Term Rewriting with Variable Binding: An Initial Algebra Approach

Makoto Hamana
Department of Computer Science, Gunma University, Japan
hamana@cs.gunma-u.ac.jp

ABSTRACT

We present an extension of first-order term rewriting systems, which involves variable binding in the term language. We develop the systems called binding term rewriting systems (BTRSs) in a stepwise manner; firstly we present the term language, then formulate equational logic, and finally define rewrite systems by two styles: rewrite logic and pattern matching styles. The novelty of this development is that we follow an initial algebra approach in an extended notion of Σ-algebras in various functor categories. These are based on Fiore-Plotkin-Turi's presheaf semantics of variable binding and Lüth-Ghani's monadic semantics of term rewriting systems. We characterise the terms, equational logic and rewrite systems for BTRSs are initial algebras in suitable categories. Finally, we discuss our design choice of BTRSs from a semantic perspective.

Categories and Subject Descriptors

D.3.1 [**Programming Laguages**]: Formal Definitions and Theory—*syntax,semantics*; F.3.2 [**Logics and Meaning of Programs**]: Semantics of Programming Languages—*Algebraic approaches to semantics,Denotational semantics*; F.4.2 [**Grammars and Other Rewriting Systems**]

General Terms

Theory, Languages

Keywords

Term rewriting systems, abstract syntax with variable binding, higher-order abstract syntax, initial algebra semantics, monadic semantics of term rewriting systems

1. INTRODUCTION

The notion of term rewriting and its formal system of the first-order term rewriting systems (TRSs) are nowadays a fundamental tool in theoretical computer science to analyse properties of systems involving rewriting. Higher-order extension of TRSs are known as several formats: major representatives are Klop's Combinatory Reduction Systems [21], Nipkow's Higher-order Rewrite Systems [27], and Blanqui, Okada and Jouannaud's systems [3, 2]. We generally call them HORSs here. An expected application of HORSs is to investigate the metatheory of systems involving variable binding mechanism [2] like logic programming language λProlog [26] and proof assistants such as Isabelle, Coq or LEGO.

The theory of TRSs has many useful notions and techniques on reasoning and proving the properties of rewriting (especially about termination [37]). Hence, it is quite natural to expect importing such notions and techniques developed for TRSs into the world of HORSs.

However, it is not very smoothly done; sometimes it needs new ideas (e.g. termination [19]), or sometimes naive extension fails (e.g. modularity of confluence: two disjoint union of confluent HORSs is not confluent in general, unlike in the case of TRSs), and no uniform way of lifting notions of TRSs to HORSs is known. It seems that a fundamental difficulty comes from the mixture of first-order rewriting and implicit $\beta(\eta)$-reduction.

So, it is interesting to seek a reasonable extension of TRSs not sticking the notion of (full) higher-order, but still having the notion of variable binding, and more naturally inheriting the properties of TRSs. For this aim, the following two elegant semantical work on syntax and rewriting systems are helpful:

- Fiore, Plotkin and Turi's semantics of *abstract syntax with variable binding* [10] by Σ-algebras in the functor category $\mathbf{Set}^{\mathbb{F}}$ where $\mathbb{F}$ is the category of variables and renamings,
- Lüth and Ghani's monadic semantics of *term rewriting systems* [23, 24] by Σ-algebras in the category **Pre** of preorders.

These two results use the notion of Σ-algebras in different categories. Hence, combining these two, a natural idea arises: algebras in the functor category $\mathbf{Pre}^{\mathbb{F}}$ must model the notion of *term rewriting with variable binding*. This is a good guide to seek what is a correct extension of TRSs having the notion of variable binding. This paper aims to develop such systems called *binding term rewriting systems* (BTRSs). Our development is highly motivated by algebraic semantics along this idea. The main methodology is to explicitly define syntactic systems as *initial algebras* in Σ-algebras in various categories. We use the initial algebra property as a criteria whether the defined term language, equational logic and rewrite systems are "correct" ones from the viewpoint of algebraic semantics. Because if it can be shown to be an initial algebra, it admits a compositional interpretation by structural recursion. So our method is an initial algebra approach to rewriting systems.

By some analysis, we will find that the category $(\mathbf{Pre}^{\mathbb{I}})^{\mathbb{T}}$ will be more suitable than $\mathbf{Pre}^{\mathbb{F}}$ (cf. Section 6.1), where $\mathbb{I}$ is the category of abstract variables and *injective* renamings and $\mathbb{T}$ is a discrete category of types. Then, suitable class of Σ-algebras in $(\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$ can be a

PPDP'03, August 27–29, 2003, Uppsala, Sweden.

semantics of such systems and the initial algebras can characterise syntax and rewrite relation of binding term rewriting systems.

This paper is organised as follows. We first define a term language involving variable binding in Section 2. We then give an algebraic semantics of the term language in Section 3. In Section 4, we define equational logic using the term language and give its algebraic semantics. In Section 5, we define binding term rewriting systems. In Section 6, we discus our design choice of syntax and semantics from a semantic perspective, and also mention related work and future work.

2. TERM LANGUAGE

First, we give an intuitive explanation of our term language. The term language we use is originally developed by Plotkin [30] and used in [13]. In this paper, we present a slight simplified version of it.

We use two kinds of identifiers, *variables* and *names* for the language. Names are a kind of object-level variables for some intended object system and are only boundable identifiers by binders (variables are not boundable). Our term language is typed and also manages the information called *stages*. A stage expresses a set of *all possible free names* in a term. If we write

$$x : \sigma : m \vdash t : \sigma : n,$$

for a type judgment, this means that the set n contains all free names occurring in t. So, n can be considered as another typing context for another class of identifiers "names".

Also, the variable x in the typing context has the stage m, which expresses all possible free names of a term that can be substituted for x. This information is not an artificial ingredient, which often implicitly appears in a mathematical context of computer science or logic. For example, when we write a lambda term

$$\lambda a.M$$

in a certain mathematical context, we usually implicitly assume that the term M (may) contains the variable a (if not, we explicitly write the condition like $a \notin \mathsf{FV}(M)$). We treat a as a *name* and M as a *variable*, and the containment of a in M is expressed in the typing context as the type judgment like

$$M : \iota : \{a\} \vdash \mathsf{lam}([a]M) : \iota : \emptyset.$$

Here, we also treat λ as a function symbol lam taking a name abstraction term. And since we will assume that the names are natural numbers, formally a must be written as 1 in our syntax.

The aim of our term language and equational logic is to formalise this kind of system and description (e.g. for the lambda calculus) as an object system. And then rewrite systems using this term language can be computation rules involving variable binding. Throughout this paper, we use the example of lambda calculus as a typical system involving variable binding. Looking at Examples 1,2,3 might help to get intuition of this term language.

We notice that our development is *not* only for the lambda calculus. In this paper, we aim to develop a general but small as possible language that can treat variable binding and establish an algebraic semantics uniformly.

Next we formally define this term language.

2.1 Syntax

Variables and Names. We assume two classes of identifiers, variables and names. We use $x, y, \ldots$ to range over variables. A *name* is a natural number[1] in $\mathbb{N}$, which is used for a boundable identifier. We use $m, n, ..$ to range over names and often write $n = \{1, \ldots, n\}$ for a finite set of names.

Types. Types are defined by

$$\tau ::= \iota \mid [\mathsf{N}]\tau$$

where ι is the base type and $[\mathsf{N}]\tau$ is the type of name abstraction. The set of all types is denoted by Ty (note: N is not a base type in this syntax).

Signature. A *binding signature* Σ is a set of function symbols together with an arity function assigning to each function symbol f, a finite sequence of types as the source and the base type ι as the target. This is denoted by $f : \tau_1, \ldots, \tau_n \to \iota$. In case of $n = 0$, a function symbol $f : \iota$ is regarded as a constant. We denote a set of all function symbols having the target type σ by Σ_σ.

Terms. Terms are defined by

$$t ::= f(t_1, \ldots, t_l) \mid [n]t \mid \xi x \mid \mathsf{nam}(n)$$
$$\xi ::= [1 := a_1, \ldots, n := a_n]$$

where $f \in \Sigma$ is a function symbol, a's are names. A term $[n]t$ is an *abstraction*, $\mathsf{nam}(n)$ a *name term*, and ξx a *variable with renamer* where ξ is a renamer[2]. A *renamer* is a syntactic representation of the graph of an *injective* function between finite subsets of natural numbers. If $\xi : m \to n$ is an injective function, the corresponding renamer ξ is written as $[1 := \xi(1), \ldots, m := \xi(m)]$. When writing a variable with renamer ξx, ξ may be omitted if it is an identify function. We abuse the notation $\xi : m \to n$ for the renamer and the corresponding function. Intuitive meaning of ξx is that ξ is a kind of explicit substitution, which is applied when the variable x is instantiated. A type judgment is of the form

$$X \vdash t : \sigma : n$$

where

- a *typing context* X is a finite set of *variable declarations* $\{x_1 : \sigma_1 : n_1, \ldots, x_l : \sigma_l : n_l\}$
- a *type* σ,
- a *stage* n is a natural number, which is considered as a set $\{1, \ldots, n\}$ of names.

As usual, if we write $X, x{:}\sigma{:}m \vdash t : \sigma : n$, we assume the context denotes the set $X \cup \{x{:}\sigma{:}m\}$ and $x{:}\sigma{:}m \notin X$. We may also call a term of this language a *binding term*.

Typing rules. The typing rules for terms are given in Figure 1. The $\mathsf{Ty}, \mathbb{N}$-indexed set $T(X)$ of terms is defined by $T(X)_{\tau,n} = \{t \mid X \vdash t : \tau : n\}$.

Our choice of taking a name for abstractions $[\_]\_$ is followed by the method of *de Bruijn levels* [5] (see also the example in [10] for the lambda calculus and [22]).

[1] A different more "nameful" choice is possible by taking arbitrary countably infinite set N of names and assuming a linear order on it. This nameful version of the term language is given in [15]. In this paper, rather than user-friendness of syntax, to stress the connection between the semantics and syntax closely, we choose this de Bruijn like representation of names, not only for bound names, but also for free names.

[2] This terminology is taken from the similar notion in Hashimoto and Ohori's *context calculus* [16].

Variables

$$\frac{}{X, x{:}\sigma{:}m \vdash \xi x : \sigma : n}\ \ \xi : m \to n \text{ is injective}$$

Names

$$\frac{}{X \vdash \mathsf{nam}(k) : \iota : n}\ \ 1 \le k \le n$$

Function terms

$$\frac{X \vdash t_1 : \sigma_1 : n, \ldots, X \vdash t_l : \sigma_l : n}{X \vdash f(t_1, \ldots, t_l) : \sigma : n}\ \ f : \sigma_1, \ldots, \sigma_l \to \sigma \in \Sigma$$

Abstractions

$$\frac{X \vdash t : \sigma : n+1}{X \vdash [n+1]t : [\mathsf{N}]\sigma : n}$$

Figure 1: Typing rules

2.2 Renamings and substitutions

Renaming. A *renaming* $\rho : m \to n$ is an *injective* function between sets of names. The renaming ρ is obviously extended to functions on typing contexts defined by

$$\rho(x_1 : \sigma_1 : m_1, \ldots) = x_1 : \sigma_1 : \rho(m_1), \ldots.$$

Renaming on terms. Let $X \vdash t : \sigma : m$. A renaming $\rho : m \to n$ is extended to the function $\rho^\sharp : T(X)_{\sigma,m} \to T(X)_{\sigma,n}$ on terms as follows:

$$\rho^\sharp(\xi x) = \xi' x, \quad \text{where } \xi'(i) = \rho(\xi(i)),$$
$$\rho^\sharp(f(t_1, \ldots, t_n)) = f(\rho^\sharp(t_1), \ldots, \rho^\sharp(t_n)),$$
$$\rho^\sharp([m+1]t) = [n+1]\rho^\sharp_{m+1}(t), \quad \rho^\sharp(\mathsf{nam}(k)) = \mathsf{nam}(\rho(k))$$

where $\rho_{m+1} : m+1 \to n+1$ is defined by $\rho_{m+1}(m+1) = n+1$, $\rho_{m+1}(k) = \rho(k)$ for $k \neq m+1$. Then, the following ruled is derived by the typing rules.

$\sharp$-Weakening

$$\frac{X \vdash t : \sigma : m}{X \vdash \rho^\sharp t : \sigma : n}\ \ \rho : m \to n$$

Another extension is possible. Before defining it, we note that for $\xi : m' \to m, \rho : m \to n \in \mathbb{I}$, there exist (not uniquely in general) $\rho_\xi, \xi_\rho \in \mathbb{I}$ such that

$$\begin{tikzcd} m' \arrow[r, "\xi"] \arrow[d, "\rho_\xi"'] & m \arrow[d, "\rho"] \\ n' \arrow[r, "\xi_\rho"'] & n \end{tikzcd} \tag{1}$$

commutes. Then, a renaming $\rho^\natural : T(X)_{\sigma,m} \to T(X)_{\sigma,n}$ on terms can be defined by:

$$\rho^\natural(\xi x) = \xi_\rho\ x \tag{2}$$

and the other cases are the same as $\rho^\sharp$, by just replacing $\rho^\sharp$ with $\rho^\natural$. As we saw the above diagram, the equation (2) does not uniquely define the result, so actually $\rho^\natural$ is a multi-valued function. Then, the following is a derived rule

$\natural$-Weakening

$$\frac{X \vdash t : \sigma : m}{\rho X \vdash \rho^\natural t : \sigma : n}\ \ \rho : m \to n$$

As noted, the conclusion has many possibility. This situation will be clear in the case of variable:

$$(\natural)\frac{x : \sigma : m' \vdash \xi x : \sigma : m}{x : \sigma : n' \vdash \xi_\rho x : \sigma : n}\ \rho : m \to n$$

which says how the upper judgment can be weakened by ρ, where weakening is not just applied to the stage m of the term, but also the stage m' of the variable declaration. As the diagram (1), such weakening depends on ξ_ρ. Clearly, $\sharp$-weakening is a special case of $\natural$-weakening when $n' = m'$ and $\rho_\xi = \mathrm{id}_{m'}$. In such case, ξ_ρ is uniquely determined as the definition of $\rho^\sharp$.

From these typing, the mnemonic for these two ways of extension is that $\rho^\sharp$ is "half applied renaming" because only the term part is applied, and $\rho^\natural$ is "naturally applied renaming" because both stages in typing context and term are applied.

Substitutions on terms. A *substitution* θ is a mapping which maps each variable declaration in X to some element of $T(X')$ as

$$x : \tau : n \ \mapsto\ t \in T(X')_{\tau,n}.$$

We use the shorthand notation $\{\vec{x} \mapsto \vec{t}\}$ for this. The substitution θ is extended to a $\mathsf{Ty}, \mathbb{N}$-indexed function θ^* from $T(X)$ to $T(X')$ defined by

$$\theta^*(\xi x) = \xi^\sharp \circ \theta(x) \ \ (x{:}\tau{:}m \in X),$$
$$\theta^*(f(t_1, \ldots, t_l)) = f(\theta^*(t_1), \ldots, \theta^*(t_l))$$
$$\theta^*([n]t) = [n]\theta^*_{>n}(t), \quad \theta^*(\mathsf{nam}(a)) = \mathsf{nam}(a),$$

where $\theta_{>n} : X \to T(X)$ is a substitution defined by

$$\theta_{>n}(x : \sigma : k) = t'$$

where $\theta(x{:}\sigma{:}k) = t$, and for the set $\{i_1, \ldots, i_l\}$ of all names occurring in $[\_]$-binders in t, t' is obtained from t replacing every (regardless of free or bound) name i_j with $i_j - k + n$.

Roughly speaking, $\theta_{>n}(t)$ gives a term $\theta(t)$, in which all binders are changed to be greater than n. This is needed to make the substituted term followed by the method of de Bruijn levels. Hereafter, we omit the superscript $*$ from substitutions.

3. ALGEBRAIC INTERPRETATION

In this section, we interpret the term language in Σ-algebras. This is based on Fiore, Plotkin and Turi's semantics [10] extended with type-indexing.

Let $\mathbb{I}$ be the category which has finite sets $n = \{1, \ldots, n\}$ (n is possibly 0) as objects, and *injective functions* between them as morphisms $m \to n$. We also call $n \in \mathbb{I}$ a *stage*. And let $\mathbb{T}$ be a discrete category of types with the objects $\mathsf{Ty} \cup \{\mathsf{N}\}$. In algebraic semantics, we include N as the type of names.

Also, we use the following categories: the category **Eq** has the objects as equivalence relations and the morphisms functions preserving such equivalence relation. The category **Pre** has the objects as preorders $\le_X$ on X (or equivalently, preordered sets $(X, \le_X)$), and the morphisms $f : \le_X \to \le_Y$ preorder-preserving functions (i.e. $x \le_X y \ \Rightarrow\ f(x) \le_Y f(y)$).

We use algebras in the various functor categories $(\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$, $(\mathbf{Eq}^{\mathbb{I}})^{\mathbb{T}}$, $(\mathbf{Pre}^{\mathbb{I}})^{\mathbb{T}}$ to model the term language, equational logic and rewriting systems for binding terms, respectively.

3.1 Σ-Algebras

For uniform treatment, we define a Σ-algebra by a slightly general form using a signature functor.

Definition 1. Let C be a category **Set**, **Eq** or **Pre**. For a binding signature Σ, a *signature functor* $\Sigma : (C^{\mathbb{I}})^{\mathbb{T}} \to (C^{\mathbb{I}})^{\mathbb{T}}$ is defined by $\Sigma A = B$ where

$$B_\sigma = \coprod_{f:\sigma_1,\ldots,\sigma_l\to\sigma\in\Sigma_\sigma} \prod_{1\le i\le l} A_{\sigma_i} .$$

A Σ-*algebra* $\mathcal{A}$ *in* $(C^{\mathbb{I}})^{\mathbb{T}}$ is a pair (A, α) consisting of

- a carrier $A \in (C^{\mathbb{I}})^{\mathbb{T}}$, and
- an algebra structure $\alpha : \Sigma A \to A \in \text{arr } (C^{\mathbb{I}})^{\mathbb{T}}$ such that[3] $\alpha_\sigma = [f_{\mathcal{A}}]_{f\in\Sigma_\sigma}$ for every $\sigma \in \mathbb{T}$,

where $f_{\mathcal{A}}$ is an *operation*

$$f_{\mathcal{A}} : A_{\sigma_1}\times\cdots\times A_{\sigma_l} \to A_\sigma \ \in \text{arr } C^{\mathbb{I}}$$

defined for every function symbol $f : \sigma_1, \ldots, \sigma_l \to \sigma \in \Sigma_\sigma$.

A homomorphism $h : (A, \alpha) \to (B,\beta)$ between Σ-algebras is a morphism $h : A \to B$ in $(C^{\mathbb{I}})^{\mathbb{T}}$ such that $h \circ \alpha = \beta \circ \Sigma h$. □

The point of this definition is that $f_{\mathcal{A}}$ is not just a stage-indexed function, it is a natural transformation between presheaves, and the naturality condition imposes a uniform interpretation of the function symbol at all stages.

This definition of algebras in $(C^{\mathbb{I}})^{\mathbb{T}}$ is a generalisation of the first-order case [36], and the binding case [10], and an instance of Kelly and Power's general theory of algebras by enriched monads [20, 31], now our definition is the theory monadic over $(C^{\mathbb{I}})^{\mathbb{T}}$ and enriched over $C^{\mathbb{I}}$.

For interpreting the term language, we set C = **Set**.

Example 1. Define the signature Σ^Λ for the untyped lambda calculus by the three function symbols

$$\mathsf{nam} : \mathsf{N} \to \iota, \quad \mathsf{lam} : [\mathsf{N}]\iota \to \iota, \quad \mathsf{app} : \iota, \iota \to \iota.$$

Then the signature functor $\Sigma^\Lambda : (\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}} \to (\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$ is defined by

$$\Sigma^\Lambda A = (0,\ A_{\mathsf{N}} + A_{[\mathsf{N}]\iota} + A_\iota \times A_\iota,\ 0, \ldots)$$

where 0 is the initial object of $\mathbf{Set}^{\mathbb{I}}$ (i.e. the constant functor always returning the empty set). All the omitted components by "..." are 0's. Now, we are regarding $A \in (\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$ as $|\mathbb{T}|$-tuple of presheaves in $\mathbf{Set}^{\mathbb{I}}$ and assuming the second component of the tuple corresponds to the type ι. The reason why only the ι-component is defined is that the targets of all function symbols have the type ι. And one can easily see that the summation in this component corresponds to the source types of function symbols. For a Σ^Λ-algebra $\Lambda = (\Lambda, \alpha : \Sigma^\Lambda\Lambda \to \Lambda)$, the operations are morphisms (actually, natural transformations) in $\mathbf{Set}^{\mathbb{I}}$ as follows

$$\mathsf{nam}_\Lambda : \Lambda_{\mathsf{N}} \to \Lambda_\iota \quad \mathsf{lam}_\Lambda : \Lambda_{[\mathsf{N}]\iota} \to \Lambda_\iota \quad \mathsf{app}_\Lambda : \Lambda_\iota\times\Lambda_\iota \to \Lambda_\iota.$$

□

Our next aim is to show that the term language defined in Section 2 forms a Σ-algebra. In the term language, we have defined a special construct $\mathsf{nam}(n)$ for a name term. Now, we assume $\mathsf{nam} : \mathsf{N} \to \iota \in \Sigma$ is a function symbol as the above example. We will also formulate models of the term language as Σ-algebras. So in our typical use of Σ-algebras in $(\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$, the signature contains the function symbol nam. We also require a particular interpretation of the type N to denote the notion of names in Σ-algebras. This requirement derives the following definition.

[3] [ ] denotes a copair of coproducts.

Definition 2. If we write Σ_{N} for a signature, we assume that it has the function symbol $\mathsf{nam} : \mathsf{N} \to \iota$. Moreover, if we say Σ_{N}-*algebra* $\mathcal{A}$, the additional condition on the carrier $A_{\mathsf{N}} = \mathsf{N}$ must be satisfied, where $\mathsf{N} \in (\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$ is *the presheaf of abstract variables*[4][10] defined by

$$\mathsf{N}(n) = n, \quad \mathsf{N}(\rho) = \rho \quad \text{for } \rho \in \mathbb{I}.$$

□

3.2 Semantics of (meta)variables

Using these setting, we can form a term algebra consisting of all closed terms as a Σ_{N}-algebra. To incorporate open terms, the presheaf of variables must be defined. Let X be a set of variable declarations. From this, the type-indexed *presheaf* $\dot{X} \in (\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$ of *variables* is defined as follows:

$$\dot{X}_\sigma\ (n) = \{\xi x \mid X \vdash \xi x : \sigma : n,\ \xi : k \to n\},$$
$$\dot{X}_\sigma\ (\rho) = \rho^\sharp_{\restriction \dot{X}_\sigma(m)} \quad \text{for } \rho : m \to n \in \mathbb{I},$$

($\restriction$ denotes a restriction of function). The idea of this definition is the presheaf of variables is not just consisting of variables, it must be closed under arbitrary renaming. So, elements of it must be variables *with renamers*.

Also, $\dot{X}_\sigma\ (n)$ is consisting of all variables with renamers *derived from* the set X of variable declarations. By the typing rule of variables, the stage of such a derived variable might be larger than the one in the declaration, e.g.

$$x : \iota : 2 \vdash \xi x : \iota : n$$

where $\xi : 2 \to n \in \mathbb{I}$ and n is an arbitrary stage that is larger than 2. This means that the stage of variable declaration (in this case, 2) expresses the *minimal* stage that the variable can have.

A *valuation* $\chi : X \to M$ is a mapping which maps each variable declaration in X to some element of M as

$$x : \sigma : k \ \mapsto\ m \in M_\sigma(k).$$

This can be extended to a morphism $\dot{\chi}:\dot{X} \to M$ in $(\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$ defined by

$$\dot{\chi}_\sigma\ (n) : \dot{X}_\sigma\ (n) \to M_\sigma(n)$$
$$\dot{\chi}_\sigma\ (n)(\xi x) = M_\sigma(\xi)(\chi(x : \sigma : k))$$

where $\xi : k \to n$. We can see that $\dot{\chi}$ is natural in $n \in \mathbb{I}$ (and obviously in σ). This map $\dot{\chi}$ is a part of algebra structure map of a binding algebra defined below.

This definition clearly reflects the intuitive meaning of ξx, i.e. the valuation of ξx at a stage n is applying (or "melting") the "suspended" substitution of the renamer ξ to the value of x obtained by χ.

3.3 Binding algebras

Let C be a category **Set**, **Eq** or **Pre**. We define the functor (for stage extension) $\delta : C^{\mathbb{I}} \to C^{\mathbb{I}}$ as follows: for $L \in C^{\mathbb{I}}, n \in \mathbb{I}, \rho \in \text{arr } \mathbb{I}$,

$$(\delta L)(n) = L(n + 1), \quad (\delta L)(\rho) = L(\rho + \text{id}_1)$$

[4] Fiore, Plotkin and Turi's terminology of "variables" in [10] corresponds to our "names". So, to be exact, we are presenting "name binding" rather than "variable binding" in this paper. Also the same entity is called "atoms" in [12, 29, 30], "object variables" in [13]. Our "variables" does not appear in [10] and corresponds to the ones in [12, 29]. This kind of syntactic construct of "variables" is also often called "metavariables" (e.g. in Klop's combinatory reduction systems [21]).

and for $f : L \to M \in \mathrm{arr}\, C^{\mathbb{I}}$, the map $\delta f : \delta L \to \delta M \in \mathrm{arr}\, C^{\mathbb{I}}$ is given by

$$(\delta f)(n) = f(n+1) : L(n+1) \to M(n+1) \ (n \in \mathbb{I}).$$

Definition 3. A *typed binding algebra $\mathcal{M}$ in $(C^{\mathbb{I}})^{\mathbb{T}}$* is a Σ_{N}-algebra (M, α) in $(C^{\mathbb{I}})^{\mathbb{T}}$ equipped with natural isomorphisms

$$\mathsf{bind}_{M,\sigma} : \delta M_\sigma \xrightarrow{\cong} M_{[\mathsf{N}]\sigma}$$

for every $\sigma \in \mathsf{Ty}$. We write $(M, \alpha, \{\mathsf{bind}_{M,\sigma}\}_{\sigma\in\mathsf{Ty}})$ for a typed binding algebra.

A homomorphism $h : \mathcal{L} \to \mathcal{M}$ of typed binding algebras is a Σ_{N}-algebra homomorphism h which preserves bind, i.e.

$$\begin{tikzcd}
\delta L_\sigma \arrow[r, "\mathsf{bind}_{L,\sigma}"] \arrow[d, "\delta h_\sigma"'] & L_{[\mathsf{N}]\sigma} \arrow[d, "h_{[\mathsf{N}]\sigma}"] \\
\delta M_\sigma \arrow[r, "\mathsf{bind}_{M,\sigma}"'] & M_{[\mathsf{N}]\sigma}
\end{tikzcd}$$

commutes for every $\sigma \in \mathsf{Ty}$. Hereafter, we simply say (Σ_{N}-)*binding algebra.* □

In other words, in a binding algebra $\mathcal{M}$, the carriers satisfy

$$\delta M_\sigma \cong M_{[\mathsf{N}]\sigma}$$

by specific isomorphisms given by bind. The map bind is a key ingredient for the notion of binding in this framework of algebraic semantics, that is, binding is the way of transferring an element of type σ to that of type $[\mathsf{N}]\sigma$. This will be clear in the case of $C = \mathbf{Set}$ and $\mathcal{M} = \mathcal{T}X$ which is a term algebra defined in Section 3.4. Then, it means a transposition of a term, clearly such a transposition can be given by the typing rule of abstraction:

$$\frac{X \vdash t : \sigma : n+1}{X \vdash [n+1]t : [\mathsf{N}]\sigma : n}$$

Namely, this is the way of making binding by de Bruijn levels [10, 5, 22]. This idea brings us to the following definition of a binding algebra of terms.

3.4 Term algebra

For a binding algebra $\mathcal{M}$, the initial algebra semantics tells us that a unique homomorphism from the term algebra to $\mathcal{M}$ gives a compositional interpretation of a term in $\mathcal{M}$. We explicitly define this interpretation.

First we define the term algebra. Let X be a set of variable declarations. The *term algebra* $\mathcal{T}X = (TX, [\dot{\chi}, \gamma], \{\mathsf{bind}_{TX,\sigma}\}_{\sigma\in\mathsf{Ty}})$ is a $\dot{X} + \Sigma_{\mathsf{N}}$-binding algebra consisting of the following data:

- the carrier $TX \in (\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$ defined by
$$TX_\sigma(n) = T(X)_{\sigma,n}, \quad TX_\sigma(\rho) = \rho^\sharp \ \text{ for } \sigma \neq \mathsf{N},$$
$$TX_{\mathsf{N}}(n) = n, \quad TX_{\mathsf{N}}(\rho) = \rho$$
- the valuation $\dot{\chi}{:}\dot{X} \to TX$ defined by
$$\chi(x : \sigma : n) = [1 := 1, \dots, n := n]\ x \quad \text{for } x \in X_\sigma(n)$$
- the operations $f_{\mathcal{T}X} : TX_{\sigma_1}, \dots, TX_{\sigma_l} \to TX_\sigma$ for $f : \sigma_1, \dots, \sigma_l \to \sigma \in \Sigma_\sigma$ defined by
$$f_{\mathcal{T}X}(n)(t_1, \dots, t_l) = f(t_1, \dots, t_l)$$
- the maps $\mathsf{bind}_{TX,\sigma}(n) : TX_\sigma(n+1) \to TX_{[\mathsf{N}]\sigma}(n)$ defined by
$$\mathsf{bind}_{TX,\sigma}(n)(t) = [n+1]t.$$

It is clear that bind gives natural isomorphisms.

3.5 Interpretation as homomorphism

Let $\mathcal{M} = (M, [\chi, \alpha], \{\mathsf{bind}_{M,\sigma}\}_{\sigma\in\mathsf{Ty}})$ a $\dot{X} + \Sigma_{\mathsf{N}}$-binding algebra. We interpret the term language by the interpretation $\mathcal{M}[\![\_]\!]$. This is given by the family of functions

$$\mathcal{M}[\![\_]\!]_\sigma(n) : TX_\sigma(n) \to M_\sigma(n)$$

for $n \in \mathbb{I}$ and $\sigma \in \mathbb{T}$ defined as follows:

$$\begin{aligned}
&\mathcal{M}[\![\xi x]\!]_\sigma(n) = \dot{\chi}_\sigma\ (n)(\xi x) \quad (\xi : k \to n)\\
&\mathcal{M}[\![f(t_1, \dots, t_l)]\!]_\sigma(n) = f_{\mathcal{M}}(n)(\mathcal{M}[\![t_1]\!]_{\sigma_1}(n), \dots, \mathcal{M}[\![t_l]\!]_{\sigma_l}(n))\\
&\mathcal{M}[\![[n+1]t]\!]_{[\mathsf{N}]\sigma}(n) = \mathsf{bind}_{M,\sigma}(n)(\mathcal{M}[\![t]\!]_\sigma(n+1)).
\end{aligned}$$

From this definition, one may find that bind is roughly the operation (in an algebra) corresponding to $[\_]\_$ construct. But it cannot be a usual function symbol because of the unusual nature of transportation of the stage from $n+1$ to n. So $[\_]$-binder must be a special syntactic construct of the term language. The following is our main result on the term language.

Theorem 1. *$\mathcal{T}X$ is an initial $\dot{X}$ $+\Sigma_{\mathsf{N}}$-binding algebra, and $\mathcal{M}[\![\_]\!]$ is a unique homomorphism from $\mathcal{T}X$ to $\mathcal{M}$.*

Proof. It is obvious $\mathcal{T}X$ is a binding algebra by definition. By checking that the map $\mathcal{M}[\![\_]\!]_\sigma(n)$ is natural in $\sigma \in \mathbb{T}$ and $n \in \mathbb{I}$, $\mathcal{M}[\![\_]\!]$ is a homomorphism. For uniqueness, let $\mathcal{L}$ be another $\dot{X}$ $+\Sigma_{\mathsf{N}}$-binding algebra and consider a homomorphism $h : \mathcal{L} \to \mathcal{M}$. We can show the following

$$\begin{tikzcd}
\mathcal{T}X \arrow[r, "\mathcal{L}[\![_]\!]"] \arrow[dr, "\mathcal{M}[\![_]\!]"'] & \mathcal{L} \arrow[d, "h"] \\
& \mathcal{M}
\end{tikzcd}$$

commutes by induction on terms. Then take $\mathcal{L} = \mathcal{T}X$. □

This result not just says we can interpret the term language by a Σ-algebra, this also show the definition of term language and associated renaming ($\rho^\sharp$) on terms are "correct definition" in view of algebraic semantics. All syntactic notions on the language corresponds to semantical notions. For instance, in the initial algebra $\mathcal{T}X$, since TX is a functor,

$$\begin{array}{ccccc}
m & t & \in & TX_\sigma(m) \\
\big\downarrow{\scriptstyle\rho} & \big\downarrow & & \big\downarrow{\scriptstyle TX_\sigma(\rho) = \rho^\sharp} \\
n & \rho^\sharp t & \in & TX_\sigma(n)
\end{array}$$

holds. In the term language, this corresponds to the $\sharp$-Weakening rule

$$\frac{X \vdash t : \sigma : m}{X \vdash \rho^\sharp t : \sigma : n.}$$

Example 2. Consider Σ^Λ for the signature for the lambda calculus given in Example 1. Now we can give initial algebra semantics for the $\Sigma^\Lambda_{\mathsf{N}}$-algebra of lambda terms Λ by a non-syntactic binding algebra, called *clone* [10, 4].

Let C be arbitrary cartesian closed category with the currying cur and the evaluation ev, and $\mathsf{fold} : D^D \triangleleft D : \mathsf{unfold}$ a retraction in C.

A *clone*[5] $O(D) \in (\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$ is defined by

$$O(D)_{[\mathsf{N}]^k\iota}(n) = C(D^n, \underbrace{D \Rightarrow \cdots \Rightarrow}_{k \text{ times}} D)$$
$$O(D)_\sigma(\rho)(f) = f \circ \langle \pi_{\rho(1)}, \ldots, \pi_{\rho(m)} \rangle$$

where $\rho : m \to n$.

Then, $O(D)$ is a $\Sigma^{\Lambda}_{\mathsf{N}}$-binding algebra by defining $O(D)_{\mathsf{N}} = \mathsf{N}$ and the operations as follows:

$$\begin{aligned}
\mathsf{nam}_{O(D)}(n) &: n \to C(D^n, D)\\
& \quad i \mapsto \pi_i\\
\mathsf{lam}_{O(D)}(n) &: C(D^n, D^D) \to C(D^n, D)\\
& \quad f \mapsto \mathsf{fold} \circ f\\
\mathsf{app}_{O(D)}(n) &: C(D^n, D) \times C(D^n, D) \to C(D^n, D)\\
& \quad (f, t) \mapsto \mathsf{ev} \circ \langle \mathsf{unfold} \circ f, t \rangle\ .
\end{aligned}$$

The binding map

$$\mathsf{bind}_{O(D),\iota}(n) : O(D)_\iota(n+1) \to O(D)_{[\mathsf{N}]\iota}(n)\ ,$$

which is actually of the type $C(D^{n+1}, D) \to C(D^n, D^D)$, is defined by

$$\mathsf{bind}_{O(D),\iota}(n)(f) = \mathsf{cur}(f)\ . \tag{3}$$

So obviously, bind gives isomorphisms. Hence these data give a $\Sigma^{\Lambda}_{\mathsf{N}}$-binding algebra and it follows from Theorem 1 that $O(D)[\![\_]\!] : \Lambda \to O(D)$ is a unique homomorphism. This is initial algebra semantics for the lambda calculus. The equation (3) is somewhat dogmatically saying that "binding" is "currying" in the model of lambda calculus. We can easily show that this gives the traditional compositional interpretation function $\Lambda(n) \to C(D^n, D)$ ($n \in \mathbb{N}$) of lambda terms. □

4. EQUATIONAL LOGIC

Before defining rewriting system, we formulate equational logic for binding terms. It is useful to seek a right definition of rewriting on binding terms, because rewriting can be considered as equational deduction without using symmetricity. Also, it is needed to formulate the notion of pattern matching in defining one-step rewriting.

4.1 Logic

For terms $X \vdash s : \sigma : n$ and $X \vdash t : \sigma : n$, an *equation* is of the form

$$X \vdash s = t : \sigma : n.$$

Let E be an arbitrary set of equations (called *axioms*). We write

$$X \vdash s =_E t : \sigma : n$$

if it is derivable from E using the inference rules of the equational logic given in Fig. 2.

4.2 Algebraic semantics

We define the notion of (Σ, E)-binding algebras for our equational logic as algebras in the category $(\mathbf{Eq}^{\mathbb{I}})^{\mathbb{T}}$.

As a notational convention, for $=_A \in (\mathbf{Eq}^{\mathbb{I}})^{\mathbb{T}}$, we may write $=_{A_\sigma(n)}$ for $(=_A)_\sigma(n) \in \mathbf{Eq}$.

[5]It is short for "closed set of operations" [4], where closedness is by compositions. Here we use the presheaf version of it [10] extended with $\mathbb{T}$-indexing.

Inference rules
Rules for congruence

$$(\text{Ax})\frac{X \vdash s = t : \sigma : n \in E}{X \vdash s =_E t : \sigma : n}$$

$$(\text{Var})\frac{}{X, x{:}\sigma{:}m \vdash \xi x =_E \xi x : \sigma : n}\ \xi : m \to n$$

$$(\text{Name})\frac{}{X \vdash \mathsf{nam}(k) =_E \mathsf{nam}(k) : \iota : n}\ 1 \le k \le n$$

$$(\text{Fun})\frac{X \vdash t_1{:}\sigma_i{:}n \ \cdots\ X \vdash t_l{:}\sigma_l{:}n \quad X \vdash t_k =_E t'_k : \sigma_k : n}{X \vdash f(t_1, \ldots, t_k, \ldots, t_l) =_E f(t_1, \ldots, t'_k, \ldots, t_l) : \sigma : n}$$

$$(\text{Abs})\frac{X \vdash s =_E t : \sigma : n+1}{X \vdash [n+1]s =_E [n+1]t : [\mathsf{N}]\sigma : n}$$

$$(\text{Subst})\frac{Y \vdash s : \sigma : m \quad X, x : \sigma : m \vdash t =_E t' : \tau : n}{X \cup Y \vdash \{x \mapsto s\}t =_E \{x \mapsto s\}t' : \tau : n}$$

Rules for equivalence relation

$$(\text{Sym})\frac{X \vdash s =_E t : \sigma : n}{X \vdash t =_E s : \sigma : n}$$

$$(\text{Tr})\frac{X \vdash s =_E t : \sigma : n \quad X \vdash t =_E u : \sigma : n}{X \vdash s =_E u : \sigma : n}$$

Rules for variants

$$(\natural)\frac{X \vdash s =_E t : \sigma : m}{\rho X \vdash \rho^\natural s =_E \rho^\natural t : \sigma : n}\ \rho : m \to n$$

Figure 2: Equational logic

Definition 4. Let Σ_{N} be a signature and E a set of equations. A (Σ_N, E)*-binding algebra* $\mathcal{A}$ is given by a tuple

$$(A, \alpha, \{\mathsf{bind}_{A,\sigma}\}_{\sigma \in \mathsf{Ty}}, =_A)$$

such that

- $(A, \alpha : \Sigma_{\mathsf{N}} A \to A, \{\mathsf{bind}_{A,\sigma}\}_{\sigma \in \mathsf{Ty}}$ is a Σ_{N}-binding algebra in $(\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$,
- $=_A \in (\mathbf{Eq}^{\mathbb{I}})^{\mathbb{T}}$ satisfies E, i.e. for
  - every $Y \vdash s = t : \sigma : n \in E$,
  - any valuation $\theta : Y \to A$,
  - a unique $\dot{Y} + \Sigma_{\mathsf{N}}$-binding algebra homomorphism $h : \mathcal{TY} \to (A, [\theta, \alpha], \{\mathsf{bind}_{A,\sigma}\}_{\sigma \in \mathsf{Ty}})$ in $(\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$,

  $h_\sigma(n)(s) =_{A_\sigma(n)} h_\sigma(n)(t)$ holds,
- $f_{\mathcal{A}}$ preserves $=_A$ for each $f \in \Sigma_\sigma$, i.e. when $a_1 =_{A_{\sigma_1}(n)} a'_1, \ldots, a_l =_{A_{\sigma_l}(n)} a'_l$, $f_{\mathcal{A}}(n)(a_1, \ldots, a_l) =_{A_\sigma(n)} f_{\mathcal{A}}(n)(a'_1, \ldots, a'_l)$, for every $n \in \mathbb{I}$.
- $\mathsf{bind}_{A,\sigma}$ preserves $=_A$, i.e. when $a =_{A_\sigma(n)} a'$, $\mathsf{bind}_{A,\sigma}(n)(a) =_{A_{[\mathsf{N}]\sigma}(n)} \mathsf{bind}_{A,\sigma}(n)(a')$, for every $\sigma \in \mathsf{Ty}, n \in \mathbb{I}$. □

A (Σ_{N}, E)-binding algebra defined above fits into the notion of "binding algebra in $(\mathbf{Eq}^{\mathbb{I}})^{\mathbb{T}}$" in Definition 3 if we present it by the triple

$$(=_A,\ \alpha' : \Sigma_{\mathsf{N}}(=_A) \to =_A,\ \{\mathsf{bind}_{=_A,\sigma} : \delta(=_{A_\sigma}) \to (=_{A_{[\mathsf{N}]\sigma}})\}_{\sigma \in \mathsf{Ty}})$$

where the signature functor Σ_{N} is considered as a functor on $(\mathbf{Eq}^{\mathbb{I}})^{\mathbb{T}}$ by the same definition, and

$$\alpha' \triangleq \alpha \times \alpha, \quad \mathsf{bind}_{=_A,\sigma} \triangleq \mathsf{bind}_{A,\sigma} \times \mathsf{bind}_{A,\sigma}.$$

Notice that α' and $\mathsf{bind}_{=_A,\sigma}$ can actually be maps in $\mathbf{Eq}^{\mathbb{I}}$ and $(\mathbf{Eq}^{\mathbb{I}})^{\mathbb{T}}$ respectively because we required $f_{\mathcal{A}}$ and $\mathsf{bind}_{A,\sigma}$ preserved $=_A$. Hence, $(=_A, \alpha')$ defines a Σ_{N}-algebra in $(\mathbf{Eq}^{\mathbb{I}})^{\mathbb{T}}$, and $\mathsf{bind}_{=_A,\sigma}$ gives a natural isomorphism in $\mathbf{Eq}^{\mathbb{I}}$.

Let X a variable declaration set. Then, we define the functor $=_X \in (\mathbf{Eq}^{\mathbb{I}})^{\mathbb{T}}$ giving equivalence relations by

$$(=_X)_\sigma(n) = \{(\xi x, \xi x) \mid \xi x \in \dot{X}_\sigma\ (n)\}$$
$$(=_X)_\sigma(\rho) = \rho^\sharp_{\restriction \dot{X}_\sigma(m)} \times \rho^\sharp_{\restriction \dot{X}_\sigma(m)} \qquad \text{for } \rho : m \to n \in \mathbb{I},$$

and $=_{T_E X} \in (\mathbf{Eq}^{\mathbb{I}})^{\mathbb{T}}$ by

$$(=_{T_E X})_\sigma(n) = \{(s,t) \mid X \vdash s =_E t : \sigma : n\}$$
$$(=_{T_E X})_\sigma(\rho) = \rho^\sharp \times \rho^\sharp.$$

Theorem 2. *Let $\mathcal{T}X = (TX, \tau, \{\mathsf{bind}_{TX,\sigma}\}_{\sigma \in \mathbb{T}y})$ be the term algebra. For a set E of equations,*

$$\mathcal{T}_E X \;=\; (TX, \tau, \{\mathsf{bind}_{TX,\sigma}\}_{\sigma \in \mathbb{T}y}, =_{T_E X})$$

is an initial $(=_X + \Sigma_N, E)$-binding algebra.

Proof. By the definition of inference rules of equational logic, it is clear that $f_{\mathcal{T}X}$ and $\mathsf{bind}_{\mathcal{T}X}$ preserve $=_{T_E X}$. So, $\mathcal{T}_E X$ is a $(=_X + \Sigma_{\mathsf{N}}, E)$-binding algebra. The initiality is similar to Theorem 1. □

Corollary 3. *The equational logic is sound and complete for the algebraic semantics, i.e.*

$$X \vdash s =_E t : \sigma : n$$

is derivable if and only if

$$\mathcal{M}[\![s]\!]_\sigma(n) =_{M_\sigma(n)} \mathcal{M}[\![t]\!]_\sigma(n)$$

holds for all $(=_X + \Sigma_N, E)$-binding algebra $\mathcal{M} = (M, \alpha, =_M)$.

Proof. ($\Leftarrow$): Assume $X \vdash s =_E t$. Then

$$s \quad (=_{T_E X})_\sigma(n) \quad t.$$

Since there exists a unique homomorphism $h : \mathcal{T}_E X \to \mathcal{M}$, apply it to the above.

($\Rightarrow$): Assume $\mathcal{M}[\![s]\!]_\sigma(n) =_{M_\sigma(n)} \mathcal{M}[\![t]\!]_\sigma(n)$ for any $(=_X + \Sigma, E)$-algebra $\mathcal{M}$. Take $\mathcal{M} = \mathcal{T}_E X$:

$$\mathcal{T}X[\![s]\!]_\sigma(n) \quad (=_{T_E X})_\sigma(n) \quad \mathcal{T}X[\![t]\!]_\sigma(n).$$

So $X \vdash s =_E t : \sigma : n.$ □

Example 3. We again consider the signature $\Sigma_{\mathsf{N}}^{\Lambda}$ for the lambda calculus extended with the function symbol for substitution

$$\mathsf{sub} : [\mathsf{N}]\iota, \iota \to \iota.$$

Now we can write the β-axiom $(\lambda a.x)y = x[a{:=}y]$ in our syntax as follows:

$$x{:}\iota{:}1,\ y{:}\iota{:}0 \vdash \mathsf{app}(\mathsf{lam}([1]x), y) = \mathsf{sub}([1]x, y) : \iota : 0, \qquad (4)$$

where sub is used for encoding meta-level substitution (here, the bound variable a corresponds to the name 1). To completely axiomatise the β-axiom, we also need the axioms for sub as follows:

$$y{:}\iota{:}0 \vdash \mathsf{sub}([1]\mathsf{nam}(1), y) = y : \iota : 0$$
$$y{:}\iota{:}0 \vdash \mathsf{sub}([2]\mathsf{nam}(1), y) = \mathsf{nam}(1) : \iota : 1$$
$$e_1{:}\iota{:}1,\ e_2{:}\iota{:}1,\ y{:}\iota{:}0 \vdash \mathsf{sub}([1]\mathsf{app}(e_1, e_2), y)$$
$$= \mathsf{app}(\mathsf{sub}([1]e_1, y), \mathsf{sub}([1]e_2, y)) : \iota : 0$$
$$x{:}\iota{:}2,\ y{:}\iota{:}0 \vdash \mathsf{sub}([1]\mathsf{lam}([2]x), y)$$
$$= \mathsf{lam}([1](\mathsf{sub}([2][1{:=}2, 2{:=}1]x, y))) : \iota : 0. \qquad (5)$$

This is just a usual recursive definition of substitution of lambda terms written in our equational logic[6]. If we naively write the right-hand side of the equation (5), it might be $\mathsf{lam}([2](\mathsf{sub}([1]x, y)))$. But the method of de Bruijn levels requires that the binders must appear as firstly [1] and next [2]. So this was written using the swapping renamer for x. This also shows that *renamer is a really needed* syntactic construct if we want to express this kind of equation (i.e. the same variable x appears both sides of equation which depends on bound names) using de Bruijn levels.

We write Λ for the set of axioms for β and sub above. One may think that the β-axiom (4) is only stated for $y{:}\iota{:}0$, i.e. y cannot contain any names, it is so restrictive. But it is not true, we can apply arbitrary weakening renaming to the axiom. For instance, applying $\rho : 0 \to 1$ by ($\natural$)-rule, we obtain a *variant* of the β-axiom

$$x{:}\iota{:}1,\ y{:}\iota{:}1 \vdash \mathsf{app}(\mathsf{lam}([2][1{:=}2]x), y) =_\Lambda \mathsf{sub}([2][1{:=}2]x, y){:}\iota{:}1 \qquad (6)$$

which means that y can contain the name 1. Interestingly, this automatically gives a *capture avoiding substitution*, because y can only contain the name 1, and the binder is now shifted to [2] by weakening, so capturing of names by the binder cannot happen. This is not an ad hoc hack, this is an immediate consequence of our definition of equational logic motivated by the presheaf semantics.

Moreover, we can soundly interpret these axioms in the $\Sigma_{\mathsf{N}}^{\Lambda} \cup \{\mathsf{sub}\}$-binding algebra $\mathcal{O}(\mathcal{D})$ with the operation

$$\mathsf{sub}_{\mathcal{O}(\mathcal{D})}(n) : C(D^n, D^D) \times C(D^n, D) \to C(D^n, D)$$
$$(f, t) \mapsto \mathsf{ev} \circ \langle f, t \rangle\ .$$

It is easy to check that this validates the axioms for sub. Then, with the valuation

$$x : \iota : 1 \mapsto f \in C(D, D) = \mathcal{O}(\mathcal{D})_\iota(1)$$
$$y : \iota : 0 \mapsto t \in C(1, D) = \mathcal{O}(\mathcal{D})_\iota(0),$$

the β-axiom can be validated:

$$\mathcal{O}(\mathcal{D})[\![\mathsf{app}(\mathsf{lam}([1]x), y)]\!]_\iota(0) = \mathsf{ev} \circ \langle \mathsf{unfold} \circ \mathsf{fold} \circ \mathsf{cur}(f),\ t\rangle$$
$$= \mathsf{ev} \circ \langle \mathsf{cur}(f),\ t\rangle$$
$$= \mathcal{O}(\mathcal{D})[\![\mathsf{sub}([1]x, y)]\!]_\iota(0).$$

Importantly, all the variants of β-axiom like (6) can also be validated because of soundness of the equational logic[7]. Hence, the $(\Sigma_{\mathsf{N}}^{\Lambda} \cup \{\mathsf{sub}\}, \Lambda)$-binding algebra $(\mathcal{O}(\mathcal{D}), =)$ is a well-known *λ-algebra* [1]. This shows again the appropriateness of our definition of equational logic, now the classical notion of λ-algebra which also semantically expresses the notion of binding is actually an instance of binding algebra.

5. BINDING TERM REWRITING SYSTEMS

In this section, we define term rewriting systems which treat binding terms.

5.1 Rewrite logic and rewrite relation

Definition 5. (**BTRS**) For terms $X \vdash l : \sigma : n$ and $X \vdash r : \sigma : n$, a *rewrite rule* is of the form

$$X \vdash l \to r : \sigma : n$$

[6] This also defines a *substitution algebra* [10]. And the same substitution operation is defined in [10] in a more categorical language.

[7] This is a completely different point from the first-order representation of variable binding using (variable,term)-pairs. We can say that the inference of equational logic correctly preserves the notion of λ-binding in this sense.

such that all variables in r occur in l, and l is not a variable. A *binding term rewriting system* (BTRS) is a set of rewrite rules.

Let $\mathcal{R}$ be a BTRS. We will define a rewrite relation $\to_{\mathcal{R}}$ on terms in two styles: one is to deduce a valid one-step rewriting using inference rules like equational logic, the other is ordinary operational way [6] by selecting a redex position and doing matching for a given term etc.

The former style is called rewrite logic given in Fig. 3, and the latter style of definition is given in Definition 6.

Rules for rewrite relation

$$\text{(Rule)}\frac{X \vdash l \to r : \sigma : n \in \mathcal{R}}{X \vdash l \to_{\mathcal{R}} r : \sigma : n}$$

$$\text{(Fun)}\frac{X \vdash t_1{:}\sigma_i{:}n \ \cdots \ X \vdash t_l{:}\sigma_l{:}n \quad X \vdash t_k \to_{\mathcal{R}} t'_k : \sigma_k : n \quad f \in \Sigma_\sigma \quad l > 0}{X \vdash f(t_1, \ldots, t_k, \ldots, t_l) \to_{\mathcal{R}} f(t_1, \ldots, t'_k, \ldots, t_l) : \sigma : n}$$

$$\text{(Abs)}\frac{X \vdash s \to_{\mathcal{R}} t : \sigma : n+1}{X \vdash ([n+1]s) \to_{\mathcal{R}} ([n+1]t) : [\mathrm{N}]\sigma : n}$$

$$\text{(Subst)}\frac{Y \vdash s : \sigma : m \quad X, x : \sigma : m \vdash t \to_{\mathcal{R}} t' : \sigma : n}{X \cup Y \vdash \{x \mapsto s\}t \to_{\mathcal{R}} \{x \mapsto s\}t' : \sigma : n}$$

Rules for variants

$$(\natural)\frac{X \vdash s \to_{\mathcal{R}} t : \sigma : m}{\rho X \vdash \rho^\natural s \to_{\mathcal{R}} \rho^\natural t : \sigma : n} \ \rho : m \to n$$

Figure 3: Rewrite logic for one-step rewriting

Notation 1. For a term t, the notation $t_{|p}$ denotes the subterm of t at a position p, and $t[s]_p$ the term that is obtained from t by replacing the subterm at the position p by the term s.

Definition 6. The rewrite rules of a BTRS $\mathcal{R}$ define a *rewrite relation* $\to'_{\mathcal{R}}$ on terms as follows: for $Y \vdash s : \sigma : n$,

$$s \to'_{\mathcal{R}} s[\theta\rho^\natural r]_p$$

if there exist

- a rule $X \vdash l \to r : \sigma : m \in \mathcal{R}$
- a position p in s,
- a renaming $\rho : m \to n$,
- a substitution $\theta : \rho X \to T(Y)$

such that

$$Y \vdash (\theta\rho^\natural l) =_\varnothing s_{|p} : \sigma : n.$$

Note: this uses the equality in the equational logic with empty axiom.

5.2 Pattern matching

In the definitoin of $\to'_{\mathcal{R}}$, the substitution θ and the renaming ρ are matchers of two terms $X \vdash l : \sigma : m$ and $Y \vdash s_{|p} : \sigma : n$. The definition just requires the existence of them. By using the unification algorithm BUA [13, 14] for terms with name binding, we can acturally compute them.

Matching can be considered as a special case of unification, where the answer substitution is only applied to one side. A *matching problem* is a pair $(X \vdash l : \tau : m,\ Y \vdash t : \tau : n)$ of terms where X only contains the free variables of l. A *matcher* to the matching problem is a pair (θ, ρ) satisfying

$$Y \vdash (\theta\rho^\natural l) =_\varnothing t : \tau : n$$

where

- $\theta : \rho X \to T(Y)$ is a substitution,
- $\rho : m \to n$ is a renaming.

Matchers for a matching problem can also be computed by BUA without the elimination rule of variable in the right-hand side, because now we need a substitution only applied to the left-hand side. Anotherway to compute matchers by BUA is to just syntactically replace variables in the right-hand side of a matching problem with constants, which gives the same effect by BUA without the elimination rule of variable in the right-hand side.

Originally, BUA was developped for the term language with all renamings (i.e. not restricted to injective ones) [13, 14]. By just restricting renamers in the language to injective, and rejecting non-injective computed unifers, we can use BUA as an unification algorithm for the languge with injective renamings (i.e. the setting of this paper). Matching by this modified BUA is also sound, complete and decidable, which is inherited by the properties of BUA for unification problems [13].

5.3 Relating rewrite relation and rewrite logic

Lemma 4. *(**Subject Reduction**) If $X \vdash s : \sigma : n$ and $s \to'_{\mathcal{R}} t$, then $X \vdash t : \sigma : n$.*

Lemma 5. *For arbitrary renaming $\rho : m \to n$ and substitution $\theta : X \to T(Y)$, there exists a substitution $\overline{\theta} : \rho X \to T(Y)$ such that $\rho^\natural \circ \theta = \overline{\theta} \circ \rho^\natural_{\restriction X} : X \to T(Y)$.*

Proof. Define $\overline{\theta}(x) = \rho^\natural\,\theta(x)$ for each $x : \sigma : \rho(m) \in \rho X$. □

Theorem 6. *Let $Y \vdash s : \sigma : n$ and $Y \vdash t : \sigma : n$. $Y \vdash s \to_{\mathcal{R}} t : \sigma : n$ if and only if $s \to'_{\mathcal{R}} t$.*

Proof. ($\Leftarrow$): Suppose $s \to'_{\mathcal{R}} t \equiv s[\theta\rho^\natural r]_p$ by $X' \vdash l \to r : \sigma : m \in \mathcal{R}$, a position p, a renaming $\rho : m \to n$, and a substitution θ. We can construct a rewrite logic proof by applying firstly (Rule), and next ($\natural$) by $\rho^\natural$, then (Subst) by θ, and congruence rules to make a context. This becomes a proof of $Y \vdash s \to_{\mathcal{R}} t : \sigma : n$.
($\Rightarrow$): The essential difference between two styles of the definition of rewrite relation is that in the matching style, an instance of the left-hand side of a rule is of the form $\theta\rho^\natural l$. However, in the rewrite logic, a rule can be instantiated by mixed combinations of θ's by (Subst) and ρ's by ($\natural$). But these can be re-ordered by Lemma 5 and eventually becomes of the form $\theta\rho^\natural l$, which fits the definition in the matching style. □

Hence, $\to_{\mathcal{R}}$ coincides with $\to'_{\mathcal{R}}$ as the rewrite relation on terms.

We also define *many-step rewriting* $\to^*_{\mathcal{R}}$ by reflexive and transitive closure of $\to_{\mathcal{R}}$. As for rewrite logic, it is defined by just the inference system of the equational logic without (Sym).

5.4 Algebraic semantics

Algebraic semantics of BTRSs is similar to the one for equation logic in Section 4. Now we consider algebras in $(\mathbf{Pre}^{\mathbb{I}})^{\mathbb{T}}$ instead of $(\mathbf{Eq}^{\mathbb{I}})^{\mathbb{T}}$. For $\leq_A \in (\mathbf{Pre}^{\mathbb{I}})^{\mathbb{T}}$, we may write $\leq_{A_\sigma(n)}$ for $(\leq_A)_\sigma(n) \in \mathbf{Pre}$.

Definition 7. Let Σ_{N} be a signature and $\mathcal{R}$ a BTRS. A $(\Sigma_N, \mathcal{R})$-*binding algebra* $\mathcal{A}$ is given by a tuple

$$(A, \alpha, \{\mathsf{bind}_{A,\sigma}\}_{\sigma \in \mathsf{Ty}}, \leq_A)$$

such that

- $(A, \alpha : \Sigma_{\mathsf{N}} A \to A, \{\mathsf{bind}_{A,\sigma}\}_{\sigma\in\mathsf{Ty}})$ is a Σ_{N}-binding algebra in $(\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$,
- $\leq_A \in (\mathbf{Pre}^{\mathbb{I}})^{\mathbb{T}}$ respects $\mathcal{R}$, i.e. for
  - every $Y \vdash s \to t : \sigma : n \in \mathcal{R}$,
  - any valuation $\theta : Y \to A$,
  - a unique $\dot{Y} + \Sigma_{\mathsf{N}}$-binding algebra homomorphism $h : \mathcal{TY} \to (A, [\dot{\theta}, \alpha], \{\mathsf{bind}_{A,\sigma}\}_{\sigma\in\mathsf{Ty}})$ in $(\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$,

  $h_\sigma(n)(s) \leq_{A_\sigma(n)} h_\sigma(n)(t)$ holds,
- $f_{\mathcal{A}}$ preserves $\leq_A$ for each $f \in \Sigma$, and
- $\mathsf{bind}_{A,\sigma}$ preserves $\leq_A$ for each $\sigma \in \mathsf{Ty}$. □

A $(\Sigma_{\mathsf{N}}, \mathcal{R})$-binding algebra defined above fits into the notion of "binding algebra in $(\mathbf{Pre}^{\mathbb{I}})^{\mathbb{T}}$" in Definition 3 if we present it by the triple

$$(\leq_A,\ \alpha' : \Sigma_{\mathsf{N}}(\leq_A) \to \leq_A,\ \{\mathsf{bind}_{\leq_A,\sigma} : \delta(\leq_{A_\sigma}) \to \leq_{A_{[\mathsf{N}]\sigma}}\}_{\sigma\in\mathsf{Ty}})$$

where $\Sigma_{\mathsf{N}} : (\mathbf{Pre}^{\mathbb{I}})^{\mathbb{T}} \to (\mathbf{Pre}^{\mathbb{I}})^{\mathbb{T}}$, $\alpha' \triangleq \alpha\times\alpha$, $\mathsf{bind}_{\leq_A,\sigma} \triangleq \mathsf{bind}_{A,\sigma}\times\mathsf{bind}_{A,\sigma}$ as in the case of equational logic.

Let X a variable declaration set. We define $\leq_X \in (\mathbf{Pre}^{\mathbb{I}})^{\mathbb{T}}$ by $\leq_X \triangleq =_X$ (because $(\mathbf{Eq}^{\mathbb{I}})^{\mathbb{T}}$ is a subcategory of $(\mathbf{Pre}^{\mathbb{I}})^{\mathbb{T}}$), and $\leq_{T_{\mathcal{R}}X} \in (\mathbf{Pre}^{\mathbb{I}})^{\mathbb{T}}$ by

$$(\leq_{T_{\mathcal{R}}X})_\sigma(n) = \{(s,t) \mid X \vdash s \to^*_{\mathcal{R}} t : \sigma : n\}$$
$$(\leq_{T_{\mathcal{R}}X})_\sigma(\rho) = \rho^\sharp\times\rho^\sharp.$$

Then, we can establish initiality and sound and completeness of rewrite logic, which are proved by the same way as the case of equational logic.

Theorem 7. *Let $\mathcal{T}X = (TX, \tau, \{\mathsf{bind}_{TX,\sigma}\}_{\sigma\in\mathsf{Ty}})$ be the term algebra. Then, for a BTRS $\mathcal{R}$,*

$$T_{\mathcal{R}}X \ = \ (TX, \tau, \{\mathsf{bind}_{TX,\sigma}\}_{\sigma\in\mathsf{Ty}}, \leq_{T_{\mathcal{R}}X})$$

is an initial $(\leq_X + \Sigma_{\mathsf{N}}, \mathcal{R})$-binding algebra. □

Corollary 8. *The rewrite logic is sound and complete for the algebraic semantics, i.e.*

$$X \vdash s \to^*_{\mathcal{R}} t : \sigma : n$$

is derivable in the rewrite logic extended with reflexivity and transitivity rules if and only if

$$\mathcal{M}[\![s]\!]_\sigma(n) \leq_M \mathcal{M}[\![t]\!]_\sigma(n)$$

holds for all $(\leq_X + \Sigma_{\mathsf{N}}, \mathcal{R})$-binding algebra $\mathcal{M} = (M, \alpha, \leq_M)$. □

An immediate example is a BTRS for the lambda calculus. This can be given by just regarding the equational axioms for the lambda calculus presented in Example 3 as rewrite rules of a BTRS. Using matching style definition, we can compute β-normal forms (if exist) of given terms by this BTRS.

6. DISCUSSION AND RELATED WORK

In this section, we discus our design choice of syntax and semantics from a semantic perspective. We also mention related and future work.

6.1 The category $(\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$

In several work on language with binding, the presheaf category $\mathbf{Set}^{\mathbb{F}}$ is often chosen for semantics [10, 13, 8] (and partly [11]); the category $\mathbb{F}$ has all renamings rather than only injective ones in case of $\mathbb{I}$. We have been working with $(\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$ and its variants in this paper. Our choice fits into Pitts' slogan on language with names (also observed by Hofmann [17]) "Name-swapping (is an injective renaming) has much more convenient logical properties than renaming" [28]. For instance, if we state an axiom such as "$\mathsf{ineq}(a,b) = \mathsf{true}$" for inequality, and apply a non-injective renaming $\{a,b\} \to \{a\}$ to the axiom, we get an incorrect theorem "$\mathsf{ineq}(a,a) = \mathsf{true}$". So, at least for equational logic and rewriting, we must choose $\mathbf{Eq}^{\mathbb{I}}$ and $\mathbf{Pre}^{\mathbb{I}}$ for semantics. Moreover, even for term language, non-injective renaming is not suitable. Because, for example, consider the term [1][2]nam(2). If we admit a contractive renaming such as $2 \to 1$, we obtain by bind maps

$$\begin{tikzcd} {[1][2]\mathsf{nam}(2) : [\mathsf{N}][\mathsf{N}]\iota : 0} \arrow[r, mapsto, "\mathsf{bind}^{-1}\circ\mathsf{bind}^{-1}"] & {\mathsf{nam}(2) : \iota : 2} \arrow[d, mapsto] \\ {[1]\mathsf{nam}(1) : [\mathsf{N}]\iota : 0} & {\mathsf{nam}(1) : \iota : 1} \arrow[l, mapsto, "\mathsf{bind}"'] \end{tikzcd}$$

but we do not want to identify different binders with the single one by renaming operation. So we chose $(\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$ and assumed the naturality only in stages, and we defined that $\mathbb{T}$ was just a discrete category of types.

It is interesting to consider the language that has many-sorted names rather than the single sorted ones developed in this paper. This is needed to model typed systems like the simply typed lambda calculus. Fiore [8] uses algebras in the category $(\mathbf{Set}^{\mathbb{F}\downarrow\mathbb{T}})^{\mathbb{T}}$ where the category $\mathbb{T}$ consists of all simple types for this purpose. So as an extension of the language developed here, we can expect to obtain a language (and equational logic, BTRSs) with many-sorted names using the category $(\mathbf{Set}^{\mathbb{I}\downarrow\mathbb{T}})^{\mathbb{T}}$ with suitable $\mathbb{T}$ along the same line of this paper.

6.2 Type $[\mathsf{N}]\sigma$, fresh name application and β_0

We have not included β-like axiom in our equational logic. Since an intuitive meaning of the type $[\mathsf{N}]\sigma$ is a type of abstraction by a name, one may think that this kind of thing can be done by introducing a function type $\mathsf{N}\Rightarrow\sigma$. This idea has been taken by the work on λProlog and L_λ [26, 25], higher-order abstract syntax (HOAS) [7] and the work using $\mathbf{Set}^{\mathbb{F}}$.

But we have been here working in $\mathbf{Set}^{\mathbb{I}}$. This category has been used to model systems having the notion of names: Pitts and Stark's ν-calculus [33], π-calculus [9, 34], CCS [11], and Hofmann's semantics analysis of HOAS [17]. We discuss the type $[\mathsf{N}]\sigma$, related application and β-axiom in our system by following Stark's analysis on $\mathbf{Set}^{\mathbb{I}}$ [34]. The category $\mathbf{Set}^{\mathbb{I}}$ has a cartesian closed structure given by

$$(A\times B)(n) = A(n)\times B(n)$$
$$(A\Rightarrow B)(n) = \mathbf{Set}^{\mathbb{I}}(A(n+\_), B(n+\_))$$

and also has a symmetric monoidal closed structure: the exponential is given by

$$(A \multimap B)(n) = \mathbf{Set}^{\mathbb{I}}(A, B(n+\_)),$$

and the monoidal product can be described by

$$\begin{aligned}(A\otimes B)(n) = \{\, & (a,b) \in (A\times B)(n) \mid \\ & \exists\rho : i \to n, \phi : j \to n, a' \in A(i), b' \in B(j)\,. \\ & a \in A(\rho)(a'), b \in B(\phi)(b'),\ \mathsf{Im}\rho \cap \mathsf{Im}\phi = \emptyset\}\end{aligned} \qquad (7)$$

when A and B are pullback preserving functors (in particular, TX is pullback preserving). Thus, for the object of names N, we have two versions of exponential expressed by

$$(\mathsf{N} \Rightarrow A)(n) \cong A(n)^n \times A(n+1),$$
$$(\mathsf{N} \multimap A)(n) \cong A(n+1) = \delta A(n).$$

We have required $M_{[\mathsf{N}]\sigma} \cong \delta M_\sigma$ for a binding algebra. This means the interpretation of name abstraction type is

$$[\![[\mathsf{N}]\sigma]\!] \cong \delta[\![\sigma]\!] \cong \mathsf{N} \multimap [\![\sigma]\!],$$

and the related evaluation map at a stage n is of the form

$$\mathsf{app}_\otimes(n) : ((\mathsf{N} \multimap A) \otimes \mathsf{N})(n) \to A(n).$$

Take $A = TX$ to consider syntax. An important point is that the parameters of $\mathsf{app}_\otimes$ are given by the monoidal product $\otimes$, so by (7), the set n of names is divided into two disjoint sets and distributed to $(\mathsf{N} \multimap TX)$ and N. Namely, in $\mathsf{app}_\otimes$, an abstraction and a name to apply *cannot share the same names*, which means only *fresh names* are applicable. Roughly, $\mathsf{app}_\otimes$ can be defined in the equational logic as like

$$x : \sigma : \{a_i\}_{1,\dots,n} \vdash$$
$$\mathsf{app}_\otimes([a]x, b) = [a{:=}b, a_1{:=}b_1, \dots, a_n{:=}b_n]x \quad : \sigma : \{b, b_i\}_{1,\dots,m}$$

where $\{a_i\}_{1,\dots,n} \subseteq \{b_i\}_{1,\dots,m}, n \le m$, and the condition $b \notin \{a_i\}$ must be satisfied for such freshness requirement. However, we have no way to express such requirement for $\mathsf{app}_\otimes$ in the object-level (i.e. in the equational logic).

But interestingly, the right-hand side can be written as a variable with renamer, and it automatically satisfies the freshness $b \notin \{a_i\}$ because the renamer must be *injective* by definition. This is one of the reason we do not include an application and β-like axioms, and instead, use a variable with renamer construct for (a normal form of) an application by a name. Another reason is to make the definition of BTRSs simple, which is particularly important to analyse rewriting properties. Without application, how binder can be eliminated in practical situation is, as in $\mathsf{app}_\otimes$-equation (also see (8)), an abstraction term can be decomposed by using pattern matching.

Pitts and Gabbay's FreshML [29] is a functional programming language based on their semantics of binding – FM-set theory [12], whose underlying categorical structure is also (a subcategory of) $\mathbf{Set}^{\mathbb{I}}$ [12]. FreshML's application operator (@ [29], `concrete` [32]) exactly corresponds to $\mathsf{app}_\otimes$, where freshness is ensured by the type system. Also, the same situation arised in consideration of Miller's logic programming language L_λ. Miller stated the β_0-axiom for decidable higher-order unification. The β_0-axiom is $(\lambda x.M)y = M[x := y]$, provided a variable y is not free in $(\lambda x.M)$ [25]. Now, we can semantically understand that this axiom is not just a restriction of β-axiom, its application can be regarded as $\mathsf{app}_\otimes$.

In summary, the type $[\mathsf{N}]\sigma$ and $\mathsf{app}_\otimes$ give terms of abstraction and application by fresh names, and related β can be expressed as renamers in our language, which can be considered as normal forms of β_0-redexes.

On the other hand, the evaluation map for the cartesian closed structure

$$\mathsf{app}_\times(n) : ((\mathsf{N} \Rightarrow A) \times \mathsf{N})(n) \to A(n)$$

can contain the same names in the set n for the abstraction and the name to apply. Since we do not have the type $\mathsf{N} \Rightarrow A$ in the language, we have no syntactic construct corresponding to $\mathsf{app}_\times$. This results that terms containing a contractive (= non-injective) renamer cannot be written. It might be interesting to extend the language including the type $\mathsf{N} \Rightarrow A$ to overcome this restriction. However, the lack of type $\mathsf{N} \Rightarrow A$ is not actually a problem due to the expressive power of our equational logic, which we will discuss in Section 6.3.

As another direction, introducing the type constructor corresponding to the monoidal product $\otimes$ into the language might be interesting. The cartesian product has implicitly appeared in the type of function symbols, namely, a function symbol $f : \sigma_1, \dots, \sigma_n \to \sigma$ is interpreted as $[\![f]\!] : [\![\sigma_1]\!] \times \dots \times [\![\sigma_n]\!] \to [\![\sigma]\!]$. Instead, if we interpret the source as $\otimes$-products, we obtain *affine binding terms* with respect to name appearance, as we saw in $\mathsf{app}_\otimes$. This direction is taken by Tanaka [35], where the subcategory $\mathbb{P}$ of $\mathbb{I}$, sets and permutations, is used, and hence linear binding terms are modeled by this version of binding algebras in $\mathbf{Set}^{\mathbb{P}}$.

6.3 Non-injective renaming

In some application, a contractive renaming is essentially needed. One of the major examples appears in formalisation of the π-calculus, namely in COM-rule

$$\frac{P \xrightarrow{a(b)} P' \quad Q \xrightarrow{\bar{a}c} Q'}{P \mid Q \xrightarrow{\tau} P'\{c/b\} \mid Q'}$$

We try to write this by a rewrite rule of a BTRS. It may be written as something like

$$P' {:} \iota {:} \{a, b, c\}, Q' {:} \iota {:} \{a, c\} \vdash$$
$$\mathsf{in}(\mathsf{nam}(a), [b]P') \mid \mathsf{out}(\mathsf{nam}(a), \mathsf{nam}(c), Q') \;\to\; \tau([b{:=}c]P' \mid Q') \quad : \iota : \{a, c\} \qquad (8)$$

using a suitable signature (here, we use nameful syntax). We especially notice that the function symbol for the input prefix has the type

$$\mathsf{in} : \iota, [\mathsf{N}]\iota \to \iota$$

to indicate the name b in P' at the second argument [9, 34]. Processes can be written as variables containing some names, and the substitution $\{c/b\}$ to P' is written as a variable with renamer in the rewrite rule. This is very smoothly written, so this shows the expressiveness power of BTRSs (also usefulness of (meta)variables).

However, the renamer $[b{:=}c]$ actually must be written as $[a{:=}a, b{:=}c, c{:=}c]$, which is a contractive map from 3 to 2. Thus, in [9, 34], the type of in is taken as

$$\mathsf{in} : \iota \times (\mathsf{N} \Rightarrow \iota) \to \iota.$$

Although we do not have such a function type, this can be easily overcome. Instead of using the renamer, we can explicitly define a non-injective renaming function on the particular data type (in this case, the processes) by rewrite rules. The reason why this is possible is simple: for example, using the function symbol $\mathsf{rename} : \iota \to \iota$ and the rewrite rules such as

$$\begin{cases} \vdash \mathsf{rename}(\mathsf{nam}(a)) & \to \mathsf{nam}(a) \\ \vdash \mathsf{rename}(\mathsf{nam}(b)) & \to \mathsf{nam}(a)\,, \end{cases}$$

we can realise a non-injective renaming. Because, to understand semantically, the meaning of it is a function

$$\mathsf{rename}_{\mathcal{M}}(n) : M_\iota(n) \to M_\iota(n).$$

Namely, "$\to$" indicated in the arity of function symbol means an arrow of $(\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$, whose components themselves are just functions, not having the injectivity condition. Since the same name set n can be used for the domain and codomain of the operation, non-injective renamings are actually definable. This way is also practically used in the example programs of FreshML [32] and FreshML also only has the type for name abstraction corresponding to $[\mathsf{N}]\sigma$.

FreshML has many similarity to BTRSs, so it is hopeful to give a *rewriting semantics* of FreshML by BTRSs. This will be useful to show rewriting properties of FreshML programs such as termination and modularity. A main difference between two formalisms is the freshness assumption of names in FreshML, which is indirectly expressible in BTRSs. How radical use of freshness can be represented by BTRS's formalism is not explored yet, it is future work.

6.4 The initial algebra property of BTRSs for termination

Although it is usually not emphasised, the notion of initial algebra semantics has already appeared in the theory of TRSs, i.e. the fundamental theorem of termination of TRSs:

THEOREM 9. *(**[18, 37]**) A TRS $\mathcal{R}$ is terminating if and only if it is compatible with a well-founded monotone Σ-algebra.*

A well-founded monotone Σ-algebra $\mathcal{A}$ is a (ordinary) Σ-algebra with a strict partial order $>_{\mathcal{A}}$ on the carrier such that the order is well-founded and every operation is monotone, and a TRS $\mathcal{R}$ is compatible with a Σ-algebra $\mathcal{A}$ means that for every $l \to r \in \mathcal{R}$, $[\![l]\!] >_{\mathcal{A}} [\![r]\!]$ for a unique homomorphism $[\![-]\!]$ from the term algebra to $\mathcal{A}$. This is nothing but an initial algebra semantics in the case of Σ-algebra in **Pre**, where a particular Σ-algebra whose preorder is a well-founded strict partial order is used. This result is obviously extended to BTRSs to characterise termination of BTRSs due to our algebraic framework. This fact is also hopeful to reduce the termination problem of BTRSs to that of TRSs.

6.5 Application to modularity of BTRSs

It is well-known that the first-order term algebra construction $T_\Sigma : X \mapsto T_\Sigma(X)$ gives a monad on **Set**. Likewise, we can show that the construction of a many-step rewrite relation of a BTRS $(\leq_X) \mapsto (\to^*_{\mathcal{R}})$ induces a monad on $(\mathbf{Pre}^{\mathbb{I}})^{\mathbb{T}}$. This observation in the case of TRSs (namely, inducing a monad on **Pre**) was a key idea of Lüth and Ghani's monadic semantics of TRSs to show *modularity* of confluence and termination of TRSs [23, 24]. It is rather promising to extend their method to BTRSs because our categorical algebraic framework is a natural extension of the first-order case as we saw, and also the base categories **Set** and $(\mathbf{Set}^{\mathbb{I}})^{\mathbb{T}}$ have many similar properties. This is in contrast to the case of higher-order rewrite systems, where modularity result has not explored yet, even a good tool to analyse structure of higher-order terms and rewriting structure has not been known.

The availability of the monadic method to prove modularity can be considered as an advantage of developing BTRSs using the categorical algebraic framework. So, we plan to work the modularity of confluence and termination of BTRSs as a next step of our initial algebra approach to term rewriting systems with variable binding.

Acknowledgements. I am grateful to Neil Ghani for his suggestion of working with a preorder-valued functor category. This work is supported in part a grant from the Okawa Foundation for Information and Telecommunications.

7. REFERENCES

[1] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North Holland, 1984.

[2] F. Blanqui, J.-P. Jouannaud, and M. Okada. The calculus of algebraic constructions. In *Rewriting Techniques and Applications (RTA 1999)*, LNCS 1631, pages 301–316. Springer, 1999.

[3] F. Blanqui, J.-P. Jouannaud, and M. Okada. Inductive data type systems. *Theoretical Computer Science*, 272:41–68, 2002.

[4] P. Cohn. *Universal Algebra*. Harper & Row, 1965.

[5] N. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagationes Mathematicae*, 34:381–391, 1972.

[6] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. The MIT Press/Elsevier, 1990.

[7] J. Despeyroux, A. Felty, and A. Hirschowitz. Higher-order abstract syntax in Coq. In *Typed Lambda Calculi and Applications, LNCS 902*, pages 124–138, 1995.

[8] M. Fiore. Semantic analysis of normalisation by evaluation for typed lambda calculus. In *4th International Conference on Principles and Practice of Declarative Programming (PPDP 2002)*, pages 26–37. ACM Press, 2002.

[9] M. Fiore, E. Moggi, and D. Sangiorgi. A fully-abstract model for the pi-calculus. In *11th Logic in Computer Science Conf. (LICS'96)*, pages 43–54. IEEE, 1996. Information and Computation 179, 76-117 (2002).

[10] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Proc. 14th Annual Symposium on Logic in Computer Science*, pages 193–202, 1999.

[11] M. Fiore and D. Turi. Semantics of name and value passing. In *Proc. 16th LICS Conf.*, pages 93–104. IEEE, Computer Society Press, 2001.

[12] M. Gabbay and A. Pitts. A new approach to abstract syntax involving binders. In *14th Annual Symposium on Logic in Computer Science*, pages 214–224, 1999.

[13] M. Hamana. A logic programming language based on binding algebras. In *4th International Symposium on Theoretical Aspects of Computer Software (TACS 2001)*, LNCS 2215, pages 243–262, 2001.

[14] M. Hamana. Simple β_0-unification for terms with context holes. In *16th International Workshop on Unification (UNIF 2002)*, pages 9–13, 2002. Technical report 02-5, Department of Computer Science, The University of Iowa.

[15] M. Hamana. Nameful, nameless, and de Bruijn level binding in binding algebras. draft, 2003.

[16] M. Hashimoto and A. Ohori. A typed context calculus. *Theoretical Computer Science*, 266:249–271, 2001.

[17] M. Hofmann. Semantical analysis of higher-order abstract syntax. In *14th Annual Symposium on Logic in Computer Science*, pages 204–213, 1999.

[18] G. Huet and D. S. Lankford. On the uniform halting problem for term rewriting systems. Technical Report Rapport Laboria 283, INRIA, 1978.

[19] J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *Proc. LICS'99*, pages 402–411. IEEE, 1999.

[20] G. Kelly and A. Power. Adjunctions whose counits are coequalizers and presentations of enriched monads. *Journal of Pure and Applied Algebra*, 1993.

[21] J. Klop. *Combinatory Reduction Systems*. PhD thesis, CWI, Amsterdam, 1980. volume 127 of Mathematical Centre Tracts.

[22] P. Lescanne and J. Rouyer-Degli. Explicit substitutions with de Bruijn's levels. In *Rewriting Techniques and Applications,*

6th International Conference (RTA-95), LNCS 914. Springer, 1995.

[23] C. Lüth. Compositional term rewriting: An algebraic proof of toyama's theorem. In *Rewriting Techniques and Applications (RTA-96)*, LNCS 1103, pages 261– 275, 1996.

[24] C. Lüth and N. Ghani. Monads and modular term rewriting. In *Category Theory and Computer Science*, LNCS 1290, pages 69– 86, 1997.

[25] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[26] G. Nadathur and D. Miller. An overview of λProlog. In *Fifth International Logic Programming Conference*, pages 810–827. MIT Press, 1988.

[27] T. Nipkow. Higher-order critical pairs. In *Proc. 6th IEEE Symp. Logic in Computer Science*, pages 342–349, 1991.

[28] A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*. To appear. (A preliminary version appeared in TACS 2001, LNCS 2215, 2001, pp 219–242.).

[29] A. M. Pitts and M. J. Gabbay. A metalanguage for programming with bound names modulo renaming. In *Mathematics of Program Construction, MPC2000*, LNCS 1837, pages 230–255, 2000.

[30] G. Plotkin. Another meta-language for programming with bound names modulo renaming. Lecture slides, *Winter Workshop in Logics, Types and Rewriting*, Heriot-Watt University, February 2000.

[31] E. Robinson. Variations on algebra: monadicity and generalisations of equational theories. *Formal Aspects of Computing*, 13(3-5):308–326, 2002.

[32] M. R. Shinwell and A. M. Pitts. *FreshML User Manual*. Cambridge University Computer Laboratory, Nov. 2002. Available at ⟨`http://www.freshml.org/docs/`⟩.

[33] I. Stark. Categorical models for local names. *Lisp and Symbolic Computation*, 9(1):77–107, Feb. 1996.

[34] I. Stark. A fully abstract domain model for the π-calculus. In *Proc. of the Eleventh Annual IEEE Symposium on Logic in Computer Science*, pages 36–42, 1996.

[35] M. Tanaka. Abstract syntax and variable binding for linear binders. In *MFCS 2000*, LNCS 1893, pages 670–679, 2000.

[36] D. Turi. *Functorial Operational Semantics and its Denotational Dual*. PhD thesis, Free University, Amsterdam, June 1996.

[37] H. Zantema. Termination of term rewriting: interpretation and type elimination. *Journal of Symbolic Computation*, 17:23–50, 1994.