!|\rho_1|\!) \cdot (\!|\rho_2|\!))$. From the definition of $\mathcal{M}$, we can easily see that $\mathcal{M}((\!|\rho_1|\!) \cdot (\!|\rho_2|\!)) = \mathcal{M}((\!|\rho_1|\!)) \cdot \mathcal{M}((\!|\rho_2|\!))$. By the induction hypothesis, $\mathcal{M}((\!|\rho_1|\!)) \cdot \mathcal{M}((\!|\rho_2|\!)) = [\![\rho_1]\!] \cdot [\![\rho_2]\!] = [\![\rho_1 \cdot \rho_2]\!]$. Hence, $\mathcal{M}((\!|\rho_1 \cdot \rho_2|\!)) = [\![\rho_1 \cdot \rho_2]\!]$.
- $\rho = \rho_1 + \rho_2$. Then $\mathcal{M}((\!|\rho_1 + \rho_2|\!)) = \mathcal{M}((\!|\rho_1|\!) \cup (\!|\rho_2|\!)) = \mathcal{M}((\!|\rho_1|\!)) \cup \mathcal{M}((\!|\rho_2|\!))$. By the induction hypothesis, $\mathcal{M}((\!|\rho_1|\!)) \cup \mathcal{M}((\!|\rho_2|\!)) = [\![\rho_1]\!] \cup [\![\rho_2]\!] = [\![\rho_1 + \rho_2]\!]$. Hence, $\mathcal{M}((\!|\rho_1 + \rho_2|\!)) = [\![\rho_1 + \rho_2]\!]$.
- $\tau = \rho^*$. Then $\mathcal{M}((\!|\rho^*|\!)) = \mathcal{M}((\!|\rho|\!)^*)$. From Definition 3, we can easily see that $\mathcal{M}((\!|\rho|\!)^*) = \mathcal{M}((\!|\rho|\!))^*$. By the induction hypothesis, $\mathcal{M}((\!|\rho|\!))^* = [\![\rho]\!]^* = [\![\rho^*]\!]$. Hence $\mathcal{M}((\!|\rho^*|\!)) = [\![\rho^*]\!]$.
- $\tau = \rho_1 \to \cdots \to \rho_n \to \alpha$, $n \geq 1$. Then

$$\begin{aligned}
&\mathcal{M}((\!|\rho_1 \to \cdots \to \rho_n \to \alpha|\!)) = \\
&\mathcal{M}(\{[\langle (\!|\rho_1|\!), \ldots, (\!|\rho_n|\!), (\!|\alpha|\!)\rangle]\}) = \\
&\mathcal{M}([\langle (\!|\rho_1|\!), \ldots, (\!|\rho_n|\!), (\!|\alpha|\!)\rangle]) = \\
&\mathcal{M}(\langle (\!|\rho_1|\!), \ldots, (\!|\rho_n|\!), (\!|\alpha|\!)\rangle) = \\
&\{A \mid A[A^1_1, \ldots, A^1_{m_1}] \cdots [A^n_1, \ldots, A^n_{m_n}] \in \mathcal{M}((\!|\alpha|\!)) \\
&\qquad \text{for all } [A^i_1, \ldots, A^i_{m_i}] \in \mathcal{M}((\!|\rho_i|\!)),\ 1 \leq i \leq n\} = \\
&\text{(by the induction hypothesis)} \\
&\{A \mid A[A^1_1, \ldots, A^1_{m_1}] \cdots [A^n_1, \ldots, A^n_{m_n}] \in [\![\alpha]\!] \\
&\qquad \text{for all } [A^i_1, \ldots, A^i_{m_i}] \in [\![\rho_i]\!],\ 1 \leq i \leq n\} = \\
&[\![\rho_1 \to \cdots \to \rho_n \to \alpha]\!].
\end{aligned}$$

□

The next lemma, which shows that our subtype ordering is sound with respect to type semantics, will be used to prove the soundness theorem.

Lemma 6. *For all τ_1 and τ_2, if $\tau_1 \preccurlyeq \tau_2$, then $[\![\tau_1]\!] \subseteq [\![\tau_2]\!]$.*

Proof. By Definition 1 and Lemma 5, this lemma is equivalent to the following statement: For all τ_1 and τ_2, if $(\!|\tau_1|\!) \preceq (\!|\tau_2|\!)$, then $\mathcal{M}((\!|\tau_1|\!)) \subseteq \mathcal{M}((\!|\tau_2|\!))$. We prove it by structural induction on τ_1.

- $\tau_1 = \alpha$. Then $(\!|\tau_1|\!) = \{[\alpha]\}$ and there exists $[\beta] \in (\!|\tau_2|\!)$ such that $\alpha \preceq \beta$. Therefore, we have: $\mathcal{M}((\!|\tau_1|\!)) = [\![\alpha]\!] = \mathsf{SN} = [\![\beta]\!] \subseteq \mathcal{M}((\!|\tau_2|\!))$.
- $\tau_1 = \epsilon$. Then $(\!|\tau_1|\!) = \{[\,]\}$ and $[\,] \in (\!|\tau_2|\!)$. Therefore, $\mathcal{M}((\!|\tau_1|\!)) = \{[\,]\} \subseteq \mathcal{M}((\!|\tau_2|\!))$.
- τ_1 is a compound type (i.e., concatenation, choice, star, or arrow). Then from $(\!|\tau_1|\!) \preceq (\!|\tau_2|\!)$, by Definition 1, we know that for each sequence of nominal atoms $\tilde{s}_1 \in (\!|\tau_1|\!)$ there is a sequence of nominal atoms $\tilde{s}_2 \in (\!|\tau_2|\!)$ such that $\tilde{s}_1 \preceq \tilde{s}_2$. From the definition of $\mathcal{M}$ it follows that $\tilde{s} \in (\!|\tau|\!)$ implies $\mathcal{M}(\tilde{s}) \subseteq \mathcal{M}((\!|\tau|\!))$ for any $\tilde{s}$ and τ. Hence, if we show that $\tilde{s}_1 \preceq \tilde{s}_2$ implies $\mathcal{M}(\tilde{s}_1) \subseteq \mathcal{M}(\tilde{s}_2)$, then the lemma will be proved, because we will have
  - for all $\tilde{s}_1 \in (\!|\tau_1|\!)$ there is $\tilde{s}_2 \in (\!|\tau_2|\!)$ such that $\mathcal{M}(\tilde{s}_1) \subseteq \mathcal{M}(\tilde{s}_2)$,
  - $\cup_{\tilde{s}_1 \in (\!|\tau_1|\!)} \mathcal{M}(\tilde{s}_1) = \mathcal{M}((\!|\tau_1|\!))$,
  - $\cup_{\tilde{s}_2 \in (\!|\tau_2|\!),\, \tilde{s}_1 \preceq \tilde{s}_2 \text{ for some } \tilde{s}_1 \in (\!|\tau_1|\!)} \mathcal{M}(\tilde{s}_2) \subseteq \mathcal{M}((\!|\tau_2|\!))$.

Hence, we need to prove that $\tilde{s}_1 \preceq \tilde{s}_2$ implies $\mathcal{M}(\tilde{s}_1) \subseteq \mathcal{M}(\tilde{s}_2)$. By the definitions of $\preceq$, $\mathcal{M}$, and concatenation of set of sequences, this statement reduces to proving two statements:

1) $\alpha \preceq \beta$ implies $\mathcal{M}(\alpha) \subseteq \mathcal{M}(\beta)$, which holds trivially, because $\mathcal{M}(\alpha) = \mathcal{M}(\beta) = \mathsf{SN}$.
2) $a \preceq a'$ implies $\mathcal{M}(a) \subseteq \mathcal{M}(a')$, where $a = \langle (\!|\rho_1|\!), \dots, (\!|\rho_n|\!), (\!|\alpha|\!) \rangle$ and $a' = \langle (\!|\rho'_1|\!), \dots, (\!|\rho'_n|\!), (\!|\alpha'|\!) \rangle$. We prove this statement now.

From $\langle (\!|\rho_1|\!), \dots, (\!|\rho_n|\!), (\!|\alpha|\!) \rangle \preceq \langle (\!|\rho'_1|\!), \dots, (\!|\rho'_n|\!), (\!|\alpha'|\!) \rangle$, by Definition 1, we have $(\!|\rho'_i|\!) \preceq (\!|\rho_i|\!)$ for all $1 \leq i \leq n$ and $(\!|\alpha|\!) \preceq (\!|\alpha'|\!)$. By the induction hypothesis, we get $\mathcal{M}((\!|\rho'_i|\!)) \subseteq \mathcal{M}((\!|\rho_i|\!))$ for all $1 \leq i \leq n$. Besides, $\mathcal{M}((\!|\alpha|\!)) = \mathcal{M}((\!|\alpha'|\!))$. From these relations, by Definition 3, we obtain

$$\begin{aligned}
&\mathcal{M}(\langle (\!|\rho_1|\!), \dots, (\!|\rho_n|\!), (\!|\alpha|\!) \rangle) = \\
&\{A \mid A[A^1_1, \dots, A^1_{m_1}] \cdots [A^n_1, \dots, A^n_{m_n}] \in \mathcal{M}((\!|\alpha|\!)) \\
&\qquad \text{for all } [A^i_1, \dots, A^i_{m_i}] \in \mathcal{M}((\!|\rho_i|\!)),\ 1 \leq i \leq n\} \subseteq \\
&\{A \mid A[A^1_1, \dots, A^1_{m_1}] \cdots [A^n_1, \dots, A^n_{m_n}] \in \mathcal{M}((\!|\alpha'|\!)) \\
&\qquad \text{for all } [A^i_1, \dots, A^i_{m_i}] \in \mathcal{M}((\!|\rho'_i|\!)),\ 1 \leq i \leq n\} = \\
&\mathcal{M}(\langle (\!|\rho'_1|\!), \dots, (\!|\rho'_n|\!), (\!|\alpha'|\!) \rangle).
\end{aligned}$$

It proves the second statement and, consequently, the lemma.

□

The core of the strong normalization proof relies on soundness of typing with respect to the semantics of types. To formally state it, we need the notion of *satisfiability*:

- A substitution σ satisfies the statement $A : \tau$, written $\sigma \models A : \tau$, if $A\sigma \in [\![\tau]\!]$.
- σ satisfies the basis Γ, written $\sigma \models \Gamma$, if $\sigma \models x : \tau$ for all $(x : \tau) \in \Gamma$.
- Γ satisfies the statement $A : \tau$, written $\Gamma \models A : \tau$, if $\sigma \models \Gamma$ implies $\sigma \models A : \tau$ for all substitutions σ.

The soundness theorem asserts that a basis satisfies all statements that can be derived from it:

Theorem 2 (Soundness)**.** *If* $\Gamma \vdash A : \tau$ *then* $\Gamma \models A : \tau$.

Proof. By induction on the derivation of $A : \tau$.

- $A = x$. By Lemma 2, $\Gamma \vdash A : \tau$ implies $(x : \tau) \in \Gamma$. To prove $\Gamma \models A : \tau$, we assume $\sigma \models \Gamma$ for an arbitrary σ. From this assumption and $(x : \tau) \in \Gamma$, by the definition of satisfiability, we get $\sigma \models A : \tau$ and, hence $\Gamma \models A : \tau$.
- $A = f$. By Lemma 2, $\Gamma \vdash A : \tau$ implies $\tau = \mathit{type}(f)$. By definition, $\mathit{type}(f)$ is either a basic type α, or a functional type $\rho_1 \to \cdots \to \rho_n \to \alpha$, for some α and $\rho_1, \dots, \rho_n$, $n > 0$. If $\mathit{type}(f) = \alpha$, then $[\![\mathit{type}(f)]\!] = \mathsf{SN}$ and f belongs to it, because f is strongly normalizable. If $\mathit{type}(f) = \rho_1 \to \cdots \to \rho_n \to \alpha$, then to show that $f \in [\![\rho_1 \to \cdots \to \rho_n \to \alpha]\!]$ we reason as follows: Take arbitrary $[\overline{A^1}] \in [\![\rho_1]\!], \dots, [\overline{A^n}] \in [\![\rho_n]\!]$. By Lemma 4, $[\overline{A^1}] \in \mathsf{SAT}^*, \dots, [\overline{A^n}] \in \mathsf{SAT}^*$. Therefore, all terms that appear in $[\overline{A^1}], \dots, [\overline{A^n}]$ are strongly normalizable. Therefore, $f[\overline{A^1}] \cdots [\overline{A^n}] \in \mathsf{SN}$. On the other hand, $\mathsf{SN} = [\![\alpha]\!]$, i.e., we obtained that $f[\overline{A^1}] \cdots [\overline{A^n}] \in [\![\alpha]\!]$ for an arbitrary $[\overline{A^n}] \in [\![\rho_n]\!]$. By the definition of type semantics, it implies $f[\overline{A^1}] \cdots [\overline{A^{n-1}}] \in [\![\rho_n \to \alpha]\!]$. Repeating this argument $n-1$ times leads to the result that for an arbitrary $[\overline{A^1}] \in [\![\rho_1]\!]$ we have $f[\overline{A^1}] \in [\![\rho_2 \to \cdots \to \rho_n \to \alpha]\!]$. By the definition of type semantics, we conclude that $f \in [\![\rho_1 \to \cdots \to \rho_n \to \alpha]\!] = [\![\mathit{type}(f)]\!]$.
Hence, we got $A \in [\![\tau]\!]$. By the definition of satisfiability, $\Gamma \models A : \tau$ holds.
- $A = B[B_1, \dots, B_n]$. Then $\tau = \varphi$. By Lemma 2, there exist $\rho, \rho_1, \dots, \rho_n$, $n > 0$ such that $\Gamma \vdash B : \rho \to \varphi$, $\Gamma \vdash B_1 : \rho_1, \dots, \Gamma \vdash B_n : \rho_n$, and $\rho_1 \cdot \dots \cdot \rho_n \preccurlyeq \rho$. By the induction hypothesis we have $\Gamma \models B : \rho \to \varphi$, $\Gamma \models B_1 : \rho_1, \dots, \Gamma \models B_n : \rho_n$. Therefore, by the definition of satisfiability, for an arbitrary $\sigma \models \Gamma$ we know $\sigma \models B : \rho \to \varphi$, $\sigma \models B_1 : \rho_1, \dots, \sigma \models B_n : \rho_n$. That means, we have $B\sigma \in [\![\rho \to \varphi]\!]$, $B_1\sigma \in [\![\rho_1]\!], \dots, B_n\sigma \in [\![\rho_n]\!]$. Then, for the concatenation we get $[B_1\sigma, \dots B_n\sigma] \in [\![\rho_1 \cdot \dots \cdot \rho_n]\!]$. Since $\rho_1 \cdot \dots \cdot \rho_n \preccurlyeq \rho$, by Lemma 6, we have $[B_1\sigma, \dots B_n\sigma] \in [\![\rho]\!]$. From this inclusion and the fact $B\sigma \in [\![\rho \to \varphi]\!]$, by the definition of $[\![\rho \to \varphi]\!]$, we obtain $B\sigma[B_1\sigma, \dots B_n\sigma] \in [\![\varphi]\!]$. By the definition of satisfiability, we get $\sigma \models B[B_1, \dots B_n] : \varphi$. Hence, $\Gamma \models A : \tau$ holds.
- $A = B[\,]$. Similar to the previous case.
- $A = \lambda x.B$. By Lemma 2, there exist ρ and φ such that $\tau = \rho \to \varphi$ and $\Gamma, x : \rho \vdash B : \varphi$. By the induction hypothesis, we have $\Gamma, x : \rho \models B : \varphi$. Fix a substitution σ arbitrarily and suppose $\sigma \models \Gamma$ in order to show $\sigma \models \lambda x.B : \rho \to \varphi$. That is, we must show $(\lambda x.B)\sigma[A'_1, \dots, A'_n] \in [\![\varphi]\!]$ for all $[A'_1, \dots, A'_n] \in [\![\rho]\!]$. We assume $[A'_1, \dots, A'_n] \in [\![\rho]\!]$. Let σ' be the substitution obtained by the composition $\sigma\{x \mapsto [A'_1, \dots, A'_n]\}$. Then $\sigma' \models \Gamma, x : \rho$ and $B\sigma' \in [\![\varphi]\!]$. Note that $B\sigma' = B\sigma\{x \mapsto [A'_1, \dots, A'_n]\} = (\lambda x.B)\sigma[A'_1, \dots, A'_n]$, which finishes the proof.

□

From Theorem 2 and the definition of satisfiability, we get strong normalization:

Theorem 3 (Strong Normalization)**.** *If* $\Gamma \vdash A : \tau$*, then* $A \in [\![\tau]\!]$.

Proof. Analogous to the proof of strong normalization in simply typed lambda calculus, see, e.g., [3, Theorem 4.3.6]. □

VII. Conclusion

We have formally developed a type system based on regular expressions and a semantic (set-theoretic) subtyping relation. As we wrote in the introduction, our approach is quite generic and foundational: not all the particular features of programming languages dealing with regular expressions are considered since we choose to define a core typed functional calculus. With this approach we provide a general model which can be used to study important semantic definitions, such as strong normalization and subject reduction.

ACKNOWLEDGMENT

This research has been partially supported by LIACC through Programa de Financiamento Plurianual of the Fundação para a Ciência e Tecnologia (FCT), by Rustaveli National Science Foundation under the project FR/325/4-120/14, and by the Austrian Science Fund (FWF) under the project SToUT (P 24087-N18).

REFERENCES

[1] Valentin M. Antimirov. Rewriting regular inequalities (extended abstract). In Horst Reichel, editor, *Fundamentals of Computation Theory, 10th International Symposium, FCT '95, Dresden, Germany, August 22-25, 1995, Proceedings*, volume 965 of *Lecture Notes in Computer Science*, pages 116–125. Springer, 1995.

[2] Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, Amsterdam, 2nd edition, 1984.

[3] Henk Barendregt. Lambda calculi with types. In Samson Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science, volume 2: Background: Computational Structures*, pages 117–309. Clarendon Press, Oxford, 1992.

[4] Véronique Benzaken, Giuseppe Castagna, and Alain Frisch. CDuce: an XML-centric general-purpose language. In Colin Runciman and Olin Shivers, editors, *ICFP*, pages 51–63. ACM, 2003.

[5] Bruno Buchberger. Mathematica as a rewrite language. In Tetsuo Ida, Atsushi Ohori, and Masato Takeichi, editors, *2nd Fuji International Workshop on Functional and Logic Programming, November 1-4, 1996, Shonan Village Center, Japan*, pages 1–13. World Scientific, 1996.

[6] Giuseppe Castagna, Giorgio Ghelli, and Giuseppe Longo. A calculus for overloaded functions with subtyping. *Inf. Comput.*, 117(1):115–135, 1995.

[7] Jorge Coelho and Mário Florido. Xcentric: A logic-programming language for XML processing. In *PLAN-X 2007, Programming Language Technologies for XML, An ACM SIGPLAN Workshop colocated with POPL 2007, Nice, France, January 20, 2007*, pages 93–94, 2007.

[8] Besik Dundua, Mário Florido, Temur Kutsia, and Mircea Marin. Constraint logic programming for hedges: A semantic reconstruction. In Michael Codish and Eijiro Sumii, editors, *Functional and Logic Programming - 12th International Symposium, FLOPS 2014, Kanazawa, Japan, June 4-6, 2014. Proceedings*, volume 8475 of *Lecture Notes in Computer Science*, pages 285–301. Springer, 2014.

[9] Besik Dundua, Mário Florido, Temur Kutsia, and Mircea Marin. CLP(H): Constraint logic programming for hedges. *Theory and Practice of Logic Programming*, 2015. To appear.

[10] Alain Frisch, Giuseppe Castagna, and Véronique Benzaken. Semantic subtyping: Dealing set-theoretically with function, union, intersection, and negation types. *J. ACM*, 55(4):19:1–19:64, September 2008.

[11] Haruo Hosoya and Benjamin C. Pierce. Regular expression pattern matching for XML. *J. Funct. Program.*, 13(6):961–1004, 2003.

[12] Temur Kutsia and Mircea Marin. Regular expression order-sorted unification and matching. *J. Symb. Comput.*, 67:42–67, 2015.

[13] Peter J. Landin. Correspondence between ALGOL 60 and church's lambda-notation: part I. *Commun. ACM*, 8(2):89–101, 1965.

[14] Peter J. Landin. A correspondence between ALGOL 60 and church's lambda-notations: Part II. *Commun. ACM*, 8(3):158–167, 1965.

[15] Peter J. Landin. A lambda calculus approach. In L. Fox, editor, *Advances in Programming and Non-Numerical Computation, Symposium Publications Division*, chapter 5, pages 97–141. Pergamon Press, 1966.

[16] Peter J. Landin. The next 700 programming languages. *Commun. ACM*, 9(3):157–166, 1966.

[17] John McCarthy. Towards a mathematical science of computation. In *IFIP Congress*, pages 21–28, 1962.

[18] John McCarthy. A basis for a mathematical theory of computation. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 33–69. North Holland, 1963.

[19] Benjamin C. Pierce. *Types and programming languages*. MIT Press, 2002.

[20] Martin Sulzmann and Kenny Zhuo Ming Lu. Xhaskell - adding regular expression types to haskell. In Olaf Chitil, Zoltán Horváth, and Viktória Zsók, editors, *IFL*, volume 5083 of *LNCS*, pages 75–92. Springer, 2007.

[21] Stephen Wolfram. *The Mathematica book (5. ed.)*. Wolfram-Media, 2003.