Higher-Order Semantic Labelling for Inductive Datatype Systems

Makoto Hamana *
Gunma University / The University of Tokyo
hamana@ipl.t.u-tokyo.ac.jp

Abstract

We give a novel transformation for proving termination of higher-order rewrite systems in the format of Inductive Data Type Systems (IDTSs) by Blanqui, Jouannaud and Okada. The transformation called higher-order semantic labelling attaches algebraic semantics of the arguments to each function symbol. We systematically define the labelling and show that labelled systems give termination models in the framework of Fiore, Plotkin and Turi's binding algebras. As applications, we give simple proofs of termination of the explicit substitution system λx and currying transformation via higher-order semantic labelling. Moreover, we prove a new result of modularity of termination of IDTSs by introducing the notion of solid IDTSs. We prove that termination is preserved under the disjoint union of an IDTS and a higher-order program scheme.

Categories and Subject Descriptors F.4.2 [*Grammars and Other Rewriting Systems*]; 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, Operational Semantics

General Terms Theory, Languages

Keywords Termination, modularity, initial algebra semantics, higher-order abstract syntax, higher-order rewriting, categorical semantics

* Currently visiting IPL, The University of Tokyo as a Domestic Researcher from Gunma University, Japan. This work is supported in part by the JSPS Grant-in-Aid for Scientific Research (90334135).

PPDP'07 July 14–16, 2007, Wroclaw, Poland.

1. Introduction

In this paper we develop a theory of *higher-order semantic labelling*, which is a transformation for proving termination (meaning strong normalisation) of Blanqui, Jouannaud and Okada's Inductive Data Type Systems [3, 1].

We illustrate how this transformation method is applicable to several seemingly different problems: we prove termination of

- the explicit substitution system of the λx-calculus (Sect. 4), and
- the currying transformation (Sect. 5.1) that is an extension of Kennaway, Klop, Sleep and de Vries' result.

We also prove modularity of termination for the disjoint union of an Inductive Data Type System (IDTS) and a higher-order recursive program scheme (HO-RPS) (Sect. 5.2) via higher-order semantic labelling and introduction of the "solid" property of IDTSs. To the author's best knowledge, these are new results on modularity of termination of higher-order rewrite rules.

Higher-order extensions of term rewriting systems [34] are known as several formats: major representatives are Combinatory Reduction Systems (CRSs) [21, 22], Higher-order Rewrite Systems [28], and Inductive Data Type Systems [3]. There exist several termination criteria: higher-order recursive path order (HORPO) [16, 18], the General Schema [3, 1], hereditary monotone functional interpretation [31], binding algebra interpretation [14]. Recently improvements of HORPO/General Schema are actively investigated [6, 32, 17, 2].

This paper is based on the algebraic semantics of CRS by the author [14] by binding algebras and Σ-monoids. The notion of binding algebras and Σ-monoids was introduced by Fiore, Plotkin and Turi [11] to give algebraic semantics of higher-order abstract syntax. Typed versions of binding algebras for typed higher-order abstract syntax are extensively investigated [10, 26, 33, 19]. A higher-order syntax for free Σ-monoids was developed by the author [13].

Organisation. This paper is organised as follows. We first review the definition of Inductive Data Type Systems and its

semantics in Sect. 2. We give higher-order semantic labelling of IDTSs in Sect. 3. In Sect. 4, we consider an application of higher-order semantic labelling to the termination problem of the λX-calculus. In Sect. 5, we investigate semantic labelling with term model and its application to currying transformation and modularity of termination of IDTSs.

Convention. We use the vector notation $\vec{A}$ for a sequence $A_1, \cdots, A_l$, and $|\vec{A}|$ for its length. We use the notation $s[x := t]$ for the usual capture-avoiding substitution that replaces all x in s with t. We may call this meta-level substitutions to distinguish them from explicit substitutions. $\mathsf{FV}(t)$ denotes the set of all free variables in t. We often omit superscripts or subscripts (for types, typing contexts) of a mathematical object if they are clear from contexts.

2. Preliminaries

2.1 Inductive Data Type Systems

The Inductive Data Type Systems (IDTS) are introduced by Blanqui, Jounnaud and Okada [3]. This is a framework of rewrite rules on terms built on inductively defined datatypes and simple types. The IDTS is extended further by Blanqui [1] by admitting higher-order pattern matching when applying rewrite rules. We use this version of IDTS [1] with the following restrictions.

- We use simple types up to the second-order.
- IDTS's binder $(-.-)$ always appears as an argument of a function symbol. For example, $lam(x^b.x)$ for a function symbol $lam : (b \to b) \to b$.
- We do not assume a global set of variables. Instead, we use local typing contexts.

Since the fragment we use is just a subclass of IDTSs in [1], the termination criterion of IDTSs, called *the General Schema* [1, 3], is also applicable to this fragment. Notice that although this fragment has simple types up to second-order, it can encode simple types of *arbitrary order* by the feature of inductive types (see the case of simply-typed λX-calculus in Sect. 4.1). Hence this fragment has enough expressibility to represent higher-order rewrite rules involving all simple types and inductive types.

For a set $\mathcal{B}$ of *base types*, the set $I(\mathcal{B})$ of *first-order types*, and the set $T(\mathcal{B})$ of *types* are given as follows:

$$\begin{aligned} \mathcal{B} &\ni b \\ I(\mathcal{B}) &\ni \alpha ::= b \mid b_1, \ldots, b_n \to b \\ T(\mathcal{B}) &\ni \tau ::= b \mid \alpha_1, \ldots, \alpha_n \to b \end{aligned}$$

An IDTS-alphabet $\mathcal{A}$ is a triple $(\mathcal{B}, \Sigma, Z)$ (or, often just denoted by $(\mathcal{B}, \Sigma)$) where

- $\mathcal{B}$ is a set of base types (denoted by b, a),
- Σ, called a *signature*, is a family $(\Sigma_\tau)_{\tau \in T(\mathcal{B})}$ of sets of function symbols (denoted by $f : (\overrightarrow{a_1} \to b_1), \ldots, (\overrightarrow{a_n} \to b_n) \to b$)
- Z is a family $(Z_\alpha)_{\alpha \in I(\mathcal{B})}$ of sets of *metavariables* (written as small-caps letters Z or, capital letters M, N)

such that all the sets are pairwise disjoint. The raw syntax is given as follows.

- *Terms* have the form

$$t ::= x \mid x.t \mid f(t_1, \ldots, t_n).$$

  These forms are respectively called *variables*, *abstractions*, and *function terms*.

- *Meta-terms* extend terms to

$$t ::= x \mid x.t \mid f(t_1, \ldots, t_n) \mid \mathrm{Z}(t_1, \ldots, t_n).$$

  The last form is called a *meta-application*.

A context Γ is a sequence of variable:type-pairs. A meta-term t is well-typed if $\Gamma \vdash t : \tau$ is derived from the following rules for some context Γ.

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \qquad \frac{\mathrm{Z} \in Z_{b_1,\ldots,b_n \to b} \quad \Gamma \vdash t_1 : b_1 \ \cdots \ \Gamma \vdash t_n : b_n}{\Gamma \vdash \mathrm{Z}(t_1, \ldots, t_n) : b}$$

$$\frac{\Gamma, \overrightarrow{i_1 : a_1} \vdash t_1 : b_1 \ \cdots \ \Gamma, \overrightarrow{i_n : a_n} \vdash t_n : b_n}{\Gamma \vdash f(\ \overrightarrow{i_1^{a_1}}.t_1, \ \ldots, \ \overrightarrow{i_n^{a_n}}.t_n\ ) : b}$$

where $f : (\overrightarrow{a_1} \to b_1), \ldots, (\overrightarrow{a_n} \to b_n) \to b \in \Sigma$.

Notation 2.1 Since the binders are clear from the type of function symbol, we often abbreviate $f(\ \overrightarrow{i_1^{a_1}}.t_1, \ \ldots, \ \overrightarrow{i_n^{a_n}}.t_n\ )$ as just $f(t_1, \ldots, t_l)$.

A *rewrite rule* $l \to r$ consists of two meta-terms l and r with the following additional restrictions: (i) $\vdash l : \tau$ and $\vdash r : \tau$ for some type τ (hence l and r are closed (w.r.t. variables) meta-terms), (ii) l must be a function term where all meta-applications have the form $\mathrm{Z}(x_1, \ldots, x_n)$ (called a higher-order pattern) with distinct variables x_i, (iii) r can only contain metavariables occurring in the left-hand side. We may use the notation $Z | \Gamma \vdash s \to t : \tau$ for a rule or a rewrite step if metavariables and variables in s and t are included in Z and Γ respectively. We may also simply write $Z \vdash s \to t : \tau$ or $\Gamma \vdash s \to t : \tau$ if another part is not important.

A *valuation* θ is a mapping that assigns to a metavariable $\mathrm{Z} : \vec{b} \to \tau$ a term t

$$\theta : \mathrm{Z} \longmapsto t$$

where $x_1 : b_1, \cdots, x_n : b_n \vdash t : \tau$. Any valuation θ is extended to a function θ^* on meta-terms:

$$\begin{aligned} \theta^*(x) &= x \\ \theta^*(x.t) &= x.\theta^*(t) \\ \theta^*(f(t_1, \ldots, t_n)) &= f(\theta^*(t_1), \ldots, \theta^*(t_n)) \\ \theta^*(\mathrm{Z}(t_1, \ldots, t_n)) &= \theta(\mathrm{Z})\,(\theta^*(t_1), \ldots, \theta^*(t_n)) \end{aligned}$$

The right-hand side of the last equation means replacing free variables $x_1, \ldots, x_n$ in $\theta(\mathrm{Z})$ with $\theta^*(t_1), \ldots, \theta^*(t_n)$. We may also write $t\theta$ for $\theta^* t$.

A set of rewrite rules under a alphabet $(\mathcal{B}, \Sigma, Z)$ is called an *IDTS* and denoted by $(\mathcal{B}, \Sigma, \mathcal{R})$, or simply $(\Sigma, \mathcal{R})$ or $\mathcal{R}$. The *rewrite relation* $\rightarrow_{(\Sigma,\mathcal{R})}$ or simply $\rightarrow_{\mathcal{R}}$ is generated by context and valuation closure of a given IDTS $(\Sigma, \mathcal{R})$.

By the notation $s \rhd t$, we mean that a well-typed meta-term t is a (strict) sub-meta-term of a well-typed meta-term s. Also, we regard $\rhd$ as a relation. We use $\trianglerighteq = \rhd \cup \mathrm{id}$. As usual, turning over a symbol, we mean its inverse (e.g. $\rhd^{-1} = \lhd$), and $(-)^*$, $(-)^+$ for the reflexive and transitive closure, and the transitive closure, respectively.

2.2 Binding algebras – semantic structure of syntax with variable binding

In a previous work, algebraic semantics of CRSs is given by binding algebras [14]. Since IDTS can be seen as a typed version of CRS, algebraic semantics of IDTSs can be given by typed binding algebras [10, 26, 33, 19]. We review the notion of typed binding algebras in this subsection.

A key idea of binding algebras is to model variables in abstract syntax as natural numbers. To match this semantic treatment and syntactic presentation tightly, in this paper we assume the method of de Bruijn levels (N.B. not indices) [8, 24] for a formal treatment of named variables modulo α-equivalence in IDTSs. Under the method of de Bruijn levels, variables are natural numbers.

A variable $i \in \mathbb{N}$ of type $\tau \in \mathcal{B}$ is denoted by $i : \tau$ or i^τ. In this setting, a typing context Γ is a finite subset $\{1 : \tau_1, \cdots, l : \tau_l\}$ of typed variables.

Now, we model these variables and contexts in a categorical setting. Let $\mathbb{F}$ be the category which has finite cardinals $n = \{1, \ldots, n\}$ (n is possibly 0) as objects, and all functions between them as arrows $m \to n$. For a set $\mathcal{B}$ of all base types, the slice category $\mathbb{F}\downarrow\mathcal{B}$ is seen as the category of contexts and their renamings, i.e. objects $\Gamma : n \to \mathcal{B}$ are contexts, and arrows $f : \Gamma \to \Gamma'$ are functions $f : n \to n'$ such that $\Gamma = \Gamma' \circ f$ for $\Gamma : n \to \mathcal{B}$, $\Gamma' : n' \to \mathcal{B}$. Binary coproducts $\Gamma + \Gamma'$ are given by copairs $[\Gamma, \Gamma'] : n + n' \to \mathcal{B}$. By abuse of notation, for a type τ, we denote just by τ a context $\tau : 1 \to \mathcal{B},\ 1 \mapsto \tau$. Thus we have a coproduct $\Gamma + \tau$.

We use the functor category $(\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{B}})^{\mathcal{B}}$ to define binding algebras. An object of a functor category may be called a *presheaf*. Ordinary universal algebra is a structure on sets. Binding algebra is a structure on presheaves. The reason we use presheaves is to uniformly deal with (context,type)-indexed sets of syntactic entities. A typical example is well-typed terms indexed by contexts and types, i.e. for IDTSs, the set of all meta-terms. For instance, for an IDTS-alphabet $(\mathcal{B}, \Sigma, Z)$, we can define the indexed set of all meta-terms by

$$M_\Sigma Z_\tau(\Gamma) = \{t \mid \Gamma \vdash t : \tau\}.$$

With a suitable arrow part, this $M_\Sigma Z$ forms a presheaf, i.e., an object of $(\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{B}})^{\mathcal{B}}$.

We define the functor $\delta_\tau : \mathbf{Set}^{\mathbb{F}\downarrow\mathcal{B}} \to \mathbf{Set}^{\mathbb{F}\downarrow\mathcal{B}}$ for context extension by a variable of type τ, which is used to model a bound variable, by $(\delta_\tau A)(\Gamma) = A(\Gamma + \tau)$ for $A \in \mathbf{Set}^{\mathbb{F}\downarrow\mathcal{B}}$. We write $\delta_{\vec{\tau}}$ for the composition $\delta_{\tau_1} \circ \cdots \circ \delta_{\tau_l}$. To a signature Σ, we associate the *signature functor* $\Sigma : (\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{B}})^{\mathcal{B}} \to (\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{B}})^{\mathcal{B}}$ given by

$$(\Sigma A)_b = \coprod_{f:(\vec{a_1}\to b_1),\cdots,(\vec{a_l}\to b_l)\to b \in \Sigma} \prod_{1\le i\le l} \delta_{\vec{a_i}} A_{b_i}.$$

A point of this modeling is that an arrow type, say $a \to b$, is interpreted as a presheaf $\delta_a A_b$. This is the crucial difference of binding algebras from polynomial functor-algebras on sets for modeling first-order abstract syntax or inductive types.

A Σ-*(binding) algebra* is a pair of a presheaf (for a carrier) and a natural transformation (for operations) $(A, \alpha : \Sigma A \to A)$ where the *algebra structure* α is a copair $[f^{\mathcal{A}}]_{f\in\Sigma}$ of all operations on A corresponding to function symbols. A *homomorphism* of Σ-algebras is a map $\phi : (A, \alpha) \to (B, \beta)$ such that $\phi \circ \alpha = \beta \circ \Sigma\phi$.

A set of variables of a certain type taken from a context is modeled by the presheaf $\mathrm{V} \in (\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{B}})^{\mathcal{B}}$ defined by Yoneda embedding $\mathrm{V}_\tau = \mathbb{F}\downarrow\mathcal{B}(\tau, -)$, hence $\mathrm{V}_\tau(\Gamma) \cong \{x \mid x : \tau \in \Gamma\}$.

To model substitution, we use the notion of monoid in a monoidal category. The structure of Σ-monoids is for this purpose where a substitution is modeled by a multiplication of a monoid. The definition is given below.

Remark 2.2 In this work, Σ-monoid structure is used in the proofs in Sect.3, but in the applications of semantic labelling from Sect.4, Σ-monoids are not used. So the readers who are interested in applications can skip the following definitions.

The category $(\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{B}})^{\mathcal{B}}$ forms a monoidal category with the unit V and the "substitution" monoidal product $\bullet$ given by

$$(A \bullet B)_\tau(\Delta) = (\coprod_{\Gamma\in\mathbb{F}\downarrow\mathcal{B}} A_\tau(\Gamma) \times \prod_{i\in m} B_{\tau_i}(\Delta))/\sim$$

where $\Gamma = \{i : \tau_1, \ldots, m : \tau_m\}$ and $\sim$ is the equivalence relation generated by

$$(t; u_{\rho 1}, \ldots, u_{\rho m}) \sim (A(\rho)(t); u_1, \ldots, u_l)$$

for an arrow $\rho : \Gamma \to \Gamma'$ of $\mathbb{F}\downarrow\mathcal{B}$.

A Σ-*monoid* (A, α, ν, μ) consists of a *monoid object* [25] $M = (M, \eta : \mathrm{V} \to M, \mu : M \bullet M \to M)$ in the monoidal category $((\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{B}})^{\mathcal{B}}, \bullet, \mathrm{V})$ with a Σ-binding algebra $\alpha : \Sigma M \to M$ that satisfies $\mu \circ (\alpha \bullet \mathrm{id}_M) = \alpha \circ \Sigma\mu \circ st$, where the strength st is the one defined in [11]. A morphism of Σ-monoids $(M, \alpha) \longrightarrow (M', \alpha')$ is a morphism in $(\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{B}})^{\mathcal{B}}$ which is both Σ-algebra homomorphism and monoid morphism.

By [11, 26, 33] and as a straightforward variation of [13], we have

Theorem 2.3 $M_\Sigma Z$ forms a free Σ-monoid. Moreover, when $Z = 0$, the presheaf $M_\Sigma 0 = T_\Sigma \mathrm{V}$ of all terms forms an initial $\mathrm{V} + \Sigma$-algebra and an initial Σ-monoid.

A morphism $\phi : Z \longrightarrow A$ of $(\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{B}})^{\mathcal{B}}$ is called an *assignment* when its target has a Σ-monoid structure (A, α, η, μ). By freeness of $M_\Sigma Z$, any assignment $\phi : Z \longrightarrow A$ is extended to a Σ-monoid morphism $\phi^* : M_\Sigma Z \longrightarrow A$:

$$\begin{aligned} M_\Sigma Z_b(\Gamma) &\longrightarrow A_b(\Gamma) \\ x &\longmapsto \eta(\Gamma)(x) \qquad (x \in \Gamma) \\ f(\Theta_1.t_1, \ldots, \Theta_n.t_n) &\longmapsto f_A(\Gamma)(\phi^*(\Gamma{+}\Theta_1)(t_1), \ldots) \\ \mathsf{z}(t_1, \ldots, t_n) &\longmapsto \mu(\Gamma)(\,\phi(\Gamma)(\mathsf{z}); \phi^*(\Gamma)(t_1), \ldots). \end{aligned}$$

A valuation θ defined in Sect. 2.1 can be seen as an assignment $\theta : Z \longrightarrow T_\Sigma \mathrm{V}$ and in this case, its Σ-monoid morphism extension θ^* coincides with the extension of valuation described in Sect. 2.1 (cf. [14]).

2.3 Algebraic semantics of rewrite rules on higher-order terms

For a presheaf A, we write $\geq_A$ for a family of partial orders $(\geq_{A_\tau(\Gamma)})$ where $\geq_{A_\tau(\Gamma)}$ is a partial order on a set $A_\tau(\Gamma)$ for each type τ and context Γ. Let $(A_1, \geq_{A_1}), \ldots, (A_l, \geq_{A_l}), (B, \geq_B)$ be presheaves equipped with partial orders. A map $f : A_1 \times \cdots \times A_l \longrightarrow B$ in $\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{B}}$ is *weakly monotone* if for all Γ, all $a_1, b_1 \in A_1(\Gamma), \ldots, a_l, b_l \in A_l(\Gamma)$ with $a_k \geq_{A_k(\Gamma)} b_k$ for some k and $a_j = b_j$ for all $j \neq k$, then $f(\Gamma)(a_1, \ldots, a_l) \geq_{B(\Gamma)} f(\Gamma)(b_1, \ldots, b_l)$. A *weakly monotone* $\mathrm{V}{+}\Sigma$*-algebra* $(A, \geq_A)$ is a $\mathrm{V}{+}\Sigma$-algebra equipped with partial orders such that every operation is weakly monotone.

For a $\mathrm{V} + \Sigma$-algebra A, a *term-generated assignment* $\phi : Z \longrightarrow A$ is a morphism of $(\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{B}})^{\mathcal{B}}$ that can be expressed as a composite

$$\phi = ! \circ \theta$$

where $!$ is the unique homomorphism from the initial $\mathrm{V}{+}\Sigma$-algebra $T_\Sigma \mathrm{V}$ to A, and θ is a valuation $Z \longrightarrow T_\Sigma \mathrm{V}$.

A weakly monotone $\mathrm{V} + \Sigma$-algebra $(A, \geq_A)$ *satisfies* a rewrite rule $Z|\Gamma \vdash l \to r : \tau$ of an IDTS if

$$!\theta^*_\tau(\Gamma)(l) \;\geq_{A_\tau(\Gamma)}\; !\theta^*_\tau(\Gamma)(r)$$

for all term-generated assignments $\phi = ! \circ \theta : Z \longrightarrow A$. Why the notion of term-generated assignments is needed is that the IDTS's rewrite relation is generated on terms (not on meta-terms). A *quasi-model* A for $(\Sigma, \mathcal{R})$ is a weakly monotone $\mathrm{V}{+}\Sigma$-algebra A that satisfies all rules in $\mathcal{R}$.

3. Higher-Order Semantic Labelling

We now proceed to defining semantic labelling for IDTSs. This extends semantic labelling for first-order term rewriting systems (TRSs) by Zantema [35] by employing the initial algebra semantics of typed higher-order abstract syntax using typed binding algebras [10, 26]. The main feature of semantic labelling is that termination of the original rewrite system is preserved by transforming it to a semantically labelled system. Often termination of the labelled systems can be shown by applying the General Schema criterion even when the original system does not follow it since the labelled system has much more information in function symbols.

Henceforth, we assume an IDTS-alphabet $(\mathcal{B}, \Sigma, Z)$ and a weakly monotone $\mathrm{V}{+}\Sigma$-algebra M. We introduce labelling of functions symbols: choose for every $f \in \Sigma$ a corresponding non-empty well-founded poset $(S_f, \geq_S)$ of labels, called a *semantic label set*. The signature $\overline{\Sigma}$ for labelled function symbols is defined by

$$\overline{\Sigma}_\tau = \{f_p \mid f \in \Sigma_\tau,\ p \in S_f\}.$$

A function symbol is labelled if S_f contains more than one element. For unlabelled f, the set S_f containing only one element can be left implicit; in that case we write f instead of f_p.

Choose for $f : (\vec{a}_1 \to b_1), \ldots, (\vec{a}_l \to b_l) \to b \in \Sigma$, a *semantic label map* that is a weakly monotone morphism of $\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{B}}$ defined by

$$\langle\!\langle - \rangle\!\rangle^f : \delta_{\vec{a}_1} M_{b_1} \times \cdots \times \delta_{\vec{a}_l} M_{b_l} \longrightarrow (K_{S_f})_b$$

where $K_{S_f} \in (\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{B}})^{\mathcal{B}}$ is the constant presheaf defined by $(K_{S_f})_b(\Gamma) = S_f$. Then, as in the case of ordinary signature, we define $M_{\overline{\Sigma}} Z$ by the presheaf of all meta-terms generated by the labelled signature $\overline{\Sigma}$. Labelling of function symbols is given by the following map.

Definition 3.1 (**Labelling map**) Let $\phi : Z \longrightarrow M$ be a term-generated assignment such that $\phi = ! \circ \theta$. The *labelling map* $\phi^{\mathsf{L}} : M_\Sigma Z \longrightarrow M_{\overline{\Sigma}} Z$ is a map of $(\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{B}})^{\mathcal{B}}$ defined by

$$\begin{aligned} &\phi^{\mathsf{L}}_{\tau,\Gamma} : M_\Sigma Z_{\tau,\Gamma} \longrightarrow M_{\overline{\Sigma}} Z_{\tau,\Gamma} \\ &\phi^{\mathsf{L}}_{\tau,\Gamma}(x) = x \\ &\phi^{\mathsf{L}}_{\tau,\Gamma}(\mathsf{z}(\vec{t})) = \mathsf{z}(\phi^{\mathsf{L}}_{\vec{\sigma},\Gamma}\vec{t}) \\ &\phi^{\mathsf{L}}_{\tau,\Gamma}(f(\Theta_1.t_1, \ldots, \Theta_l.t_l)) \\ &= f_{\langle\!\langle !\theta^*(t_1), \ldots, !\theta^*(t_l) \rangle\!\rangle^f_{\tau,\Gamma}}(\phi^{\mathsf{L}}_{\tau_1,\Gamma+\Theta_1} t_1, \ldots, \phi^{\mathsf{L}}_{\tau_l,\Gamma+\Theta_l} t_l) \end{aligned}$$

It is obvious that the presheaf $M_{\overline{\Sigma}} Z$ of $\overline{\Sigma}$-meta-terms forms a free $\overline{\Sigma}$-monoid. Moreover, $M_{\overline{\Sigma}} Z$ forms a non-free Σ-monoid. This has the Σ-monoid structure

$$(M_{\overline{\Sigma}} Z, [f^\phi]_{f\in\Sigma}, \nu^\phi, \beta^\phi)$$

described below for each term-generated assignment $\phi : Z \longrightarrow M$. For clarity, here we only give the $\mathrm{V}{+}\Sigma$-algebra structure part. The monoid structure is given in Appendix A. Let $||-||$ be the function that erases all labels in a labeled meta-term.

Unit. $\nu^\phi : \mathrm{V} \to M_{\overline{\Sigma}} Z$ is defined by $x \mapsto x$.

$$\begin{aligned}
\mathsf{ap}_{\sigma,\tau}(\lambda_{\sigma,\tau}(x.M(x)), N) &\to M(x)\langle x := N\rangle \\
\mathsf{ap}(M(x), N(x))\langle x := K\rangle &\to \mathsf{ap}(M(x)\langle x := K\rangle, N(x)\langle x := K\rangle) \\
\lambda(y.M(x,y))\langle x := K\rangle &\to \lambda(x.M(y,x)\langle y := K\rangle) \\
\mathsf{v}(x)\langle x := K\rangle &\to K \\
M\langle x := K\rangle &\to M
\end{aligned}$$

Figure 1. The IDTS $\mathcal{R}_{\lambda \mathrm{X}}$ of the λX-calculus

Operations. For $f : (\overrightarrow{a_1} \to b_1), \ldots, (\overrightarrow{a_n} \to b_n) \to b \in \Sigma$, the corresponding operation

$$f^\phi : \delta_{\vec{a}_1} M_{\overline{\Sigma}} Z_{b_1} \times \cdots \times \delta_{\vec{a}_l} M_{\overline{\Sigma}} Z_{b_l} \longrightarrow M_{\overline{\Sigma}} Z_b$$

is defined by

$$f^\phi(\Gamma)(s_1, \ldots, s_l) = f_{\langle\langle !\theta^* || s_1 || \ldots, !\theta^* || s_l || \rangle\rangle^f_{\tau,\Gamma}}(s_1, \ldots, s_l).$$

Theorem 3.2 For each term-generated assignment $\phi : Z \longrightarrow M$, $(M_{\overline{\Sigma}} Z, [f^\phi]_{f\in\Sigma}, \nu^\phi, \beta^\phi)$ is a Σ-monoid.

Corollary 3.3 For each term-generated assignment $\phi : Z \longrightarrow M$, the labelling map $\phi^{\mathsf{L}} : M_\Sigma Z \longrightarrow M_{\overline{\Sigma}} Z$ is the unique Σ-monoid morphism from the free Σ-monoid to $(M_{\overline{\Sigma}} Z, [f^\phi]_{f\in\Sigma}, \nu^\phi, \beta^\phi)$ that extends $\eta_Z : Z \longrightarrow M_{\overline{\Sigma}} Z$, $\mathrm{z} \longmapsto \mathrm{z}$.

Proof It is clear that $(\eta_Z)^* = \phi^{\mathsf{L}}$ by just comparing the definitions of ϕ^{L} and the Σ-monoid extension $(-)^*$. Hence ϕ^{L} gives a Σ-monoid morphism. □

Lemma 3.4 Let $\phi : 0 \longrightarrow M$ and $\theta : Z \longrightarrow M_\Sigma 0$ be assignments. Then

(i) $\phi^{\mathsf{L}} \circ \beta = \beta^\phi \circ (\phi^{\mathsf{L}} \bullet \phi^{\mathsf{L}})$.
(ii) $\phi^{\mathsf{L}} \circ \theta^* = (\phi^{\mathsf{L}}\theta)^* \circ (\phi^*\theta)^{\mathsf{L}}$.

Proof (i) Since ϕ^{L} is a monoid morphism, the multiplication β is preserved.
(ii) By induction on meta-terms in $M_\Sigma Z$. The cases x and $f(\vec{s}) \in M_\Sigma Z_\tau(\Gamma)$ are straightforward. For the case $\mathrm{z}(\vec{t}) \in M_\Sigma Z_\tau(\Gamma)$, we have the following.

$$\begin{aligned}
\mathrm{lhs} &= \phi^{\mathsf{L}}\theta^*(\mathrm{z}(\vec{t})) \\
&= \phi^{\mathsf{L}}\beta(\theta\mathrm{z}; \theta^*\vec{t}) = \beta^\phi(\phi^{\mathsf{L}}\theta\mathrm{z}; \phi^{\mathsf{L}}\theta^*\vec{t}) \qquad \text{(by Lemma 3.4)} \\
\mathrm{rhs} &= (\phi^{\mathsf{L}}\theta)^*(\phi^*\theta)^{\mathsf{L}}\mathrm{z}(\vec{t}) \\
&= (\phi^{\mathsf{L}}\theta)^*\mathrm{z}((\phi^*\theta)^{\mathsf{L}}\vec{t}) \\
&= \beta^\phi(\phi^{\mathsf{L}}\theta\mathrm{z}; (\phi^{\mathsf{L}}\theta)^*(\phi^*\theta)^{\mathsf{L}}\vec{t}) \\
&= \beta^\phi(\phi^{\mathsf{L}}\theta\mathrm{z}; \phi^{\mathsf{L}}\theta^*\vec{t}) = \mathrm{lhs} \qquad \text{(by I.H.)}
\end{aligned}$$

□

Given an IDTS $(\Sigma, \mathcal{R})$ and a V+Σ-algebra M, we define the labelled rules with respect to M by

$$\overline{\mathcal{R}} = \{\phi^{\mathsf{L}}_{\tau,0} l \to \phi^{\mathsf{L}}_{\tau,0} r \mid Z \vdash l \to r : \tau \in \mathcal{R}, \text{ term-generated assignment } \phi : Z \longrightarrow M\}.$$

Thus $\overline{\mathcal{R}}$ is a set of rewrite rules consisting of labelled meta-terms. So, $(\overline{\Sigma}, \overline{\mathcal{R}})$ forms an IDTS that gives rewriting on $\overline{\Sigma}$-terms.

Notation 3.5 A meta-term of the form

$$1.\ldots n.\mathrm{z}(1, \ldots, n)$$

in an IDTS is often abbreviated as just z for a metavariable $\mathrm{z} \in Z_{b_1,\ldots,b_n \to b}$.

Moreover, we define the IDTS Decr (called "decreasing rules") over $\overline{\Sigma}$ to consist of the rules

$$f_p(\mathrm{z}_1, \ldots, \mathrm{z}_l) \;\to\; f_q(\mathrm{z}_1, \ldots, \mathrm{z}_l)$$

for all $f : (\vec{\tau}_1 \to \sigma_1), \ldots, (\vec{\tau}_l \to \sigma_l) \to \tau \in \Sigma$ and all $p >_S q \in S_f$, where $>_S$ denotes the strict part of $\geq_S$.

Using Lemma 3.4, we have the following key proposition to establish semantic labelling.

Proposition 3.6 Let M be a quasi-model for an IDTS $\mathcal{R}$. If we have $\Gamma \vdash s \to_{\mathcal{R}} t : \tau$ then

$$\Gamma \vdash \phi^{\mathsf{L}}_{\tau,\Gamma}(s) \;\to^*_{\mathsf{Decr}} \cdot \to_{\overline{\mathcal{R}}}\; \phi^{\mathsf{L}}_{\tau,\Gamma}(t)$$

holds for the assignment $\phi : 0 \longrightarrow M$.

Proof By induction on proof trees of $\to_{\mathcal{R}}$.

(i) Case $\Gamma \vdash \theta^*_{\tau,\Gamma} l \to_{\mathcal{R}} \theta^*_{\tau,\Gamma} r : \tau$.
This is derived from $Z|\Gamma \vdash l \to r \in \mathcal{R}$ where $\theta : Z \longrightarrow M_\Sigma 0$. Let $\phi : 0 \longrightarrow M$ be the assignment. Now we have a labeled rule

$$(\phi^*\theta)^{\mathsf{L}}_{\tau,\Gamma} l \to (\phi^*\theta)^{\mathsf{L}}_{\tau,\Gamma} r \in \overline{\mathcal{R}}.$$

By Lemma 3.4 and closedness of $\overline{\mathcal{R}}$-rewrite by the valuation $\phi^{\mathsf{L}}\theta : Z \longrightarrow M_{\overline{\Sigma}} 0$, we have

$$\begin{aligned}
\phi^{\mathsf{L}}_{\tau,\Gamma}(\theta^*_{\tau,\Gamma} l) &= (\phi^{\mathsf{L}}\theta)^*_{\tau,\Gamma}(\phi^*\theta)^{\mathsf{L}}_{\tau,\Gamma} l \\
&\to_{\overline{\mathcal{R}}} (\phi^{\mathsf{L}}\theta)^*_{\tau,\Gamma}(\phi^*\theta)^{\mathsf{L}}_{\tau,\Gamma} r \\
&= \phi^{\mathsf{L}}_{\tau,\Gamma}(\theta^*_{\tau,\Gamma} r)
\end{aligned}$$

(ii) Case $\Gamma \vdash f(\ldots,s,\ldots) \to_{\mathcal{R}} f(\ldots,t,\ldots):\tau$.
This is derived from $\Gamma,\Theta \vdash s \to_{\mathcal{R}} t:\sigma$. Since $(M,\geq_M)$ is a quasi-model, we have $!\theta^*_{\sigma,\Gamma+\Theta}s \geq_{M_\sigma(\Gamma+\Theta)} !\theta^*_{\sigma,\Gamma+\Theta}t$ for an term-generated assignment $\phi = !\theta$ into M. By induction hypothesis, we have $\phi^{\mathsf{L}}_{\sigma,\Gamma+\Theta}s \to^*_{\mathsf{Decr}} \cdot \to_{\overline{\mathcal{R}}} \phi^{\mathsf{L}}_{\sigma,\Gamma+\Theta}t$. Notice also that $\langle\!\langle - \rangle\!\rangle$ is weakly monotone. So,

$$\begin{aligned}
&\phi^{\mathsf{L}}_{\tau,\Gamma}(f(\ldots,s,\ldots))\\
&= f_{\langle\!\langle \ldots,!\theta^*_{n+i}s,\ldots\rangle\!\rangle_{\tau,\Gamma}}(\ldots,\phi^{\mathsf{L}}_{\sigma,\Gamma+\Theta}s,\ldots)\\
&\to^*_{\mathsf{Decr}} f_{\langle\!\langle \ldots,!\theta^*_{\sigma,\Gamma+\Theta}t,\ldots\rangle\!\rangle_{\tau,\Gamma}}(\ldots,\phi^{\mathsf{L}}_{\sigma,\Gamma+\Theta}s,\ldots)\\
&\to^*_{\mathsf{Decr}} \cdot \to_{\overline{\mathcal{R}}} f_{\langle\!\langle \ldots,!\theta^*_{\sigma,\Gamma+\Theta}t,\ldots\rangle\!\rangle_{\tau,\Gamma}}(\ldots,\phi^{\mathsf{L}}_{\sigma,\Gamma+\Theta}t,\ldots)\\
&= \phi^{\mathsf{L}}_{\tau,\Gamma}(f(\ldots,\phi^{\mathsf{L}}_{\sigma,\Gamma+\Theta}t,\ldots))
\end{aligned}$$

□

Theorem 3.7 (**Higher-order semantic labelling**) Let M be a quasi-model for an IDTS $\mathcal{R}$ and $\overline{\mathcal{R}}$ the labelled IDTS with respect to M. Then $\mathcal{R}$ is terminating if and only if $\overline{\mathcal{R}} \cup \mathsf{Decr}$ is terminating.

Proof For both directions, we prove contrapositions. [⇐]: By Prop. 3.6. [⇒]: By erasing all labels in rewrite steps. □

4. Termination of λX via Higher-Order Semantic Labelling

As an application of higher-order semantic labelling, we prove the termination of an explicit substitution system λX-calculus [5, 4]. The λX-calculus uses explicit names and is known as the simplest explicit substitution system; it has no composition of substitutions. The following is the λX-calculus taken from [4].

$$\begin{aligned}
(\lambda x.M)N &\to M\langle x := N\rangle\\
(MN)\langle x := K\rangle &\to M\langle x := K\rangle\ N\langle x := K\rangle\\
(\lambda y.M)\langle x := K\rangle &\to \lambda y.M\langle x := K\rangle \qquad \text{if } x \neq y\\
x\langle x := K\rangle &\to K\\
M\langle x := K\rangle &\to M \qquad \text{if } x \notin \mathsf{FV}(M)
\end{aligned}$$

The λX-calculus has good properties: confluence and preservation of strong normalisation (PSN), and the simply typed λX-calculus is strongly normalising.

Proving termination of the simply-typed λX-calculus is not an obvious task. Actually, the direct proofs using the reducibility method are rather involved and long [15, 23]. Moreover, the λX-calculus as an IDTS does not follow the General Schema [1] directly since a precedence between the application and the explicit substitution operators cannot be determined from the rules. It is also difficult to interpret these rules by some "measure" that is strictly decreasing by a well-founded order. A candidate of such interpretation is to interpret λX-terms as simply-typed λ-terms. But we immediately find that this does not succeed since the substitution propagation rules are interpreted as the same λ-terms, i.e. not decreasing. In this section, we give a short termination proof of $\mathcal{R}_{\lambda\mathrm{X}}$ via higher-order semantic labelling.

4.1 Formulating λX-calculus as an Inductive Data Type System

First, we formulate the simply-typed λX-calculus as an IDTS. This means to give an inductive type for λX-terms. We fix the set BType of base types for the λX-calculus. We choose the IDTS-alphabet consisting of the following IDTS's base types $\mathcal{B}$ and signature $\Sigma_{\lambda\mathrm{X}}$:

$$\begin{array}{ll}
\mathrm{BType} & \ni \iota\\
\mathcal{B} & \ni \sigma,\tau ::= \iota \mid \mathsf{Var}_\tau \mid \mathsf{Arr}(\sigma,\tau)\\
\mathsf{v}_\sigma & : \mathsf{Var}_\sigma \to \sigma\\
\mathsf{ap}_{\sigma,\tau} & : \mathsf{Arr}(\sigma,\tau),\sigma \to \tau\\
\lambda_{\sigma,\tau} & : (\mathsf{Var}_\sigma \to \tau) \to \mathsf{Arr}(\sigma,\tau)\\
\mathsf{xsub}_{\sigma,\tau} & : (\mathsf{Var}_\sigma \to \tau),\sigma \to \tau
\end{array}$$

The type Var_σ represents the type of λX's variables of type σ, and the type $\mathsf{Arr}(\sigma,\tau)$ represents the λX's arrow type. We may also assume constants of type Var_σ.

The use of a type $\mathsf{Var}_\sigma \to \tau$ to represent binders of λ-terms is known in the field of mechanized reasoning. It is an essential idea to implement *higher-order abstract syntax* (HOAS) that *admits induction* in Coq [9]. Semantic analysis uncovers algebraic structures of HOAS: Fiore, Plotkin and Turi's binding algebra [11], which is also used as the semantic basis of our theory. This relates to modeling arrow types by the context extension: actually, $\delta_\sigma A \cong \mathrm{V}_\sigma \to A$ holds [11, 10].

This inductive representation is important for the termination proof. The General Schema requires *strict positivity* and an existence of a well-founded order on base types as its assumption [3, 1]. The signature $\Sigma_{\lambda\mathrm{X}}$ actually satisfies these conditions by taking $\mathsf{v}, \mathsf{ap}, \lambda$ as the constructors of Arr-types. The reason we do not just reuse the IDTS's arrow type $\sigma \to \tau$ for the λX's arrow type is that it does not satisfy such strict positivity condition. A naive choice $\lambda_{\sigma,\tau} : (\sigma \to \tau) \to (\sigma \to \tau)$ for abstractions has a negative occurrence of a type (consider $\lambda_{\sigma,\sigma}$). Precise definitions of these conditions on the General Schema are postponed to Sect. 5.

The λX-calculus is formulated as the IDTS $(\Sigma_{\lambda\mathrm{X}}, \mathcal{R}_{\lambda\mathrm{X}})$ given in Fig. 1 which does not follow the General Schema yet. For readability, we write $s\langle x := t\rangle$ for $\mathsf{xsub}_{\sigma,\tau}(x^{\mathsf{Var}_\sigma}.\, s, t)$ and omit types.

4.2 Termination proof

First we observe that the IDTS $\mathcal{R}_{\lambda\mathrm{X}}$ has a quasi-model consisting of all simply-typed λ-terms, which will be used as the label structure. Let $\vdash_\lambda$ denote the judgment relation of the simply-typed λ-calculus. We define the translation $\widehat{(-)}$ from the IDTS's base types $\mathcal{B}$ to simple types as follows:

$$\widehat{\iota} = \iota \qquad \widehat{\mathsf{Var}_\tau} = \widehat{\tau} \qquad \widehat{\mathsf{Arr}(\sigma,\tau)} = \widehat{\sigma} \to \widehat{\tau}.$$

$$
\begin{aligned}
\mathsf{ap}_{(\lambda x.s)t}(\lambda(x.M(x)), N) &\to M(x)\langle x := N\rangle_{s[x:=t]} \\
\mathsf{ap}_{s\,t}(M(x), N(x))\langle x := K\rangle_{s[x:=u]t[x:=u]} &\to \mathsf{ap}_{s[x:=u]t[x:=u]}\ (M(x)\langle x := K\rangle_{s[x:=u]}, N(x)\langle x := K\rangle_{t[x:=u]}) \\
\lambda(y.M(x,y))\langle x := K\rangle_{(\lambda y.s)[x:=u]} &\to \lambda(x.M(y,x)\langle y := K\rangle_{s[x:=u]}) \\
\mathsf{v}(x)\langle x := K\rangle_u &\to K \\
M\langle x := K\rangle_s &\to M.
\end{aligned}
$$

The set Decr of decreasing rules is consisting of the rules for all $s > t \in S$

$$
\begin{aligned}
\mathsf{ap}_s(M, N) &\to \mathsf{ap}_t(M, N) \\
M(x)\langle x := K\rangle_s &\to M(x)\langle x := K\rangle_t\ .
\end{aligned}
$$

Figure 2. The labelled IDTS $\overline{\mathcal{R}}_{\lambda\mathrm{X}} \cup \mathsf{Decr}$

For an IDTS's context Γ, we denote by $\hat{\Gamma}$ a λ-calculus's context obtained by applying this translation to each type in Γ. We define the presheaf Λ of all λ-terms by $\Lambda_\tau(\Gamma) = \{t \mid \hat{\Gamma} \vdash_\lambda t : \hat{\tau}\}$. The presheaf Λ has an obvious $\Sigma_{\lambda\mathrm{X}}$-term algebra structure that interprets the symbols $\mathsf{v}, \lambda, \mathsf{ap}$ in $\lambda\mathrm{X}$ as variable, abstraction, application operations in the λ-calculus:

$$
\begin{aligned}
\mathsf{v}^\Lambda_\tau(\Gamma)(x^{\mathsf{Var}_\tau}) &= x^\tau \\
\lambda^\Lambda_{\sigma,\tau}(\Gamma)(t) &= \lambda x^\sigma.t \\
\mathsf{ap}^\Lambda_{\sigma,\tau}(\Gamma)(s,t) &= s\ t
\end{aligned}
$$

and the operation for the explicit substitution

$$
\begin{aligned}
&\mathsf{xsub}^\Lambda_{\sigma,\tau}(\Gamma) : \Lambda_\tau(\Gamma + \mathsf{Var}_\sigma) \times \Lambda_\sigma(\Gamma) \to \Lambda_\tau(\Gamma); \\
&\mathsf{xsub}^\Lambda_{\sigma,\tau}(\Gamma)(s,t) = s[x := t]
\end{aligned}
$$

that interprets explicit substitution as the "meta-level" substitution. This shows that the presheaf Λ forms a $\mathrm{V} + \Sigma_{\lambda\mathrm{X}}$-algebra.

We define the family of partial orders $\succcurlyeq$ on Λ by $s \succcurlyeq_{\tau,\Gamma} t \Leftrightarrow s \to^*_\beta t$ for $\Gamma \vdash_\lambda s : \tau,\ \Gamma \vdash_\lambda t : \tau$. Then $(\Lambda, \succcurlyeq)$ is a weakly monotone $\Sigma_{\lambda\mathrm{X}}$-algebra. Since the unique homomorphism $\mathrm{X}(-)$ from the initial $\mathrm{V} + \Sigma_{\lambda\mathrm{X}}$-algebra $T_{\Sigma_{\lambda\mathrm{X}}}\mathrm{V}$ to the $\mathrm{V} + \Sigma_{\lambda\mathrm{X}}$-algebra Λ evaluates all explicit substitutions (cf. [12] for more detailed treatment of algebraic characterisation of explicit substitutions), for any ground (w.r.t. metavariables) instance $\Gamma \vdash l \to r : \tau$ of rules of $\mathcal{R}_{\lambda\mathrm{X}}$, we have

$$
\mathrm{X}_{\tau,\Gamma}(l) \succcurlyeq \mathrm{X}_{\tau,\Gamma}(r).
$$

Thus $(\Lambda, \succcurlyeq)$ is a quasi-model of $(\Sigma_{\lambda\mathrm{X}}, \mathcal{R}_{\lambda\mathrm{X}})$.

Using this, we attach labels to the IDTS $\mathcal{R}_{\lambda\mathrm{X}}$. We label the function symbols ap and xsub. We choose the semantic label set $S = S_{\mathsf{ap}} = S_{\mathsf{xsub}} = \bigcup_{\tau,\Gamma} \Lambda_\tau(\Gamma)$. Define the well-founded partial order on S by $s \geq t \Leftrightarrow s\ (\to_\beta \cup \rhd)^*\ t$. The semantic label maps are defined by $\langle\!\langle s,t\rangle\!\rangle^{\mathsf{ap}}_\Gamma = s\,t$, $\langle\!\langle s,t\rangle\!\rangle^{\mathsf{xsub}}_\Gamma = s[x := t]$, which are weakly monotone. The labelled IDTS $\overline{\mathcal{R}}_{\lambda\mathrm{X}}$ consists of the rules generated by assigning to $M \mapsto s, N \mapsto t, K \mapsto u$ for all $s,t,u \in S$ given in Figure 2.

We take the precedence $\sqsupset$ ordered by for all $s > t \in S$, $\mathsf{ap}_s \sqsupset \mathsf{xsub}_t \sqsupset \mathsf{ap}_t \sqsupset \lambda$. Since the base types $\mathcal{B}$ is not changed in the labelled system, the positivity conditions are satisfied. Then, $\overline{\mathcal{R}}_{\lambda\mathrm{X}} \cup \mathsf{Decr}$ follows the General Schema. Hence $\mathcal{R}_{\lambda\mathrm{X}}$ is terminating.

The idea of the use of labelling to prove termination of $\lambda\mathrm{X}$ is also taken in [4], where a transformation of $\lambda\mathrm{X}$ to a first-order TRS and a variant of first-order semantic labelling are used. The crucial difference is that we have used higher-order semantics (λ-terms) rather than first-order one given by the maximal reduction lengths of λ-terms in [4].

Our approach to use a term model has generality not only for the case of $\lambda\mathrm{X}$. In the next section, we will consider other applications.

5. Labelling with Higher-Order Term Model

In the termination proof of $\lambda\mathrm{X}$, we have used the term model Λ for labelling. In the first-order case, semantic labelling with a term model were considered under the name *self-labelling* [27] by Middeldorp, Ohsaki, and Zantema, and showed its usefulness by applying it to several different termination problems of TRSs. In this section, we extend it to the higher-order case. We first review some notions of the assumption of the General Schema [3, 1].

Definition 5.1 Let $(\mathcal{B}, \Sigma, Z)$ be an IDTS-alphabet. We assume that each base type $\tau \in \mathcal{B}$ has an associated set of *constructors* taken from Σ. The set of all declarations of constructors defines a preorder $\leq_\mathcal{B}$ on $\mathcal{B}$ by $\tau \leq_\mathcal{B} \sigma$ if τ occurs in $\vec{\alpha}$ for a constructor $c : \vec{\alpha} \to \sigma$. Any constructor must not be the root symbol of left-hand side of a rewrite rule. Let $=_\mathcal{B}$ be the equivalence relation generated by $\leq_\mathcal{B}$. An inductive type σ is *strictly positive* if for any type τ s.t. $\tau =_\mathcal{B} \sigma$,

τ does not occur at a negative position of the type of a constructor c for σ.

A metavariable Z is *accessible* in a meta-term t if there are distinct bound variables $\vec{x}$ such that $\mathsf{Z}(\vec{x}) \in \mathsf{Acc}(t)$, where $\mathsf{Acc}(t)$ is the least set satisfying the following clauses:

(i) $t \in \mathsf{Acc}(t)$.
(ii) if $x.t \in \mathsf{Acc}(t)$ then $t \in \mathsf{Acc}(t)$.
(iii) if $c(\vec{s}) \in \mathsf{Acc}(t)$ then each $s_i \in \mathsf{Acc}(t)$.
(iv) if $f(\vec{s}) \in \mathsf{Acc}(t)$ and s_i is of basic type then $s_i \in \mathsf{Acc}(t)$.
(v) if $\mathsf{Z}(\vec{s}) \in \mathsf{Acc}(t)$ and Z does not occur in $\vec{s}$ and t, then each $s_i \in \mathsf{Acc}(t)$.

More schematically, a metavariable Z is accessible in t if Z appears as like $t = f(\ldots, c(\ldots, \vec{x}.\mathsf{Z}(\vec{s}), \ldots), \ldots)$, where any constructor c can be nested any number of (including 0) times. So, for example, M is accessible in $\mathsf{ap}(\lambda(x.M(x)), N)$. Actually, all metavariables in the left-hand sides of $\mathcal{R}_{\lambda \mathrm{X}}$ are accessible. Next we introduce some notions for termination of IDTSs.

Definition 5.2 (**Solid meta-term**) A meta-term t is called *solid* if for each meta-application $\mathsf{Z}(s_1, \ldots, s_n)$ in t, all s_i do not contain function symbols.

The notion of solid meta-term is crucial in our theory presented below. This ensures that in many cases unexpected vanishing of redexes, which is a typical pitfall of termination proofs, does not happen. See Remark 5.7.

Definition 5.3 We also call an IDTS $(\mathcal{B}, \Sigma, \mathcal{R})$ *solid* if (i) all types in $\mathcal{B}$ are strictly positive and $\leq_{\mathcal{B}}$ is well-founded, and for each rule $l \to r \in \mathcal{R}$, (ii) all metavariables in r are accessible in l, and (iii) r is solid.

Note that the left-hand side l is already solid since so are Miller's higher-order patterns. Fortunately, the conditions of solid IDTSs are not so restrictive requirements in practice. An example of solid IDTS is the $\lambda\mathrm{X}$-calculus $\mathcal{R}_{\lambda \mathrm{X}}$. The simply-typed $\lambda\beta$-calculus is also solid. All examples except for recursor rules in [3, 1] are solid. It also means that there are terminating IDTSs that are not solid.

Definition 5.4 We call an IDTS $\mathcal{R}$ *precedence terminating* if there exists a well-founded order $\sqsupset$ on function symbols (called a *precedence*) such that for every rule $f(\vec{t}) \to r \in \mathcal{R}$, and every $g \in \mathsf{Fun}(r)$, $f \sqsupset g$, where $\mathsf{Fun}(r)$ denotes the set of all function symbols in r.

Lemma 5.5 Every precedence terminating solid IDTS is terminating.

Proof For each $l \to r \in \mathcal{R}$, by the definition of the General Schema and an easy induction argument, r is in a computable closure [1] of l. □

The solid property is required to use the General Schema in the above proof. Currently, we do not know how it is weakened. The next theorem states that any terminating solid IDTS can be transformed into a precedence terminating IDTS by labelling.

Theorem 5.6 Let $(\Sigma, \mathcal{R})$ be a terminating solid IDTS and $(\overline{\Sigma}, \overline{\mathcal{R}})$ its labelled IDTS by the quasi-model $(T_\Sigma \mathrm{V}, \to^*_{\mathcal{R}})$. Then, $(\overline{\Sigma}, \overline{\mathcal{R}} \cup \mathsf{Decr})$ is precedence terminating.

Proof Define the semantic label set for each $f \in \Sigma$ by $S_f = S = \bigcup_{\tau,\Gamma} T_\Sigma \mathrm{V}_\tau(\Gamma)$ with the well-founded partial order $\geq = (\to_{\mathcal{R}} \cup \rhd)^*$, and the semantic labelling map

$$\langle\!\langle \vec{s} \rangle\!\rangle^f = f(\vec{s}).$$

By semantic labelling with S, we have $(\overline{\Sigma}, \overline{\mathcal{R}})$. Define the precedence on $\overline{\Sigma}$ by

$$f_s \sqsupset g_t \quad \Leftrightarrow \quad s > t.$$

We check that $\overline{\mathcal{R}} \cup \mathsf{Decr}$ is precedence terminating.

- Case $\bar{l} \to \bar{r} \in \overline{\mathcal{R}}$. There exist $Z \vdash l \to r \in \mathcal{R}$ and a valuation $\phi : Z \longrightarrow T_\Sigma \mathrm{V}$ such that $\bar{l} = \phi^{\mathsf{L}} l \to \phi^{\mathsf{L}} r = \bar{r} \in \overline{\mathcal{R}}$. The label of the root function symbol of $\bar{l}$ is $\phi^* l$. Any label p of a function symbol in $\bar{r}$ is obtained by $p = \phi^*(t)$ for some subterm $t \trianglelefteq r$. Then

$$\phi^* l \to_{\mathcal{R}} \phi^* r \trianglerighteq \phi^* t = p. \tag{1}$$

- Case $f_s(\vec{Z}) \to f_t(\vec{Z}) \in \mathsf{Decr}$ with $s > t$. Hence $f_s \sqsupset f_t$.

□

Remark 5.7 The condition that r must be *solid* is essential to establish the subterm property in (1). In higher-order case, the subterm relation is not closed under valuations, i.e. $s \rhd t \;\Rightarrow\; \phi^* s \rhd \phi^* t$ does not hold in general, whereas in first-order case it obviously holds. Consider $\mathsf{Z}(f) \rhd f$ where f is a function symbol. Applying a valuation $\mathsf{Z} \mapsto c$, we have $c \not\rhd f$. This is due to the fact that $\mathsf{Z}(f)$ is not solid.

The proof of Thm. 5.6 is important. This particular semantic labelling, i.e. taking a term algebra as semantics and thus labelling function symbols with terms is useful to show preservation of termination of extension of a terminating system. For instance, the termination proof of $\lambda\mathrm{X}$-calculus given in Sect. 4 is an example of semantic labelling with a term algebra along this scheme.

5.1 Currying transformation

Kennaway, Klop, Sleep, and de Vries showed the preservation of termination under currying in [20] for first-order TRSs. Then, Middeldorp et al. gave alternative proof using self-labelling [27]. In this subsection we extend this result to the higher-order case via higher-order semantic labelling.

Definition 5.8 Let $(\mathcal{B}, \Sigma, \mathcal{R})$ be an IDTS. Define the new base type set $\mathcal{B}_{\mathsf{ap}}$ by

$$\mathcal{B}_{\mathsf{ap}} \ni \tau ::= \iota \mid \mathsf{Arr}(\alpha, \tau) \qquad \alpha \in I(\mathcal{B})$$

Define the signatures by

$$\begin{aligned}
\Sigma_{\mathsf{p}} &= \{f_i : \alpha_1, \ldots, \alpha_i \to \mathsf{Arr}(\alpha_{i+1}, \ldots, \mathsf{Arr}(\alpha_n, \tau)) \\
&\quad \mid f : \alpha_1, \ldots, \alpha_n \to \tau \in \Sigma,\ f \neq \mathsf{ap}\} \\
\Sigma_{\mathsf{a}} &= \{\mathsf{ap}_{\alpha,\tau} : \mathsf{Arr}(\alpha, \tau), \alpha \to \tau \mid \alpha \in I(\mathcal{B}),\ \tau \in \mathcal{B}_{\mathsf{ap}}\} \\
\Sigma_{\mathsf{ap}} &= \Sigma \cup \Sigma_{\mathsf{a}} \cup \Sigma_{\mathsf{p}},
\end{aligned}$$

and assume that all f_i $(i \neq n)$ are constructors. The IDTS $(\mathcal{B}_{\mathsf{ap}}, \Sigma_{\mathsf{ap}}, \mathcal{R}_{\mathsf{ap}})$ is the union of $\mathcal{R}$ and the set of the following rules, called *ap-rules*:

$$\mathsf{ap}(f_i(\mathrm{Z}_1, \ldots, \mathrm{Z}_i), \mathrm{Z}) \to f_{i+1}(\mathrm{Z}_1, \ldots, \mathrm{Z}_i, \mathrm{Z})$$

for all $i = 1, \ldots, n-1,\ f : \sigma_1, \ldots, \sigma_n \to \tau \in \Sigma$, where f_n is written as f.

Namely, $\mathcal{R}_{\mathsf{ap}}$ has all curried functions generated from Σ. A problem is whether termination is preserved under this addition. One would expect that this should hold, but its proof is not entirely obvious. A naive approach based on simulation of $\mathcal{R}_{\mathsf{ap}}$-reduction by $\mathcal{R}$-reduction is not applicable since 1-step $\mathcal{R}_{\mathsf{ap}}$-reduction using ap-rule may become 0-step $\mathcal{R}$-reduction. The first proof of this theorem for TRSs in [20] needs rather involved analysis (about four page long). Semantic labelling with higher-order terms is again useful.

Theorem 5.9 Assume all $f_i \notin \Sigma$, and ap does not appear in $\mathcal{R}$. If a solid IDTS $(\mathcal{B}, \Sigma, \mathcal{R})$ is terminating, its curried version $(\mathcal{B}_{\mathsf{ap}}, \Sigma_{\mathsf{ap}}, \mathcal{R}_{\mathsf{ap}})$ is terminating.

Proof Define the weak monotone Σ_{ap}-algebra

$$\mathcal{T} = (T_{\Sigma_{\mathsf{ap}}}\mathrm{V}, \to^*_{(\Sigma_{\mathsf{ap}}, \mathcal{R})})$$

with the operation for $\mathsf{ap}_{\alpha,\tau} : \mathsf{Arr}(\alpha, \tau), \alpha \to \tau$ where $\alpha = \vec{a} \to b$ by

$$\begin{aligned}
&\mathsf{ap}^{\mathcal{T}}_{\tau} : T_{\Sigma_{\mathsf{ap}}}\mathrm{V}_{\mathsf{Arr}(\alpha,\tau)} \times \delta_{\vec{a}}(T_{\Sigma_{\mathsf{ap}}}\mathrm{V})_b \to T_{\Sigma_{\mathsf{ap}}}\mathrm{V}_{\tau} \\
&\mathsf{ap}^{\mathcal{T}}_{\tau}(\Gamma)(x, t) = \mathsf{ap}(x, t) \quad \text{if } x \in \Gamma \\
&\mathsf{ap}^{\mathcal{T}}_{\tau}(\Gamma)(f_i(s_1, \ldots, s_i), t) = f_{i+1}(s_1, \ldots, s_i, t) \quad \text{if } i \neq n \\
&\mathsf{ap}^{\mathcal{T}}_{\tau}(\Gamma)(f(s_1, \ldots, s_i), t) = \mathsf{ap}(f(s_1, \ldots, s_i), t) \quad \text{if } f \neq f_i
\end{aligned}$$

and the usual term algebra interpretations for symbols in Σ. Monotonicity of $\mathsf{ap}^{\mathcal{T}}$ is shown by induction on the proof trees of $\to_{\mathcal{R}}$. Then, $\mathcal{T}$ is a quasi-model of $(\Sigma_{\mathsf{ap}}, \mathcal{R}_{\mathsf{ap}})$ since (i) for each rule in $\mathcal{R}$, it is clearly decreasing by $\to_{\mathcal{R}}$, (ii) for each ap-rule, both sides of the rule are interpreted as $f_{i+1}(s_1, \ldots, s_i, t)$.

Take the semantic label set $S = \bigcup_{\sigma,\Gamma} T_{\Sigma_{\mathsf{ap}}}\mathrm{V}_{\sigma}(\Gamma)$ with the well-founded partial order $s \geq t \ \Leftrightarrow\ s \left(\to_{(\Sigma_{\mathsf{ap}}, \mathcal{R})} \cup \rhd\right)^* t$. By applying semantic labelling with S, we have the labelled IDTS $\overline{\mathcal{R}_{\mathsf{ap}}}$, in which each label is a term obtained by evaluating all ap's. Take the precedence

$$\mathsf{ap}_s \sqsupset \mathsf{ap}_t \sqsupset f_s \sqsupset f_t \qquad \forall f(\neq \mathsf{ap}) \in \Sigma_{\mathsf{ap}}, \forall s > t \in S$$

Then, $\overline{\mathcal{R}_{\mathsf{ap}}} \cup \mathsf{Decr}$ is precedence terminating since (a) for each rule in $\overline{\mathcal{R}}$, similar to the proof of Thm. 5.6, (b) for each labelled ap-rule, ap_s is greater than f_s. Moreover, since each f_i is a constructor, all ap-rules are solid, (c) for each rule in Decr, similar to the proof of Thm. 5.6. By Lemma 5.5, $\mathcal{R}_{\mathsf{ap}}$ is terminating. □

5.2 Modularity with Higher-Order Recursive Program Schemes

A recursive program scheme (RPS) (cf. survey [7]) is a simple form of *first-order* functional program, which is given as a set of first-order rewrite rules of the form

$$f(x_1, \ldots, x_n) \to t$$

with distinct x's such that for every function symbol f in a signature there exists at most one such rule.

We consider its higher-order extension. By a *higher-order recursive program scheme* (HO-RPS), we mean an IDTS $(\Sigma, \mathcal{R}, Z)$ consisting of the following form of rules with distinct metavariable Z's and the same condition for f as RPS:

$$f(1 \ldots i_1.\mathrm{Z}_1(1, \ldots, i_1),\ \ldots,\ 1 \ldots i_n.\mathrm{Z}_n(1, \ldots, i_n)) \ \to\ t$$

for $f : (\vec{a}_1 \to b_1), \ldots, (\vec{a}_l \to b_l) \to b \in \Sigma$. Notice that now t is a (higher-order) meta-term.

Using semantic labelling with a higher-order term model and precedence termination, we prove the following: the combination of an arbitrary terminating solid IDTS and a terminating solid HO-RPS that defines new functions is terminating. To the author's best knowledge, this is a new result on modularity of termination of higher-order rewrite rules.

Modularity of higher-order rewriting is known as a difficult and subtle problem. Van Oostrom observes that almost no property is modular by finding many tricky counterexamples to modularity of higher-order rewriting systems [30]. He also remarks that extending a signature of a terminating CRS does not necessarily preserves termination [29, 30]. However, our new notion of *solid* system successfully avoids such a counterexample. First, we show a lemma that states termination is preserved under signature extension for solid HO-RPSs.

Lemma 5.10 Let $(\mathcal{B}, \Theta, \mathcal{S})$ be a terminating solid HO-RPS, $(\mathcal{B}', \Theta')$ an alphabet such that all types in $\mathcal{B} \cup \mathcal{B}'$ are strictly positive, and $\leq_{\mathcal{B} \cup \mathcal{B}'}$ is well-founded. Then, the solid IDTS $(\mathcal{B} \cup \mathcal{B}', \Theta \cup \Theta', \mathcal{S})$ is terminating.

Proof It is easy to see that for a terminating solid HO-RPS $(\mathcal{B}, \Theta, \mathcal{S})$, the precedence $\sqsupset$ defined in Def. 5.4 is well-founded order on Θ. Hence $(\mathcal{B}, \Theta, \mathcal{S})$ is also precedence

terminating. Then, $(\mathcal{B} \cup \mathcal{B}', \Theta \cup \Theta', \mathcal{S})$ is again precedence terminating using the same precedence $\sqsupset$ regarded as a strict partial order on $\Theta \cup \Theta'$, since the rules are unchanged. $\square$

Theorem 5.11 Let $(\mathcal{B}, \Sigma, \mathcal{R})$ be a terminating solid IDTS and $(\mathcal{B}', \Theta, \mathcal{S})$ a terminating solid HO-RPS such that

(i) all types in $\mathcal{B} \cup \mathcal{B}'$ are strictly positive, and $\leq_{\mathcal{B} \cup \mathcal{B}'}$ is well-founded,

(ii) $\Sigma \cap \Theta_{\text{def}} = \varnothing$,

where Θ_{def} is the subset of Θ consisting of all f such that there is a corresponding rule in Θ. Then $(\mathcal{B} \cup \mathcal{B}', \Sigma \cup \Theta, \mathcal{R} \cup \mathcal{S})$ is terminating.

Proof We denote by $t{\downarrow_{\mathcal{S}}}$ the normal form of t by the rewrite $\rightarrow_{(\Sigma \cup \Theta, \mathcal{S})}$. Let $\Sigma' = (\Sigma \cup \Theta) - \Theta_{\text{def}}$. We show that if $(\Sigma', \mathcal{R})$ is terminating then $(\Sigma \cup \Theta, \mathcal{R} \cup \mathcal{S})$ is terminating by following the scheme of the proof of Thm. 5.6. In this case we use the $\Sigma \cup \Theta$-algebra $\mathcal{T} = (T_{\Sigma'}\mathrm{V}, \rightarrow^*_{(\Sigma', \mathcal{R})})$ defined for $f \in \Theta_{\text{def}}$ by

$$f^{\mathcal{T}}(\vec{t}) = f(\vec{t}){\downarrow_{\mathcal{S}}}$$

and other operations are defined by the usual term algebra operations. The operation $f^{\mathcal{T}}$ is well-defined by the following reasons. Since $(\Sigma \cup \Theta, \mathcal{S})$ is orthogonal, $(\Sigma \cup \Theta, \mathcal{S})$ is confluent [21]. Since $(\mathcal{B}, \Theta, \mathcal{S})$ is terminating, $(\mathcal{B} \cup \mathcal{B}', \Sigma \cup \Theta, \mathcal{S})$ is also terminating by Lemma 5.10. Since $(\mathcal{B} \cup \mathcal{B}', \Sigma \cup \Theta, \mathcal{S})$ is terminating and confluent, the $\mathcal{S}$-normal form $f(\vec{t}){\downarrow_{\mathcal{S}}}$ is unique and always in $T_{\Sigma'}\mathrm{V}$ (i.e. all Θ_{def}-symbols are consumed).

We show that $\mathcal{T}$ is a quasi-model of $(\Sigma \cup \Theta, \mathcal{R} \cup \mathcal{S})$.

(i) Weak monotonicity of $\mathcal{T}$. For $f \in \Sigma'$, its operation is itself, so monotone. For $f \in \Theta_{\text{def}}$, by Lemma B.2 in Appendix.

(ii) We show that $\mathcal{T}$ satisfies all rules in $\mathcal{R} \cup \mathcal{S}$. Take $\theta : Z \longrightarrow T_{\Sigma'}\mathrm{V}$.

(ii-a) Case $l \rightarrow r \in \mathcal{R}$. $\theta^* l \rightarrow_{\mathcal{R}} \theta^* r$.

(ii-b) Case $l \rightarrow r \in \mathcal{S}$. Since $l\theta \rightarrow_{\mathcal{S}} r\theta$,

$$\theta^* l = (l\theta){\downarrow_{\mathcal{S}}} = (r\theta){\downarrow_{\mathcal{S}}} = \theta^* r.$$

For a well-founded poset of semantic labels, we take

$$(\bigcup_{\sigma, \Gamma} T_{\Sigma'}\mathrm{V}_{\sigma}(\Gamma),\ (\rightarrow_{(\Sigma', \mathcal{R})} \cup \rhd)^*).$$

The precedence is defined by $f_s \sqsupset g_t$ if

(i) for $f, g \in \Sigma'$ or $f, g \in \Theta_{\text{def}}$, $s\ (\rightarrow_{(\Sigma', \mathcal{R})} \cup \rhd)^+\ t$, or

(ii) for $f \in \Theta_{\text{def}}, g \in \Theta$ such that $f(\vec{\mathsf{Z}}) \rightarrow r \in \mathcal{S}$ and $g \in \mathsf{Fun}(r)$.

Then, it is easy to see that $\overline{\mathcal{R}} \cup \overline{\mathcal{S}} \cup \mathsf{Decr}$ is precedence terminating. $\square$

This theorem is applicable to any terminating solid IDTS and HO-RPS, i.e. even one that does not follow the General Schema. Hence this theorem is not obtained by a direct application of the General Schema criterion.

6. Conclusion

We have presented a theory of higher-order semantic labelling for inductive data type systems (IDTSs). The technical novelty is to handle metavariables and higher-order functions in IDTSs for the labelling process. This is done by exploiting typed binding algebras and the author's previous results on algebraic semantics of higher-order syntax [13] and higher-order rewriting [14]. Typed binding algebras are algebras in the presheaf category $(\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{B}})^{\mathcal{B}}$ where $\mathcal{B}$ is a set of types and $\mathbb{F}$ is skeleton of the category of finite sets modeling binding in the style of Fiore, Plotkin and Turi [11].

We have given a number of applications of higher-order semantic labelling: it shows termination of the λX-calculus, preservation of termination under currying and a new modularity result. This shows usefulness of higher-order semantic labelling.

References

[1] F. Blanqui. Termination and confluence of higher-order rewrite systems. In *Rewriting Techniques and Application (RTA 2000)*, LNCS 1833, pages 47–61. Springer, 2000.

[2] F. Blanqui. (HO)RPO revisited. Technical Report 5972, INRIA, 2006. Research report.

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

[4] R. Bloo and H. Geuvers. Explicit substitution: on the edge of strong normalization. *Theor. Comput. Sci. 211*, 1-2:375–395, 1999.

[5] R. Bloo and K. H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *CSN-95: Computer Science in the Netherlands*, pages 62–72, 1995.

[6] C. Borralleras and A. Rubio. A monotonic higher-order semantic path ordering. In *Procs. 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, LNCS 2250, pages 531–547, 2001.

[7] B. Courcelle. Recursive application schemes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 459–492. 1990. Chapter 9, vol. B.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] N. Ghani, T. Uustalu, and M. Hamana. Explicit substitutions and higher-order syntax. *Higher-Order and Symbolic Computation*, 19(2/3):263–282, 2006.

[13] M. Hamana. Free Σ-monoids: A higher-order syntax with metavariables. In *Asian Symposium on Programming Languages and Systems (APLAS 2004)*, LNCS 3302, pages 348–363, 2004.

[14] M. Hamana. Universal algebra for termination of higher-order rewriting. In *Proceedings of 16th International Conference on Rewriting Techniques and Applications (RTA'05)*, LNCS 3467, pages 135–149. Springer, 2005.

[15] D. J.Dougherty and P. Lescanne. Reductions, intersection types, and explicit substitutions. *Mathematical Structures in Computer Science*, 13(1):55–85, 2003.

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

[17] J.-P. Jouannaud and A. Rubio. Higher-order orderings for normal rewriting. In *Proc. of RTA'06*, LNCS 4098, pages 387–399, 2006.

[18] J.-P. Jouannaud and A. Rubio. Polymorphic higher-order recursive path orderings. *Journal of ACM*, 54(1), 2007.

[19] S. Katsumata. A generalisation of pre-logical predicates to simply typed formal systems. In *ICALP*, pages 831–845, 2004.

[20] J.R. Kennaway, J.W. Klop, M.R. Sleep, and F.J. de Vries. On comparing curried and uncurried rewrite systems. *Journal of Symbolic Computation*, 21(1):15–39, 1996.

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

[22] J.W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: Introduction and survey. *Theor. Comput. Sci.*, 121(1&2):279–308, 1993.

[23] S. Lengrand, P. Lescanne, D. J.Dougherty, Mariangiola Dezani-Ciancaglini, and Steffen van Bakel. Intersection types for explicit substitutions. *Information and Computation*, 189(1):17–42, 2004.

[24] 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, pages 294–308. Springer, 1995.

[25] S. Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1971.

[26] M. Miculan and I. Scagnetto. A framework for typed HOAS and semantics. In *Fifth ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP'03)*, pages 184–194. ACM Press, 2003.

[27] A. Middeldorp, H. Ohsaki, and H. Zantema. Transforming termination by self-labelling. In *Proceedings of the 13th International Conference on Automated Deduction*, LNCS 1104, pages 373–386, 1996.

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

[29] V. van Oostrom. *Confluence for Abstract and Higher-Order Rewriting*. PhD thesis, Vrije Universiteit, Amsterdam, 1994.

[30] V. van Oostrom. Counterexamples to higher-order modularity. Draft, 2005.

[31] J. van de Pol. Termination proofs for higher-order rewrite systems. In *the First International Workshop on Higher-Order Algebra, Logic and Term Rewriting (HOA'93)*, LNCS 816, pages 305–325, 1994.

[32] F. van Raamsdonk. On termination of higher-order rewriting. In *Rewriting Techniques and Applications, 12th International Conference (RTA 2001)*, pages 261–275, 2001.

[33] M. Tanaka and J. Power. A unified category-theoretic formulation of typed binding signatures. In *Proc. of the 3rd ACM SIGPLAN workshop on Mechanized reasoning about languages with variable binding (MERLIN'05)*, pages 13–24, 2005.

[34] Terese. *Term Rewriting Systems*. Number 55 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2003.

[35] H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24(1/2):89–105, 1995.

A. The Σ-monoid $M_{\overline{\Sigma}}Z$

We describe the Σ-monoid structure of $M_{\overline{\Sigma}}Z$. The unit of the monoid $M_{\overline{\Sigma}}Z$ is $\nu^\phi : \mathrm{V} \to M_{\overline{\Sigma}}Z$ as described in Sect. 3. The multiplication $\beta^\phi : M_{\overline{\Sigma}}Z \bullet M_{\overline{\Sigma}}Z \longrightarrow M_{\overline{\Sigma}}Z$ is

$$\begin{aligned}
\beta^\phi_{\tau,\Gamma}(x^\tau;\, \vec{t}) &= t_x \\
\beta^\phi_{\tau,\Gamma}(\mathsf{Z}[s_1,\ldots,s_l];\, \vec{t}) &= \mathsf{Z}[\beta^\phi_{\tau,\Gamma}(s_1;\vec{t}),\ldots,\beta^\phi_{\tau,\Gamma}(s_l;\vec{t})] \\
\beta^\phi_{\tau,\Gamma}(f_q(s_1,\ldots,s_l);\, \vec{t}) & \\
&= f_p(\beta^\phi(s_1;\, \mathsf{up}_{n+i_1}(\vec{t}), n{+}1,\ldots,n{+}i_1),\ldots)
\end{aligned}$$

where

$$\begin{aligned}
&p = \langle\!\langle !\theta^* || \beta^\phi(s_1;\, \mathsf{up}_{i_1}(\vec{t}), n+1,\ldots,n+i_1) ||, \ldots \rangle\!\rangle_{b,\Gamma} \\
&f : (\vec{a}_1 \to b_1),\ldots,(\vec{a}_l \to b_l) \to b \in \Sigma, \\
&|\vec{a}_k| = i_k \quad (\text{for each } k), \qquad |\Gamma| = n, \qquad |\vec{t}| = m.
\end{aligned}$$

Note that the length of "$\mathsf{up}_{i_1}(\vec{t}), n+1,\ldots,n+i_1$" is $m+i_1$, and it renames a variable $m+k$ to $n+k$ for each $k = i_1,\ldots,i_l$ to make bound variables sense.

To check that $M_{\overline{\Sigma}}Z$ satisfies the monoid law is straightforward induction on meta-terms. To check the Σ-monoid law $\beta^\phi \circ ([f^\phi]_{f\in\Sigma} \bullet \mathrm{id}) = [f^\phi]_{f\in\Sigma} \circ \Sigma\beta^\phi \circ \mathrm{st}$, we instantiate this at $\Gamma \in \mathbb{F}\downarrow\mathcal{B}$ and chase an element. This eventually becomes the equality

$$\begin{aligned}
&\beta^\phi_{b,\Gamma}(f_r(s_1,\ldots,s_l);\, \vec{t}) \\
&= f_p(\beta^\phi(s_1;\, \mathsf{up}_{i_1}(\vec{t}), n{+}1,\ldots,n{+}i_1),\ldots)
\end{aligned}$$

where $r = \langle\!\langle !\theta^*|s_1|, \ldots, !\theta^*|s_l| \rangle\!\rangle_{b,\Gamma}$ and p is the one given above. This obviously holds by the definition of β^ϕ.

B. Lemmas for Proposition 5.11

Lemma B.1 Let $\theta : Z \longrightarrow T_{\Sigma'}\mathrm{V},\ t \in M_{\Sigma\cup\Theta}Z(\Gamma)$.

$$(\theta t)\downarrow_{\mathcal{S}} = \theta(t\downarrow_{\mathcal{S}})$$

Proof Let $t\downarrow_{\mathcal{S}}$ be the normal form of t by rewriting on meta-terms by $\mathcal{S}$. Then $t\theta \rightarrow^*_{\mathcal{S}} = (t\downarrow_{\mathcal{S}})\theta$ is a $\mathcal{S}$-normal form. Suppose $t\theta \rightarrow^!_{\mathcal{S}} m$. Since $\mathcal{S}$ has the unique normal form property (UN, cf. [34]), $m = (t\downarrow_{\mathcal{S}})\theta$. Hence $(\theta t)\downarrow_{\mathcal{S}} = \theta(t\downarrow_{\mathcal{S}})$. □

Lemma B.2 Let $f \in \Theta_{\mathrm{def}}$. Then

$$s \rightarrow_{\mathcal{R}} t \ \Rightarrow\ f(\ldots, s, \ldots)\downarrow_{\mathcal{S}} \rightarrow^*_{\mathcal{R}} f(\ldots, t, \ldots)\downarrow_{\mathcal{S}} .$$

Proof By case analysis of proof trees of $\rightarrow_{\mathcal{R}}$.

(i) Case $l\theta \rightarrow_{\mathcal{R}} r\theta$ for $Z \vdash l \rightarrow r : b \in \mathcal{R},\ \theta : Z \longrightarrow T_{\Sigma'}\mathrm{V}$. We show $f(l\theta)\downarrow_{\mathcal{S}} \rightarrow_{\mathcal{R}} f(r\theta)\downarrow_{\mathcal{S}}$ for a unary f. The n-ary cases are similar. Take valuations $\theta' : M \mapsto l\theta,\ \theta'' : M \mapsto r\theta$ where M is a fresh metavariable. By Lemma B.1,

$$\begin{aligned} f(l\theta)\downarrow_{\mathcal{S}} &= (\theta' f(M))\downarrow_{\mathcal{S}} \\ &= \theta'(f(M)\downarrow_{\mathcal{S}}) \\ &\rightarrow^*_{\mathcal{R}} \theta''(f(M)\downarrow_{\mathcal{S}}) \\ &= (\theta'' f(M))\downarrow_{\mathcal{S}} \\ &= f(r\theta)\downarrow_{\mathcal{S}} . \end{aligned}$$

(ii) Case $g(\ldots, s, \ldots) \rightarrow_{\mathcal{R}} g(\ldots, t, \ldots)$. Similar to the case (i). Now we take $\theta' : M \mapsto g(\ldots, s, \ldots), \theta'' : M \mapsto g(\ldots, t, \ldots)$.

□