Mathematical Structures in Computer Science (2022), **32**, pp. 542–573
doi:10.1017/S0960129522000287

CAMBRIDGE UNIVERSITY PRESS

PAPER

Complete algebraic semantics for second-order rewriting systems based on abstract syntax with variable binding

Makoto Hamana

Faculty of Informatics, Gunma University, Maebashi, Japan
Email: hamana@gunma-u.ac.jp

(Received 21 March 2021; revised 19 August 2022; accepted 23 August 2022; first published online 14 October 2022)

Abstract

By using algebraic structures in a presheaf category over finite sets, following Fiore, Plotkin and Turi, we develop sound and complete models of second-order rewriting systems called second-order computation systems (CSs). Restricting the algebraic structures to those equipped with well-founded relations, we obtain a complete characterisation of terminating CSs. We also extend the characterisation to rewriting on meta-terms using the notion of Σ-monoid.

Keywords: Term rewriting; higher-order rewriting; termination; algebraic models; higher-order abstract syntax

1. Introduction

In 1998, Gordon Plotkin presented the theory of *binding algebras* (Plotkin, 1998), which aimed at applying ideas in universal algebra to type theory. It can be read as a possibility of a new algebraic foundation of rewriting systems on higher-order terms.

Aczel has developed a general framework of rewrite rules for calculi with variable binding called the Contraction Scheme (Aczel, 1978). Plotkin's programme of binding algebras later produced the notion of Σ-monoid (Fiore et al., 1999). It is noteworthy that the *free* Σ*-monoids* constructed in the author's earlier work (Hamana, 2004) are the same as the syntax of 'meta-expressions' defined by Aczel.

This similarity suggests that Σ-monoids might be qualified as a semantics of rewriting systems on abstract syntax with variable binding. Based on this idea, in this paper, we present a complete algebraic semantics of second-order rewriting systems called second-order computation systems (CSs) and apply it to the model-based *termination proof technique* of second-order CSs.

1.1 Higher-order abstract syntax and rewriting on higher-order terms

An earlier general work on rewriting systems on higher-order terms was done by Aczel (1978). He developed a system called the Contraction Scheme. Influenced by this, Klop proposed a general system called the Combinatory Reduction Systems (Klop, 1980). Mayr and Nipkow (1998) proposed a format of rewriting systems on higher-order terms, the higher-order rewrite system, which is a rewriting system modulo $\beta\eta$-equivalence and which uses the simply typed λ-calculus as a meta-language. This system is designed to be applicable to proof checkers and theorem proving systems because it is equipped with the typed λ-calculus. Blanqui, Jouannaud, and Okada

© The Author(s), 2022. Published by Cambridge University Press

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

proposed a rewriting system with stronger type system and λ-calculus (Blanqui et al., 1999, 2002) and added rewriting rules. Using the reducibility technique, they have developed techniques for proving termination.

In the field of automated theorem proving, there is an encoding technique called higher-order abstract syntax (HOAS) (Despeyroux et al., 1995; Pfenning and Elliott, 1988). This is a technique for coding binding data structures, especially object-level formulas, using the proof checker's λ-calculus as the meta-language. This idea is intuitive, but there was a fundamental problem: it is difficult to apply induction to HOAS.

No report of the relevant literature has seriously discussed HOAS and rewriting on higher-order terms in the same context. However, the basic idea is quite similar because both use the typed λ-calculus as the meta-language. Therefore, rewriting on higher-order terms can be regarded as rewriting on HOAS. The difficulty of applying induction to HOAS also appears in a different way in the field of rewriting systems on higher-order terms. It is fundamentally related to the fact that the models (van de Pol, 1994, 1996) given to date for proving termination of rewriting systems on higher-order terms are not complete. This is because they both use a certain function space to model the object language, which makes it difficult to apply induction and which makes it impossible to construct the term model for completeness. Therefore, finding a complete model for rewriting on higher-order terms requires a good model for HOAS. Plotkin et al. in their theory of binding algebras (Fiore et al., 1999) made a breakthrough in solving this problem on HOAS. They characterised HOAS, that is, abstract syntax with variable binding, as an initial algebra in a presheaf category and established an induction principle on abstract syntax with variable binding.

1.2 Algebraic models of structured operational semantics and variable binding

The theory of binding algebras by Plotkin et al. (1998) was presented at the international conference on rewriting systems in 1998, RTA'98, the year before their LICS paper (Fiore et al., 1999). In this sense, a connection exists between the theory of binding algebras and rewriting systems on higher-order terms.

Plotkin's work on binding algebras can be traced back to his algebraic work on the well-known operational semantics format structural operational semantics (SOS). SOS is a system for formally specifying the computation steps in programming languages and concurrent systems and has become a fundamentally important tool in the theoretical study of programming languages. Plotkin and Turi used the algebraic semantics of the SOS format to automatically derive a coincidence between the denotational semantics and the operational semantics of the language given by SOS. The language used in this study is the GSOS format of SOS, which is given by a first-order syntax and which contains no binding syntactic structures. Therefore, the structure of universal algebra and coalgebra was used. Extending this result to cover variable binding constructs is a necessary next step. It was mentioned as a plan in Turi and Plotkin (1997), because variable binding is an important syntactic construct in that programming languages and concurrent systems that are the main targets of SOS. Therefore, the theory of binding algebras by Plotkin et al. can be regarded as a preparatory step for the study of algebraic models of SOS that use expressions with variable binding. The present study can be regarded as one by which this idea is applied not to SOS but to rewriting systems. It becomes an appropriate algebraic model for rewriting systems on higher-order terms.

1.3 Rewriting systems and operational semantics

The operational semantics given by SOS and the theory of rewriting systems where the first-order term rewriting system (TRS) is the most common formalism have several similarities: they both

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

give relations of computation steps. However, as research fields, their work has proceeded almost independently of each other without any close connection. One reason for their lack of connection is that SOS covers a wide variety of expressions, allowing variable binding and substitutions, whereas TRS is restricted to first-order terms.

The theory of rewriting has accumulated numerous useful concepts and results. If it can be applied directly to the study of operational semantics without complicated encoding, it will be useful as a foundational theory of computation and programming languages.

As well as the substantive aim of providing a complete model for the termination of second-order CSs, the more conceptual aim of this paper is to provide a step towards bringing the theory of TRS and the theory of SOS closer together by providing algebraic models of second-order CSs.

Contributions. This paper is the fully reworked and extended version of the conference paper (Hamana, 2005). In this paper, we establish complete algebraic semantics of second-order rewriting systems called *second-order CSs*.

The complete characterisation of terminating CSs provides a method of proving the termination of CSs by algebraic interpretation. The following CS $\mathcal{C}$ for conversion into prenex normal form, that is, pushing quantifiers outside, is a typical example of rewrite rules that require the feature of variable binding (van de Pol, 1996):

$$\begin{array}{llll} P \wedge \forall(x.Q[x]) \Rightarrow \forall(x.P \wedge Q[x]) & & \forall(x.Q[x]) \wedge P \Rightarrow \forall(x.Q[x] \wedge P) \\ P \vee \forall(x.Q[x]) \Rightarrow \forall(x.P \vee Q[x]) & & \forall(x.Q[x]) \vee P \Rightarrow \forall(x.Q[x] \vee P) \\ P \wedge \exists(x.Q[x]) \Rightarrow \exists(x.P \wedge Q[x]) & & \exists(x.Q[x]) \wedge P \Rightarrow \exists(x.Q[x] \wedge P) \\ P \vee \exists(x.Q[x]) \Rightarrow \exists(x.P \vee Q[x]) & & \exists(x.Q[x]) \vee P \Rightarrow \exists(x.Q[x] \vee P) \\ \neg\forall(x.Q[x]) \Rightarrow \exists(x.\neg(Q[x])) & & \neg\exists(x.Q[x]) \Rightarrow \forall(x.\neg(Q[x])) \end{array}$$

Existing proof methods in the theory of higher-order rewriting to the CS $\mathcal{C}$ are not straightforwardly applicable (Jouannaud and Rubio, 2001). Alternatively, they require consideration of an involved function space to interpret binders (van de Pol, 1994, 1996). The present paper provides a simpler method of showing termination of CS such as $\mathcal{C}$, as shown in Example 9.7.

Organisation. This paper is organised as follows. We first review the technical background of our semantics in Section 2. We formally define *second-order CSs* in Section 3. Section 4 gives algebraic semantics of CS' syntax and valuations. Section 5 gives algebraic semantics of CS' rewriting. Section 6 gives algebraic semantics of CS' meta-rewriting. Section 7 shows that a known model of higher-order rewrite rules given by hereditary monotone functionals is an instance of our algebraic models. Finally, in Section 9, we investigate the termination of binding CSs and show examples of termination proofs using algebraic models.

2. Background on Algebraic Semantics of Second-order Abstract Syntax

In this section, we review the technical basis of our semantics, that is., the algebraic models of abstract syntax with binding (Fiore et al., 1999) and metavariables (Hamana, 2004), also called second-order abstract syntax (Fiore, 2008), and explain why this is suitable for modelling second-order CSs.

Moreover, as we will see in the next subsection and Example 3.2, second-order abstract syntax can encode λ-terms of arbitrary order (not only second-order). So, second-order abstract syntax and second-order CSs are suitable for modelling calculi using higher-order terms.

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

2.1 Introduction to second-order abstract syntax

2.1.1 Object and metavariables

Second-order CSs are founded on second-order abstract syntax. We introduce second-order abstract syntax. We first explain the distinction between object variables and metavariables, which is important for our modelling of CSs. For example, consider a λ-term

$$\lambda x.(My)$$

in a certain mathematical context. Here 'x' and 'y' are λ-calculus variables (i.e. *object-level variables*, because now the λ-calculus is the object system), while at the level of text, 'M' is a meta-level variable. We distinguish 'object-level variables' and 'metavariables' in this sense and call 'object-level variables' simply 'variables'.

2.1.2 Metavariables with arities

Metavariables have also the notion of arities. For example, writing

$$\lambda x.(M[x]\ y) \tag{1}$$

we mean that M may contain the variable x. In this case, we say that the metavariable M has arity 1. We formalise this notion of metavariables in this paper.

2.1.3 Function symbols with binding arities

Second-order abstract syntax provides a general framework for syntax with variable binding by a *signature* consisting of 'function symbols with binding arities'. In second-order abstract syntax, the λ-abstraction is not a primitive construct. The above λ-term is encoded by using second-order abstract syntax as follows:

$$\mathsf{abs}(x.\mathsf{app}(M[x], y))$$

where $x.-$ is the primitive variable binding construct of second-order abstract syntax. Here we assume that app is a function symbol with *binding arity* $\langle 0, 0\rangle$, which means that app takes two arguments and each argument binds no variable, and abs is a function symbol with binding arity $\langle 1\rangle$, which means that it takes one argument and the argument has one variable binding. We call a term like $\mathsf{abs}(x.\mathsf{app}(M[x], y))$ a *meta-term* meaning that a term involving metavariables. Likewise, using a function symbol λ with binding arity $\langle 1\rangle$ and an infix function symbol @ with arity $\langle 0, 0\rangle$, we can construct a meta-term $\lambda(x.M[x]@y)$, which is closer to the familiar notation (cf. Example 3.2).

2.1.4 Convention on α-equivalence

For a formal treatment of named variables modulo α-equivalence, we assume the method of de Bruijn levels (de Bruijn, 1972; Fiore et al., 1999; Lescanne and Rouyer-Degli, 1995) for the naming convention of variables. However, keeping de Bruijn level notation strictly is clumsy, especially in examples. Hence, we sometimes use the following convention: any term appearing in this paper hereafter is automatically normalised to a de Bruijn level α-normal form suitably. For example, $\lambda(x.\lambda(y.y@x))$ means $\lambda(1.\lambda(2.2@1)$, and $\lambda(x.M[x])$ to mean $\lambda(1.M[1])$, For metavariables, we use ordinary named notation and do not use de Bruijn levels.

2.2 The presheaf category $\mathbf{Set}^{\mathbb{F}}$

In modelling second-order abstract syntax, we use a framework of categorical algebra using presheaves. First we explain presheaves used in this paper. The category $\mathbb{F}$ has finite cardinals $n = \{1, \ldots, n\}$ (n is possibly 0) as objects, and all functions between them as arrows $n \to n'$.

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

The category $\mathbf{Set}^{\mathbb{F}}$ plays a central role in the algebraic models of syntax with variable binding (Fiore et al., 1999; Tanaka and Power, 2006). The objects of it are functors $\mathbb{F} \to \mathbf{Set}$ and the arrows are natural transformations between them. An object A of $\mathbf{Set}^{\mathbb{F}}$ is called a presheaf and is written as $A \in \mathbf{Set}^{\mathbb{F}}$.

A map between presheaves $A, B \in \mathbf{Set}^{\mathbb{F}}$ is a natural transformation $f : A \longrightarrow B$, that is, a family of functions of the form $f(n) : A(n) \longrightarrow B(n)$ parameterised by all $n \in \mathbb{N}$ that makes the naturality diagram commute

$$\begin{array}{ccc} m & A(m) & \xrightarrow{\quad f(m) \quad} & B(m) \\ \rho \downarrow & A(\rho) \downarrow & & \downarrow B(\rho) \\ n & A(n) & \xrightarrow[\quad f(n) \quad]{} & B(n) \end{array}$$

2.3 Algebraic model of abstract syntax and variable binding

In their seminal paper, Fiore et al. (1999) investigated algebraic models of abstract syntax involving variable binding. A typical example of such a syntax is the syntax for untyped λ-terms:

$$\frac{}{x_1, \ldots, x_n \vdash x_i} \qquad \frac{x_1, \ldots, x_n \vdash t \quad x_1, \ldots, x_n \vdash s}{x_1, \ldots, x_n \vdash t@s}$$

$$\frac{x_1, \ldots, x_n, x_{n+1} \vdash t}{x_1, \ldots, x_n \vdash \lambda(x_{n+1}.t)}$$

This is an abstract syntax generated by three constructors, that is, the variable former, the application @ and the abstraction λ. The point is that @ is a binary function symbol, but λ is not merely a unary function symbol. It also makes the variable x_{n+1} bound and decreases the context. This can be formulated by the function symbols with binding arities given in Section 2.1.3. In order to model the phenomenon of variable binding generally (not only for λ-terms), Fiore, Plotkin and Turi took the presheaf category $\mathbf{Set}^{\mathbb{F}}$ to be the universe of discourse. This is regarded as the category of object variables (regarded as contexts) by the method of de Bruijn index/level (i.e. natural numbers) and their renamings. A main result in Fiore et al. (1999) is that abstract syntax with variable binding is characterised as an initial algebra of suitable endofunctor generated by a signature (e.g. for λ-terms). Now a function symbol with binding arity, denoted by $f : \langle n_1, \ldots, n_l \rangle$, has l arguments and binds n_i variables in the i-th argument ($1 \leq i \leq l$). A *signature* Σ is a set Σ of function symbols with binding arities.

For example, for abstract syntax of λ-terms, we take a signature Σ consisting of $\lambda : \langle 1 \rangle$ and an infix function symbol $@ : \langle 0, 0 \rangle$, as described in Section 2.1.3. The corresponding *signature endofunctor* Σ_λ on $\mathbf{Set}^{\mathbb{F}}$ is $\Sigma_\lambda(A) = \mathrm{V} + A \times A + \delta A$ where each summand corresponds to the arity of function symbol where the context extension is defined by $(\delta A)(n) = A(n+1)$, and the presheaf $\mathrm{V} \in \mathbf{Set}^{\mathbb{F}}$ of variables is $\mathrm{V}(n) = \{1, \ldots, n\}$.

In general, given a signature Σ, we also denote by Σ the corresponding signature endofunctor[1] defined like Σ_λ (more precisely, see Definition 4.1). A Σ-*algebra* is a pair (A, α) consisting of a presheaf A and a map $\alpha : \Sigma A \to A$, called an *algebra structure* that provides the interpretations of constructors. An initial Σ_λ-algebra (Λ, in) can be constructed inductively as the presheaf Λ of all λ-terms modulo α-equivalence with free variable renaming. The initial algebra directly models the abstract syntax with binding, namely

$$\text{a judgment } n \vdash t \quad \text{is modelled as} \quad t \in \Lambda(n),$$

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

and renaming of free variables $\rho : n \to n'$ in a λ-term is modelled by the presheaf action $\Lambda(\rho) : \Lambda(n) \to \Lambda(n')$. Under the method of de Bruijn levels, n means the set of variables from 1 to n. This is a generic way of modelling abstract syntax with binding with respect to a signature functor Σ. However, the method is limited to modelling of *object-level* abstract syntax.

2.4 Free Σ-monoids: second-order abstract syntax with metavariables

Not only object-level abstract syntax, how can we deal with metavariables and the distinction of substitutions for object and metavariables in the algebraic model of syntax with binding? This problem was explored in Hamana (2004); Fiore (2008) and a clear answer has been obtained.

Given a signature functor Σ, a Σ-*monoid* (Fiore et al., 1999) is a Σ-algebra (A, α) with a monoid structure

$$\mathrm{V} \xrightarrow{\nu} A \xleftarrow{\mu} A \bullet A$$

that is compatible with the algebra structure in the monoidal category $(\mathbf{Set}^{\mathbb{F}}, \bullet, \mathrm{V})$[2]. The unit ν models the variable former, and the multiplication μ models *substitution for object variables*, where the monoidal product $\bullet$ gives the arity of substitution.

An important structure is a *free* Σ-*monoid* generated by an arbitrary presheaf Z, where generators Z is regarded as the *presheaf of metavariables*. A free Σ-monoid over $Z \in \mathbf{Set}^{\mathbb{F}}$, denoted by $\mathrm{M}_\Sigma Z$, is constructed inductively as an initial $(\mathrm{V} + \Sigma + Z \bullet -)$-algebra, which gives the language involving binding and *metavariables*. Terms of this language are expressed as the following BNF:

$$\begin{aligned} \mathrm{M}_\Sigma Z(n) \ni t &::= x \mid f(\, n{+}1.\ldots.n{+}i_1.t_1,\ \ldots,\ n{+}1.\ldots.n{+}i_m.t_m\,) \mid M[t_1, \ldots, t_m] \\ Z(m) \ni M & \end{aligned} \tag{2}$$

where $x \in \{1, \ldots, n\}$ and each $f \in \Sigma$ is a function symbol that binds i_j variables at the j-th argument. This characterisation shows that the functor $\mathrm{V} + \Sigma$ models syntax with binding (as in Section 2.1.1) and the functor $Z \bullet -$ models the syntactic construct of metavariables represented as a term

$$M[t_1, \ldots, t_m]$$

where $M \in Z(m)$ is a *metavariable* and the index m, called also *arity*, which denotes possible free variables $1, \ldots, l$ appearing in a term substituted for M. Terms $t_1, \ldots, t_m$ are replacements of the free variables $1, \ldots, m$ after instantiating M. For example, a substitution that replaces a metavariable M with a term 1@1 (where 1 is a free variable) is formulated as map θ from the presheaf Z of metavariables to $\mathrm{M}_\Sigma Z$, which maps the metavariable M as $\theta(M) = 1@1$. Then applying it to a term $\lambda(x.\, M[x] \mathbin{@} y)$

$$\theta^\sharp(\lambda(x.\, M[x] \mathbin{@} y)) = \lambda(x.\, (x@x) \mathbin{@} y) \tag{3}$$

is a substitution process. The freeness of $\mathrm{M}_\Sigma Z$ states that given θ, there uniquely exists the extension $\theta^\sharp$ to a Σ-monoid morphism that makes the following diagram commute:

$$\begin{array}{ccc} Z & \xrightarrow{\eta_Z} & \mathrm{M}_\Sigma Z \\ & {\scriptstyle\theta}\searrow & \downarrow{\scriptstyle\theta^\sharp} \\ & & A \end{array}$$

The general form of freeness is stated for an arbitrary Σ-monoid A. To consider the syntactic case, we take $A = \mathrm{M}_\Sigma Z$. Then, the notion of substitution of metavariables appears. This is syntactically understood as that θ is an *assignment for substitution* and $\theta^\sharp$ is the corresponding substitution of terms for metavariables. For (3), we put the metavariable $M \in Z(1)$.

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

Remark 2.1. Aczel first introduced the syntactic structure of meta-terms and substitution for abstract syntax with variable binding (Aczel, 1978). This formal language allowed him to consider a general framework of rewrite rules for calculi with variable binding. Hamana clarified the algebraic semantics of it using Σ-monoids (Hamana, 2005), extended it to simply typed case (Hamana, 2007). Fiore (2002) and Miculan and Scagnetto (2003) gave the simply typed and algebraic characterisation of abstract syntax with binding and substitution. Fiore and Hur (2010); Fiore and Mahmoud (2010) considered algebraic theories for second-order equational presentations. Hamana (2011) gave a polymorphic and algebraic characterisation of abstract syntax with variable binding. Fiore and Hamana (2013) formulated polymorphic algebraic theories.

Notation 2.2. We denote by $f(n+\overrightarrow{i_1}.t_1,\ldots,n+\overrightarrow{i_m}.t_m)$ to mean

$$f(\,n{+}1\cdots n{+}i_1.t_1,\ \ldots,\ n{+}1\cdots n{+}i_m.t_m\,).$$

We abbreviate the words left-hand side as 'lhs', right-hand side as 'rhs', first-order as 'FO' and higher-order as 'HO'.

A binary relation $>$ is *well-founded* if there is no infinite decreasing sequences $a_1 > a_2 > \cdots$.

3. Second-Order CSs

In this section, we introduce the framework of mono-sorted second-order CSs as a computational counterpart of second-order algebraic theories (Fiore and Hur, 2010; Fiore and Mahmoud, 2010).

We formally define *second-order CSs* as follows.

(i) A function symbol with *(binding) arity*, denoted by $f:\langle n_1,\ldots,n_l\rangle$, has l arguments and binds n_i variables in the i-th argument ($1\le i\le l$). A *signature* Σ is a set Σ of function symbols with binding arities.

(ii) Let Z be an $\mathbb{N}$-*indexed set of metavariables* parameterised by arities, where each $Z(n)$ is a set of n-ary metavariables. We may denote by $M^{(n)}$ an n-ary metavariable $M\in Z(n)$. We may simply use the set-notation to describe an $\mathbb{N}$-indexed set Z. For example, writing $Z=\{M^{(1)},N^{(2)}\}$, we mean $Z(1)=\{M\}$, $Z(2)=\{N\}$, and $Z(i)=\varnothing$ for all $i\neq 1,2$.

(iii) The *raw meta-terms* are given by the grammar:

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

The last form $M[t_1,\ldots,t_n]$ is called a *meta-application*.

(iv) A *judgment* is of the form

$$Z \triangleright n \vdash t$$

where Z is an $\mathbb{N}$-indexed set of metavariables. When Z is empty, we may write $\triangleright n\vdash t$ or simply write $n\vdash t$.

(v) A *well-formed meta-term* t is a raw meta-term such that a judgment $Z\triangleright n\vdash t$ is derived by the rules in Figure 1 for some Z, n. By these rules, a meta-term always follows the method of de Bruijn levels. Hereafter, we only deal with well-formed meta-terms.

(vi) A *term* t is a raw meta-term such that a judgment $\triangleright n\vdash t$ is derived by the rules in Figure 1 for some n. Namely, a term t is a meta-term without metavariables.

(vii) An *assignment* θ is a mapping that assigns to an n-ary metavariable M a meta-term s as $\theta: M\longmapsto s$ where $Z\triangleright n+k\vdash s$ and for some $k\in\mathbb{N}$. The same k is used for the present θ and all metavariables M in Z, considered as additional free variables $n+1,\ldots,n+k$ appearing in s.

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

$$\frac{1 \le i \le n}{Z \rhd n \vdash\ i} \qquad \frac{M \in Z(m) \quad Z \rhd n \vdash\ t_i \quad (1 \le i \le m)}{Z \rhd n \vdash\ M[t_1, \ldots, t_m]}$$

$$\frac{\begin{array}{c} f : \langle i_1, \ldots, i_m \rangle \in \Sigma \\ Z \rhd n{+}i_j \vdash\ t_j \quad (1 \le j \le m) \end{array}}{Z \rhd n \vdash\ f(\ n{+}1 \cdots n{+}i_1.t_1,\ \ldots,\ n{+}1 \cdots n{+}i_m.t_m\ )}$$

Figure 1. Typing rules of meta-terms.

$$(\text{Rule})\frac{\begin{array}{c} \rhd n + i_j \vdash\ s_j \quad (1 \le j \le k) \quad \theta = [M_1 \mapsto s_1, \ldots, M_m \mapsto s_k] \\ (M_1^{(i_1)}, \ldots, M_k^{(i_k)} \rhd 0 \vdash\ \ell \Rightarrow r) \in \mathcal{C} \end{array}}{\rhd n \vdash\ \theta^\sharp(\ell) \to_{\mathcal{C}} \theta^\sharp(r)}$$

$$(\text{Fun})\frac{f : \langle i_1, \ldots, i_m \rangle \in \Sigma \quad \rhd n + i_j \vdash\ t_j \to_{\mathcal{C}} t'_j \quad (\text{some single } j \text{ s.t. } 1 \le j \le k)}{\rhd n \vdash\ f(\ \ldots, n{+}1 \cdots n{+}i_j.t_j, \ldots) \to_{\mathcal{C}} f(\ \ldots, n{+}1 \cdots n{+}i_j.t'_j, \ldots)}$$

Figure 2. Second-order rewriting (one-step).

We may denote an assignment θ by the notation $\theta = [M_1 \mapsto a_1, \ldots, M_l \mapsto a_l]$. An assignment is extended to a *substitution* $\theta^\sharp$ *for metavariables*. By the notation

$$\theta^\sharp(t)$$

we mean that a meta-term obtained by replacing every meta-application $M[t_1, \ldots, t_n]$ in t with a meta-term s whose free variables $1, \ldots, n$ are replaced with $t_1, \ldots, t_n$. In this substitution process, to avoid unintended capture of free variables by a binder, variables may be suitably shifted according to the method of de Bruijn levels.

(viii) For meta-terms $Z \rhd 0 \vdash\ \ell$ and $Z \rhd 0 \vdash\ r$ using a signature Σ, a *rewrite rule* is of the form $Z \rhd 0 \vdash\ \ell \Rightarrow r$ satisfying:

(a) ℓ is a term of the form $f(t_1, \ldots, t_n)$, including the case $n = 0$.
(b) all metavariables in r occur in ℓ.

These are ordinary syntactic conditions of rewrite systems to make them computationally meaningful (Mayr and Nipkow, 1998). Without them, a rule like $M \Rightarrow a$ or $a \Rightarrow M$ is allowed, which collapses any term into a, or a generates any term. Since it admits a looping rewrite $a \to_{\mathcal{C}} a$, we impose these conditions. Note also that ℓ and r are meta-terms without free variables (because they are in the context 0) but may have free *meta*variables taken from Z.

(ix) A CS is a tuple $(\Sigma, \mathcal{C}, Z)$ of a signature, a non-empty set $\mathcal{C}$ of rewrite rules consisting of Σ-meta-terms and an $\mathbb{N}$-indexed set Z of all metavariables occurring in $\mathcal{C}$ satisfying

$$Z \supseteq \bigcup_{(Z_i \rhd 0 \vdash\ \ell \Rightarrow r) \in \mathcal{C}} Z_i.$$

We assume that the rules use disjoint sets Z_i of metavariables. If not, we implicitly use suitable renamed variants of rules. We may simply write a CS by $\mathcal{C}$ if Σ and Z are inferable.

(x) A one-step *rewrite* $\rhd n \vdash\ s \to_{\mathcal{C}} t$ is a judgment generated from a given CS $\mathcal{C}$ by using the inference rules in Figure 2. (Rule) instantiates a rewrite rule by replacing metavariables with terms. Hence, a rewrite step is defined on terms, not containing any metavariables. We often simply write a rewrite $s \to_{\mathcal{C}} t$ and regard $\to_{\mathcal{C}}$ as a binary relation on terms.

Remark 3.1. In the previous formulations (Hamana, 2017, 2019; Hamana et al., 2020), the lhss of second-order rewrite rules are restricted to Miller's *higher-order patterns* at second order

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

(Miller, 1991), or their slight extension of *deterministic second-order patterns* (Libal and Miller, 2016; Yokoyama et al., 2003, 2004). Second-order patterns are meta-terms in which every occurrence of a meta-application is of the form $M[x_1, \ldots, x_n]$, where $x_1, \ldots, x_n$ are distinct bound variables. Second-order patterns are useful in view of algorithmic account of CSs because there exists the most general unifier for a solvable unification problem, and an efficient unification algorithm is known (Miller, 1991).

But in view of algebraic semantics, the restriction is unnecessary, hence in this paper we consider a more general syntactic form of rewrite rules that allows non-pattern lhss. Such rules have also appeared in the literature, which we will see in Example 5.11 and Section 7.5.

Example 3.2. (Untyped λ-calculus) The signature Σ_λ for untyped λ-calculus is given by

$$\lambda : \langle 1 \rangle, \qquad @ : \langle 0, 0 \rangle$$

The CS $\mathcal{C}$ for untyped λ-calculus is given by the β and η rules:

$$\begin{array}{llllll} M^{(1)}, N^{(0)} & \triangleright\ 0 & \vdash & \lambda(x.M[x])@N & \Rightarrow & M[N] \\ M^{(0)} & \triangleright\ 0 & \vdash & \lambda(x.M@x) & \Rightarrow & M \end{array}$$

Using $\mathcal{C}$, we have a one-step rewrite

$$\triangleright y \vdash \ \lambda(x.x@y)@(\lambda(z.z)) \to_{\mathcal{C}} \lambda(z.z)@y$$

using $\theta : M \mapsto x@y,\ N \mapsto \lambda(z.z)$. Formally, in de Bruijn levels, this is derived by

$$\dfrac{\begin{array}{l} \triangleright 2 \vdash \ 2@1 \quad \triangleright\ 1 \vdash \ \lambda(2.2) \quad \theta = [M^{(1)} \mapsto 2@1,\ N^{(0)} \mapsto \lambda(2.2)] \\ (M^{(1)}, N^{(0)} \triangleright 0 \vdash \lambda(1.M[1])@N \Rightarrow M[N]) \in \mathcal{C} \end{array}}{\triangleright 1 \vdash \lambda(2.2@1)@(\lambda 2.2) \to_{\mathcal{C}} \lambda(2.2)@1}\ (\text{Rule})$$

Example 3.3. (CPS translation) The format of CS is similar to a meta-language for expressing formal systems in computer science and logic. Here we consider the following CS $\mathcal{S}$ for a call-by-value CPS translation (Danvy and Rose, 1998).

Assume the metavariables $Z = \{V^{(0)}, E^{(1)}, (E_0)^{(0)}, (E_1)^{(0)}\}$ and the signature Σ consisting of the function symbols

$$\lambda : \langle 1 \rangle, \qquad \overline{\lambda} : \langle 1 \rangle, \qquad (-\ -) : \langle 0, 0 \rangle, \qquad (-\ \overline{\ }\ -) : \langle 0, 0 \rangle \qquad \mathsf{CPS} : \langle 0 \rangle, \qquad (\!|-|\!) : \langle 0 \rangle.$$

There are two kinds of λ-terms, ordinary (consisting of λ and $(-\ -)$) and over-lined ones (consisting of $\overline{\lambda}$ and $(-\ \overline{\ }\ -)$), and the CPS translates ordinary λ-terms to over-lined λ-terms using the auxiliary translation $(\!|-|\!)$. We write the CS $\mathcal{S}$ of CPS translation in two ways: the left column is written in the usual named notation, and the right column is written in de Bruijn level notation, which is the format we use in this paper. Here we omit metavariable and variable contexts in the rules.

$\mathsf{CPS}(E_0) \Rightarrow \lambda k.(\!|E_0|\!)\ \overline{\ }\ (\overline{\lambda} m.km)$	$\mathsf{CPS}(E_0) \Rightarrow \lambda 1.(\!|E_0|\!)\ \overline{\ }\ (\overline{\lambda} 2.12)$
$(\!|V|\!) \Rightarrow \overline{\lambda} k.k\ \overline{\ }\ V$	$(\!|V|\!) \Rightarrow \overline{\lambda} 1.1\ \overline{\ }\ V$
$(\!|\lambda x.E[x]|\!) \Rightarrow \overline{\lambda} k.k\ \overline{\ }\ (\lambda x.\lambda k.(\!|E[x]|\!)\ \overline{\ }\ (\overline{\lambda} m.km))$	$(\!|\lambda 1.E[1]|\!) \Rightarrow \overline{\lambda} 1.1\ \overline{\ }\ (\lambda 2.\lambda 3.(\!|E[2]|\!)\ \overline{\ }\ (\overline{\lambda} 4.34))$
$(\!|E_0E_1|\!) \Rightarrow \overline{\lambda} k.(\!|E_0|\!)\ \overline{\ }\ (\overline{\lambda} m.(\!|E_1|\!)\ \overline{\ }\ (\overline{\lambda} n.mn(\lambda a.k\ \overline{\ }\ a)))$	$(\!|E_0E_1|\!) \Rightarrow \overline{\lambda} 1.(\!|E_0|\!)\ \overline{\ }\ (\overline{\lambda} 2.(\!|E_1|\!)\ \overline{\ }\ (\overline{\lambda} 3.23(\lambda 4.1\ \overline{\ }\ 4)))$

A point is that de Bruijn level version is obtained by just renaming variable names with numbers according to their de Bruijn levels. Notice that this differs from the more well-known method of de Bruijn *indices*. Meta-terms in de Bruijn levels can be regarded as 'normal forms' of α-equivalent meta-terms (e.g. $\overline{\lambda} k.k\ \overline{\ }\ V =_\alpha \overline{\lambda} 1.1\ \overline{\ }\ V$).

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

One of the important properties of rewriting systems is *termination*, which is also called *strong normalisation*, meaning that any computation path is finite.

Definition 3.4. We call a CS $(\Sigma, \mathcal{C}, Z)$ *terminating* if the rewrite relation $\rightarrow_{\mathcal{C}}$ on all terms is well-founded.

We will give a complete algebraic characterisation of terminating CSs in Section 5.

4. Algebraic Semantics of Meta-terms

In this section, we give a semantics of the syntax of CSs. We consider second-order abstract syntax in the framework of algebras in the presheaf category $\mathbf{Set}^{\mathbb{F}}$ reviewed in Section 2. Terms and meta-terms have algebraic properties of initial and free Σ-monoids, respectively, which are suitable to use them as the syntax of CSs.

4.1 Algebra of meta-terms

We define the functor $\delta : \mathbf{Set}^{\mathbb{F}} \rightarrow \mathbf{Set}^{\mathbb{F}}$ for context extension by

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

for $A \in \mathbf{Set}^{\mathbb{F}}$, $n \in \mathbb{F}$, $\rho : m \rightarrow n$ in $\mathbb{F}$, and a presheaf $\mathrm{V} \in \mathbf{Set}^{\mathbb{F}}$ called *the presheaf of variables* by

$$\mathrm{V}(n) = \{1, \ldots, n\}; \quad \mathrm{V}(\rho) = \rho.$$

The meaning of this definition is that each component $\mathrm{V}(n) = \{1, \ldots, n\}$ gives the set of (de Bruijn) variables from 1 to n, and $\mathrm{V}(\rho)$ gives a renaming between them.

Definition 4.1. To a signature Σ, we associate the *signature functor* $\Sigma : \mathbf{Set}^{\mathbb{F}} \rightarrow \mathbf{Set}^{\mathbb{F}}$ given by

$$\Sigma A \stackrel{def}{=} \coprod_{f:\langle n_1,\ldots,n_l\rangle \in \Sigma} \prod_{1 \leq i \leq l} \delta^{n_i} A.$$

A Σ-*algebra* is an algebra of the functor Σ. Namely, a Σ-algebra is a pair (A, α) consisting of a presheaf $A \in \mathbf{Set}^{\mathbb{F}}$, called a *carrier*, and a map $\alpha = [f^A]_{f \in \Sigma} : \Sigma A \longrightarrow A$ called the *algebra structure*, where f^A is a map of $\mathbf{Set}^{\mathbb{F}}$

$$f^A : \delta^{n_1} A \times \ldots \times \delta^{n_l} A \longrightarrow A$$

called an *operation*, defined for each function symbol $f : \langle n_1, \ldots, n_l \rangle \in \Sigma$, where $[\,]$ denotes the copair of coproducts.

Remark 4.2. For each $n \in \mathbb{F}$, the component of an operation f^A at n is a function

$$f^A_n : A(n+n_1) \times \cdots \times A(n+n_l) \longrightarrow A(n)$$

Definition 4.3. A $(\mathrm{V}+\Sigma)$-*algebra* $(A, [\nu, \alpha])$ is an algebra of the functor $(\mathrm{V}+\Sigma) : \mathbf{Set}^{\mathbb{F}} \rightarrow \mathbf{Set}^{\mathbb{F}}$. Namely, it consists of a Σ-algebra (A, α) and a map of $\mathbf{Set}^{\mathbb{F}}$

$$\nu : \mathrm{V} \rightarrow A.$$

called a *unit*.

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

Example 4.4. For the signature Σ_λ of the λ-calculus given in Example 3.2, the presheaf Λ of all λ-terms is defined by

$$\Lambda(n) = \{t \mid \triangleright\, n \vdash t\}$$

and for $\rho : m \to n$ in $\mathbb{F}$, $\Lambda(\rho) : \Lambda(m) \to \Lambda(n)$ is a renaming of λ-term by ρ, where each free variable $i \in \{1, \ldots, m\}$ is replaced with $\rho(i)$. It forms a $(\mathrm{V}+\Sigma_\lambda)$-algebra by giving the operations

$$\begin{array}{llllllll} \nu^\Lambda : \mathrm{V} & \to & \Lambda & @^\Lambda : \Lambda\times\Lambda & \to & \Lambda & \lambda^\Lambda : \delta\Lambda & \to & \Lambda \\ \nu_n(x) & = & x, & @^\Lambda_n(s,t) & = & s@t, & \lambda^\Lambda_n(t) & = & \lambda(n+1.t). \end{array}$$

These operations correspond to the constructors of variables, applications and abstractions.

Definition 4.5. A Σ-*monoid* (A, α, ν, μ) is a Σ-algebra (A, α) equipped with a unit $\nu : \mathrm{V} \to A$ and a family of functions, called a *multiplication*,

$$\mu^{(n)}_m : A(n)\times A(m)^n \longrightarrow A(m)$$

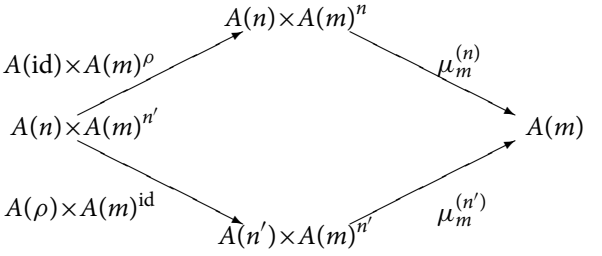
which is natural in $m \in \mathbb{F}$ and extra-natural in $n \in \mathbb{F}$, and satisfies the following axioms:

$$\begin{array}{lll} \text{(i)} & \mu^{(n)}_m(\nu_n(i); \overrightarrow{b}) & = b_i \quad (1 \le i \le n) \\ \text{(ii)} & \mu^{(m)}_m(a; \nu_m(1), \ldots, \nu_m(m)) & = a \\ \text{(iii)} & \mu^{(n)}_m(a; \mu^{(l)}_m(b_1, \overrightarrow{c}), \ldots, \mu^{(l)}_m(b_l, \overrightarrow{c})) & = \mu^{(l)}_m(\mu^{(n)}_l(a; \overrightarrow{b}); \overrightarrow{c}) \\ \text{(iv)} & \mu^{(n)}_m(f^A_n(a_1, \ldots, a_l); \overrightarrow{b}) & = f^A_m(\mu^{(n+k_1)}_{m+k_1}(a_1; \mathsf{up}^{m+k_1}_m(\overrightarrow{b}), (\nu_{m+k_1}(m+i))_{i=1,\ldots,k_1}), \ldots, \\ & & \quad \mu^{(n+k_l)}_{m+k_l}(a_l; \mathsf{up}^{m+k_l}_m(\overrightarrow{b}), (\nu_{m+k_l}(m+i))_{i=1,\ldots,k_l})) \end{array}$$

where $f : \langle k_1, \ldots, k_l\rangle \in \Sigma$, $\overrightarrow{b} = b_1, \ldots, b_n$ and $\mathsf{up}^n_m \stackrel{def}{=} A(\mathrm{up}^n_m)$, where $m \le n$ and a map $\mathrm{up}^n_m : m \to n$ in $\mathbb{F}$ is an injection defined by $j \mapsto j$. Here, an element in the product $A(n)\times A(m)^n$ is denoted by $(a; b_1, \ldots, b_n)$, or simply $(a; \overrightarrow{b})$.

A Σ-monoid models term and substitution structures.

Remark 4.6. The above definition is almost the same as the definition of abstract clone (Taylor, 1993). The naturality condition of μ expresses that μ is parameterised with respect to renaming parameters m and n. The statement '$\mu^{(n)}_m$ is extra-natural in n' means that for any $\rho : n \to n'$ in $\mathbb{F}$, the diagram



commutes.

Another way to define Σ-monoid using a monoidal structure on $\mathbf{Set}^{\mathbb{F}}$ is given as follows, which is the original definition (Fiore et al., 1999) of Σ-monoid.

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

Definition 4.7. $(\mathbf{Set}^{\mathbb{F}}, \bullet, \mathrm{V})$ forms a monoidal category (Mac Lane, 1971), where the 'substitution' monoidal product $\bullet$ is defined as follows. For presheaves A and B,

$$(A \bullet B)(n) \stackrel{def}{=} \left(\coprod_{m \in \mathbb{N}} A(m) \times B(n)^m \right) / \sim$$

where $\sim$ is the equivalence relation generated by $(t; u_{\rho 1}, \ldots, u_{\rho m}) \sim (A(\rho)(t); u_1, \ldots, u_l)$ for $\rho : m \to l \in \mathbb{F}$.

Let Σ be a signature functor. It has a canonical strength $st : \Sigma(A) \bullet A \to \Sigma(A \bullet A)$ (Tanaka and Power, 2006, Theorem 6.3) (Fiore, 2008, Corollary 7). A Σ-*monoid* $A = (A, \alpha, \nu, \mu)$ consists of a monoid object $(A, \nu : \mathrm{V} \to A, \mu : A \bullet A \to A)$ in the monoidal category $(\mathbf{Set}^{\mathbb{F}}, \bullet, \mathrm{V})$ with a Σ-algebra $\alpha : \Sigma A \to A$ such that the following commutes:

$$\begin{array}{ccccc} \Sigma(A) \bullet A & \xrightarrow{st} & \Sigma(A \bullet A) & \xrightarrow{\Sigma\mu} & \Sigma A \\ {\scriptstyle \alpha \bullet A}\downarrow & & & & \downarrow{\scriptstyle \alpha} \\ A \bullet A & & \xrightarrow[\mu]{} & & A \end{array}$$

Remark 4.8. The axioms (i)–(iii) in Definition 4.5 correspond to the monoid laws (two side unit laws and the associativity law), and the axiom (iv) is called the Σ-*monoid law* stating that μ commutes with each function symbol, which corresponds to the diagram in Definition 4.7. The axiom (iv) is a necessary property of substitution on a term-like structure with variable binding.

In Definition 4.5, a unit $\nu : \mathrm{V} \to A$ can be understood as an (interpretation) map of *object variables*, which embeds object variables to A. A multiplication μ can be understood as a map of (interpreted) substitution operation in A. Then, the axioms have the following intuitive reading.

(i) This axiom means replacing a variable i with the i-th element b_i in $\overrightarrow{b}$.

(ii) This axiom says that substituting the (interpretation of) variables $\nu(1), \ldots, \nu(n)$ in A for variables $1, \ldots, n$ does not affect the element.

(iii) This axiom is the associativity of substitutions, that is, a version of substitution lemma.

(iv) This Σ-monoid axiom says that the (interpretation of) substitution μ is pushed into the (interpretation of) term structure.

The two ways of defining Σ-monoids are equivalent. In a termination proof of CS, we often need to give a concrete example of Σ-monoids. For it, Definition 4.5 of Σ-monoids is more convenient than the original Definition 4.7. In theory, Definition 4.7 is more convenient. For instance, we will use it to construct a free Σ-monoid in Theorem 4.13. Hereafter, we use both descriptions of Σ-monoids.

Definition 4.9. A *homomorphism* h between Σ-algebras $(A, [f^A]_{f \in \Sigma})$ to $(B, [f^B]_{f \in \Sigma})$ is a map $h : A \longrightarrow B$ of $\mathbf{Set}^{\mathbb{F}}$ such that

$$\begin{array}{ccc} \Sigma A & \xrightarrow{[f^A]_{f \in \Sigma}} & A \\ {\scriptstyle \Sigma h}\downarrow & & \downarrow{\scriptstyle h} \\ \Sigma B & \xrightarrow[[f^B]_{f \in \Sigma}]{} & B \end{array}$$

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

Concretely,

$$h_n(f_n^A(a_1,\ldots,a_l)) = f_n^B(h_{n+i_1}(a_1),\ldots,h_{n+i_l}(a_l))$$

for all $f : \langle i_1,\ldots,i_l\rangle \in \Sigma,\ n \in \mathbb{N},\ a_1,\ldots,a_l \in A(n)$. This defines the category Σ-**alg** of Σ-algebras.

A *morphism of* Σ*-monoids* $h : (A, \nu^A, \mu^A) \longrightarrow (B, \nu^B, \mu^B)$ is a Σ-algebra homomorphism that is also a monoid morphism, that is, it satisfies the condition of homomorphism and the following

$$h_n(\nu_n^A(i)) = \nu_n^B(i)$$
$$h_n(\mu_n^{A(m)}(a; \overrightarrow{b})) = \mu_n^{B(m)}(h_m(a); h_n(b_1),\ldots,h_n(b_m)).$$

This defines the category Σ-**Mon** of Σ-monoids.

Example 4.10. Using the signature Σ_λ in Example 4.4, the presheaf Λ of λ-terms forms a Σ_λ-monoid (Λ, ν, μ) as follows. Since Λ is a $(\mathrm{V} + \Sigma_\lambda)$-algebra, it has the unit ν (given in Example 4.4). The multiplication μ is defined by

$$\begin{array}{lcll} \mu_n(i; \overrightarrow{t}) & = & t_i & (i \in n) \\ \mu_n(s_1@s_2; \overrightarrow{t}) & = & \mu_n(s_1; \overrightarrow{t}) \mathbin{@} \mu_n(s_2; \overrightarrow{t}) & \\ \mu_n(\lambda(n+1.\, s); \overrightarrow{t}) & = & \lambda(n+1.\, \mu_{n+1}(s; \overrightarrow{t}, n+1)). & \end{array}$$

This μ is a substitution of a λ-term. In the ordinary notation,

$$\mu_n(t, \overrightarrow{s}) = t[1 := s_1,\ldots,n := s_n],$$

where $1,\ldots,n$ are variables in de Bruijn level notation. The axioms of Σ-monoid are satisfied, because μ is the substitution operation. This is an instructive concrete example to understand the meaning of axioms. In this case, the presheaf Λ is given syntactically, and all operations on Λ and μ are also given syntactically, that is, the constructors of λ-terms and the substitution operation.

The presheaf $\mathrm{T}_\Sigma\mathrm{V} \in \mathbf{Set}^{\mathbb{F}}$ of all terms is defined by

$$\mathrm{T}_\Sigma\mathrm{V}(n) = \{t \mid \triangleright\ n \vdash t\}$$
$$\mathrm{T}_\Sigma\mathrm{V}(\rho)(i) = \rho(i) \quad (1 \le i \le m)$$
$$\mathrm{T}_\Sigma\mathrm{V}(\rho)(f(m+\overrightarrow{i_1}.t_1,\ldots,m+\overrightarrow{i_l}.t_l)) = f(m+\overrightarrow{i_1}.\mathrm{T}_\Sigma\mathrm{V}(\rho+\mathrm{id}_{i_1})(t_1),\ldots,m+\overrightarrow{i_l}.\mathrm{T}_\Sigma\mathrm{V}(\rho+\mathrm{id}_{i_l})(t_l))$$

for $f : \langle i_1,\ldots,i_l\rangle \in \Sigma,\ \rho : m \to n$ in $\mathbb{F}$, which gives renaming of variables in a term.

Theorem 4.11. The presheaf $\mathrm{T}_\Sigma\mathrm{V}$ of terms forms an initial $(\mathrm{V}+\Sigma)$-algebra, i.e. an initial object in the category $(\mathrm{V}+\Sigma)$-**alg**.

Proof. An initial $(\mathrm{V}+\Sigma)$-algebra is constructed by the colimit of the ω-chain $0 \to (\mathrm{V}+\Sigma)\,0 \to (\mathrm{V}+\Sigma)^2 0 \to \cdots$ (Adamek, 1974). These construction steps correspond to derivations of terms by term forming rules; hence, their union $\mathrm{T}_\Sigma\mathrm{V}$ gives the object part of the colimit. The arrow part of the colimit is associated with substitution. The associated algebra structure is given as follows: the map $\nu : \mathrm{V} \longrightarrow \mathrm{T}_\Sigma\mathrm{V}$ is given by

$$\nu_n : \mathrm{V}(n) \longrightarrow \mathrm{T}_\Sigma\mathrm{V}(n),$$
$$i \longmapsto i.$$

For every $f \in \Sigma$ with biding arity $\langle i_1,\ldots,i_l\rangle$, the map $f^{\mathrm{T}_\Sigma\mathrm{V}} : \delta^{i_1}\mathrm{T}_\Sigma\mathrm{V} \times \cdots \times \delta^{i_l}\mathrm{T}_\Sigma\mathrm{V} \longrightarrow \mathrm{T}_\Sigma\mathrm{V}$ in $\mathbf{Set}^{\mathbb{F}}$ is given by

$$(t_1,\ldots,t_l) \longmapsto f(n+\overrightarrow{i_1}.t_1,\ldots,n+\overrightarrow{i_l}.t_l).$$

□

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

In Fiore et al. (1999, Theorem 2.1), the 'syntactic algebra' is constructed as an initial $(\mathrm{V}+\Sigma)$-algebra, which is nothing but this $(\mathrm{V}+\Sigma)$-algebra $(\mathrm{T}_\Sigma \mathrm{V}, [\nu, [f^{\mathrm{T}_\Sigma}]_{f\in\Sigma}])$.

Proposition 4.12. There is an adjunction

$$\mathbf{Set}^{\mathbb{N}} \underset{|-|}{\overset{\underline{(-)}}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathbf{Set}^{\mathbb{F}}$$

where the functor $|-| = - \circ J$ using the inclusion $J : \mathbb{N} \to \mathbb{F}$ gives the underlying indexed set of a presheaf. Its left adjoint $\underline{(-)}$ is the left Kan extension along J, calculated (Hamana, 2004) as $\underline{Z}(n) = \coprod_{k\in\mathbb{N}} \mathbb{F}(k, n)\times Z(k)$ for $Z \in \mathbf{Set}^{\mathbb{N}}$.

Therefore, we have a way to construct a presheaf $\underline{Z} \in \mathbf{Set}^{\mathbb{F}}$ from a given $\mathbb{N}$-indexed set Z of metavariables. Using the adjointness, a map $\phi : Z \to |A|$ in $\mathbf{Set}^{\mathbb{N}}$ for $A \in \mathbf{Set}^{\mathbb{F}}$ induces a map $\hat{\phi} : \underline{Z} \to A$ in $\mathbf{Set}^{\mathbb{F}}$. We also regarded this as a way to construct a map of presheaves from an assignment ϕ for metavariables in Section 4.2.

Let Z be an arbitrary $\mathbb{N}$-indexed set of metavariables. Given a signature Σ, the *presheaf* $\mathrm{M}_\Sigma\underline{Z}$ *of meta-terms* over Z is defined by

$$\mathrm{M}_\Sigma\underline{Z}(n) = \{t \mid Z \triangleright n \vdash t\}$$

and for $\rho : m \to n$ in $\mathbb{F}$ defined by

$$\begin{aligned}
\mathrm{M}_\Sigma\underline{Z}(\rho)(i) &= \rho(i)\\
\mathrm{M}_\Sigma\underline{Z}(\rho)(f(n+\overrightarrow{i_1}.t_1, \ldots, n+\overrightarrow{i_l}.t_l)) &= f(n+\overrightarrow{i_1}.\mathrm{M}_\Sigma\underline{Z}(\rho)(t_1), \ldots, n+\overrightarrow{i_l}.\mathrm{M}_\Sigma\underline{Z}(\rho)(t_l))\\
\mathrm{M}_\Sigma\underline{Z}(\rho)(M[t_1, \ldots, t_l]) &= M[\mathrm{M}_\Sigma\underline{Z}(\rho)(t_1), \ldots, \mathrm{M}_\Sigma\underline{Z}(\rho)(t_l)].
\end{aligned}$$

Theorem 4.13. The presheaf $\mathrm{M}_\Sigma\underline{Z}$ of meta-terms forms a free Σ-monoid over $\underline{Z}$.

Proof. Due to Hamana (2004). For $Z \in \mathbf{Set}^{\mathbb{N}}$, a free Σ-monoid $(\mathrm{M}_\Sigma\underline{Z}, [f_{\mathrm{M}_\Sigma}]_{f\in\Sigma}, \nu, \mu)$ over $\underline{Z}$ is constructed as an initial $(\mathrm{V} + \Sigma + \underline{Z} \bullet -)$-algebra $(\mathrm{M}_\Sigma\underline{Z}, \nu, [f_{\mathrm{M}_\Sigma}]_{f\in\Sigma}, \sigma)$. For every $f \in \Sigma$ with binding arity $\langle i_1, \ldots, i_l\rangle$, the map $f^{\mathrm{M}_\Sigma\underline{Z}} : \delta^{i_1}\mathrm{M}_\Sigma\underline{Z}\times\cdots\times\delta^{i_l}\mathrm{M}_\Sigma\underline{Z} \longrightarrow \mathrm{M}_\Sigma\underline{Z}$ in $\mathbf{Set}^{\mathbb{F}}$ is given by

$$(t_1, \ldots, t_l) \longmapsto f(n+\overrightarrow{i_1}.t_1, \ldots, n+\overrightarrow{i_l}.t_l).$$

The unit ν is given by $\nu_n : i \mapsto i$. The map $\sigma : \underline{Z} \bullet \mathrm{M}_\Sigma\underline{Z} \longrightarrow \mathrm{M}_\Sigma\underline{Z}$ is given by

$$((\xi, M); t_1, \ldots, t_m) \longmapsto M[t_{\xi(1)}, \ldots, t_{\xi(m)}]$$

The map $\eta_{\underline{Z}} : \underline{Z} \longrightarrow \mathrm{M}_\Sigma\underline{Z}$ is given by

$$\eta_{\underline{Z},m} : (\xi, Z) \mapsto Z[\xi(1), \ldots, \xi(m)].$$

The multiplication μ is given as the substitution for variables, which makes the diagram

$$\begin{array}{ccc}
\underline{Z} \bullet \mathrm{M}_\Sigma\underline{Z} & \xrightarrow{\eta_{\underline{Z}} \bullet \mathrm{id}} & \mathrm{M}_\Sigma\underline{Z} \bullet \mathrm{M}_\Sigma\underline{Z}\\
& \searrow^{\sigma} & \downarrow{\scriptstyle \mu}\\
& & \mathrm{M}_\Sigma\underline{Z}
\end{array}$$

commute. □

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

Corollary 4.14. The presheaf $\mathrm{T}_\Sigma \mathrm{V}$ forms an initial Σ-monoid, i.e. an initial object in the category Σ-**Mon**.

Proof. Taking $Z = 0$ as the empty presheaf, we have $\mathrm{T}_\Sigma \mathrm{V} = \mathrm{M}_\Sigma 0$. □

4.2 Algebraic characterisation of substitution for metavariables

We have shown that the meta-terms form a free Σ-monoid. Formally, given $Z \in \mathbf{Set}^{\mathbb{N}}$, $\mathrm{M}_\Sigma \underline{Z}$ is a free Σ-monoid over $\underline{Z} \in \mathbf{Set}^{\mathbb{F}}$. Namely, for an arbitrary Σ-monoid $(A, \alpha, \nu^A, \mu^A)$ and $\phi : Z \to |A|$, there exists a unique Σ-monoid morphism $\phi^\sharp$ that makes the diagram

$$\begin{array}{ccc} \underline{Z} & \xrightarrow{\eta_{\underline{Z}}} & \mathrm{M}_\Sigma \underline{Z} \\ & \searrow^{\hat{\phi}} & \downarrow{\scriptstyle \phi^\sharp} \\ & & A \end{array}$$

commute.

We have seen that an $\mathbb{N}$-indexed function $\phi : Z \to |A|$ induces a map $\hat{\phi} : \underline{Z} \to A$ of $\mathbf{Set}^{\mathbb{F}}$. It is also extended to a Σ-monoid morphism $\phi^\sharp : \mathrm{M}_\Sigma \underline{Z} \to A$ defined by

$$\begin{aligned} \phi^\sharp_n(i) &= \nu^A_n(i) \\ \phi^\sharp_n(f(n + \overrightarrow{i_1}.t_1, \ldots, n + \overrightarrow{i_l}.t_l)) &= f^A_n(\phi^\sharp_{n+i_1}(t_1), \ldots, \phi^\sharp_{n+i_l}(t_l)) \\ \phi^\sharp_n(M[\overrightarrow{t}]) &= \mu^A_n(\phi_m(M); \phi^\sharp_n(t_1), \ldots, \phi^\sharp_n(t_m)) \quad \text{for } M \in Z(m). \end{aligned} \tag{4}$$

This definition gives the meaning of a meta-term in a Σ-monoid A by structural recursion. Note that it is not the standard structural recursion because the indices of ϕ vary in the recursive calls. The meaning of a variable is obtained using ν^A, which is the interpretation of variable former, a function symbol f is interpreted as f^A and a meta-application is interpreted as follows. Firstly, the value of a metavariable M is obtained using the map ϕ, and secondly, μ^A replaces semantically free m variables $1, \ldots, m$ with the meanings of $t_1, \ldots, t_m$ obtained by applying $\phi^\sharp$.

The extension captures the notion of syntactic substitutions for metavariables, where a map ϕ is regarded as an assignment of values for metavariables. We formulate substitutions along this idea.

Definition 4.15. Let Z be an $\mathbb{N}$-indexed set of metavariables, $A \in \mathbf{Set}^{\mathbb{F}}$. An *assignment* θ denoted by

$$\theta : Z \to A,$$

is an $\mathbb{N}$-indexed function $\theta : Z \to |A|$. We may denote an assignment θ by the notation

$$\theta = [M_1 \mapsto a_1, \ldots, M_m \mapsto a_m]$$

which assigns a_i to M_i for $i = 1, \ldots, m$.

Lemma 4.16. Let $k \in \mathbb{N}$. If $(A, [\nu^A, [f^A]_{f\in\Sigma}])$ is a $(\mathrm{V}+\Sigma)$-algebra, so is $(\delta^k A, [\nu^{\delta^k}, [f^{\delta^k}]_{f\in\Sigma}])$, where $\nu^{\delta^k A}_n \stackrel{def}{=} \nu^A_{n+k}$, $f^{\delta^k A}_n \stackrel{def}{=} f^A_{n+k}$. Moreover, if $(A, [f^A]_{f\in\Sigma}, \nu^A, \mu^A)$ is a Σ-monoid, so is $(A, [f^{\delta^k A}]_{f\in\Sigma}, \nu^{\delta^k A}, \mu^{\delta^k A})$, where

$$\begin{aligned} \mu^{\delta^k A}_n : \delta^k A(m) \times \delta^k A(n)^m &\to \delta^k A(n) \\ a \quad ; b_1, \ldots, b_m &\mapsto \mu^A_{n+k}(a\,; b_1, \ldots, b_m, \nu^A_{n+k}(n+1), \ldots, \nu^A_{n+k}(n+k)). \end{aligned}$$

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

We now consider the case of assignment $\theta : Z \longrightarrow \delta^k \mathrm{T}_\Sigma \mathrm{V}$. It assigns a term to each metavariable of arity n as

$$Z(n) \ni M \overset{\theta_n}{\longmapsto} t \in \mathrm{T}_\Sigma \mathrm{V}(n+k)$$

where k is the number of additional possible free variables other than $1, \ldots, n$ in the results.

Additional free variables generated by an assignment θ are never captured by any binder. For example, let M be a 0-ary metavariable, and $\theta : Z \to \delta \mathrm{T}_\Sigma \mathrm{V}; \theta_0 : M \mapsto c(1)$. The free variable 1 in $c(1)$ is never captured by applying the substitution θ, which is automatically shifted to make it free to follow the method of de Bruijn levels, for example,

$$\begin{aligned}\theta_0^\sharp(\lambda(1.M)) &= \lambda(1.\theta_1^\sharp(M)) = \lambda(1.\mu_1^{\delta \mathrm{T}_\Sigma \mathrm{V}}(\theta_0(M);)) = \lambda(1.\mu_2^{\mathrm{T}_\Sigma \mathrm{V}}(\theta_0(M); \nu_2^{\mathrm{T}_\Sigma \mathrm{V}}(2))) \\ &= \lambda(1.\mu_2^{\mathrm{T}_\Sigma \mathrm{V}}(c(1); 2)) = \lambda(1.c(2)).\end{aligned}$$

In the named notation, it corresponds to the phenomenon of avoiding variable capture of bound variables. Applying the substitution $\theta : M \mapsto c(x)$ to $\lambda(x.M)$, we need to rename the binder variable x to x' as $\theta^\sharp(\lambda(x.M)) = \lambda(x'.c(x))$.

5. Algebraic Semantics of Rewriting

In this section, we interpret CSs by Σ-algebras equipped with binary relations modelling one-step rewriting. We give a complete characterisation of CSs with respect to the algebraic semantics. The basic idea is to follow the algebraic semantics of first-order TRSs by monotone Σ-algebras. A fundamental characterisation of terminating TRSs has been established.

Theorem 5.1. **(Huet and Lankford, 1978; Zantema, 1994)** A TRS is terminating if and only if there exists a non-empty well-founded monotone algebra that satisfies all rules of the TRS.

This proposition uses the ordinary first-order universal algebra, but the framework of the ordinary first-order universal algebra is insufficient for modelling CSs. We consider second-order computation in the framework of algebras in the presheaf category $\mathbf{Set}^{\mathbb{F}}$ equipped with binary relations.

5.1 Semantics

Definition 5.2. We say that a *presheaf* $A \in \mathbf{Set}^{\mathbb{F}}$ *is equipped with a binary relation* $>_A$ when

(i) $>_A$ is a family $\{>_{A(n)}\}_{n \in \mathbb{N}}$ of binary relations $>_{A(n)}$ on $A(n)$, and

(ii) for all $a, b \in A(m)$ and $\rho : m \to n$ in $\mathbb{F}$, if $a >_{A(m)} b$, then $A(\rho)(a) >_{A(n)} A(\rho)(b)$.

It will be denoted by $(A, >_A)$.

The condition (ii) means that each binary relation is compatible with a presheaf action.

We use the following notion of monotonicity. For a binary relation $>$, we write $a \geq b$ if $a > b$ or $a = b$,

Definition 5.3. Let $(A_1, >_{A_1}), \ldots, (A_m, >_{A_m}), (B, >_B)$ be presheaves equipped with binary relations. A morphism $f : A_1 \times \cdots \times A_m \longrightarrow B$ in $\mathbf{Set}^{\mathbb{F}}$ is *monotone* if

for all $n \in \mathbb{F}$, for all $a_1, b_1 \in A_1(n), \ldots, a_m, b_m \in A_m(n)$,
if there exists a single $k \in \{1, \ldots, m\}$ such that $(a_k >_{A(n)} b_k$ and for all $j \neq k$, $a_j = b_j)$,
then $f_n(a_1, \ldots, a_m) >_{B(n)} f_n(b_1, \ldots, b_m)$.

We interpret rewrite rules in a $(\mathrm{V}+\Sigma)$-algebra.

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

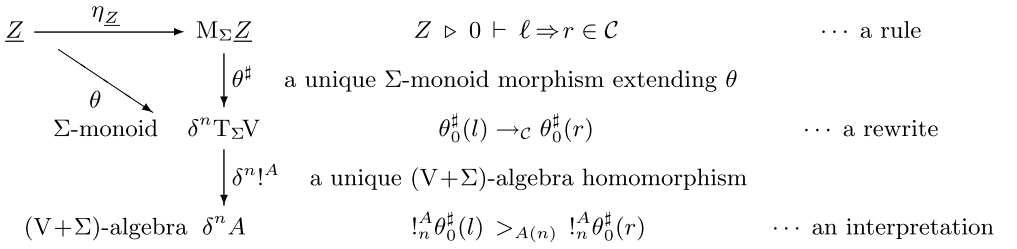


Figure 3. Interpretation of a rule.

Definition 5.4. Let A be a $(\mathrm{V}+\Sigma)$-algebra. Given an assignment $\theta : Z \to \delta^n \mathrm{T}_\Sigma \mathrm{V}$, a *term-generated assignment* $\widetilde{\theta} : Z \to \delta^n A$ is given by the composite map in $\mathbf{Set}^{\mathbb{F}}$

$$\underline{Z} \xrightarrow{\theta} \delta^n \mathrm{T}_\Sigma \mathrm{V} \xrightarrow{\delta^n !^A} \delta^n A$$

where $!^A$ is the unique $(\mathrm{V}+\Sigma)$-algebra homomorphism from the initial $(\mathrm{V}+\Sigma)$-algebra $\mathrm{T}_\Sigma \mathrm{V}$ to A. Throughout the paper, we denote by $!^A$ the unique $(\mathrm{V}+\Sigma)$-homomorphism.

This definition means that the interpretation of a metavariable M by a term-generated assignment ϕ is performed by firstly θ assigning some term t to M and then interpreting the term in a $(\mathrm{V}+\Sigma)$-algebra A. In other words, a term-generated assignment assigns a 'term-generated' element of A. This is because the (object) rewrite relation is defined on terms (not on meta-terms). To interpret a rewrite rule, unrestricted assignments $Z \to A$ are not suitable to model rewriting.

Definition 5.5. A *monotone* $(\mathrm{V}+\Sigma)$*-algebra* $(A, >_A)$ is a $(\mathrm{V}+\Sigma)$-algebra $A = (A, [\nu, [f^A]_{f\in\Sigma}])$, equipped with a binary relation $>_A$ on A such that every operation f^A is monotone. Moreover, when for every $n \in \mathbb{N}$, $>_{A(n)}$ is a well-founded relation, A is called *well-founded*.

Definition 5.6. Let $\mathcal{C}$ be a CS. A monotone $(\mathrm{V}+\Sigma)$-algebra $(A, >_A)$ *satisfies* a rewrite rule $Z \triangleright 0 \vdash \ell \Rightarrow r$ if for all term-generated assignments $\widetilde{\theta} : Z \to \delta^n A$,

$$\widetilde{\theta}_0^\sharp(\ell) >_{A(n)} \widetilde{\theta}_0^\sharp(r) \tag{5}$$

holds. Namely, for all assignments $\theta : Z \to \delta^n \mathrm{T}_\Sigma \mathrm{V}$,

$$!_n^A\theta_0^\sharp(\ell) >_{A(n)} !_n^A\theta_0^\sharp(r) \tag{6}$$

holds (see Figure 3).

Definition 5.7. A $(\mathrm{V}+\Sigma, \mathcal{C}, Z)$*-algebra* A is a monotone $(\mathrm{V}+\Sigma)$-algebra $(A, >_A)$ that satisfies all rules of a CS $(\Sigma, \mathcal{C}, Z)$ (cf. Figure 3).

For $n \in \mathbb{N}$, we define a relation $\to_{\mathcal{C}(n)} \stackrel{def}{=} \{(s, t) \mid n \vdash s \to_{\mathcal{C}} t\}$.

Proposition 5.8. The presheaf $\mathrm{T}_\Sigma \mathrm{V}$ of terms is equipped with the binary relation $\{\to_{\mathcal{C}(n)}\}_{n\in\mathbb{N}}$.

Proof. For a map $\rho : n \to n'$ in $\mathbb{F}$, let $\rho(t)$ denote a term by renaming each free variable using ρ. We show the compatibility: if we have a rewrite step $\triangleright n \vdash s \to_{\mathcal{C}} t$, then we have also $\triangleright n' \vdash \rho(s) \to_{\mathcal{C}} \rho(t)$ for any $\rho : n \to n'$ in $\mathbb{F}$. The derivation tree of the rewrite $\triangleright n \vdash s \to_{\mathcal{C}} t$ begins with

$$\frac{\triangleright 0 \vdash \ell \Rightarrow r}{\triangleright n \vdash \theta^\sharp(l) \to_{\mathcal{C}} \theta^\sharp(r)}$$

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

The derivation tree of $\triangleright n' \vdash \ \rho(s) \to_{\mathcal{C}} \rho(t)$ begins with the inference

$$\frac{\triangleright 0 \vdash \ \ell \Rightarrow r}{\triangleright n' \vdash \ \theta'^{\sharp}(l) \to_{\mathcal{C}} \theta'^{\sharp}(r)}$$

where $\theta' : M \mapsto \rho(\theta(M))$. Mimicking the derivation tree of the rewrite $\triangleright n \vdash \ s \to_{\mathcal{C}} t$ for this, finally we obtain $\triangleright n' \vdash \ \rho(s) \to_{\mathcal{C}} \rho(t)$. □

Proposition 5.9. For any $(\mathrm{V}+\Sigma, \mathcal{C})$-algebra A,

$$\triangleright n \vdash \ s \to_{\mathcal{C}} t \ \ \Rightarrow \ \ !^A_n(s) \ >_{A(n)} \ !^A_n(t).$$

Proof. By induction on the proof of $s \to_{\mathcal{C}} t$.

- Case (Rule). Suppose that

  $$\theta^{\sharp}_0(\ell) \to_{\mathcal{C}} \theta^{\sharp}_0(r)$$

  is derived from a rule $(Z \triangleright 0 \vdash \ \ell \Rightarrow r) \in \mathcal{C}$ with $\theta : Z \to \delta^n \mathrm{T}_{\Sigma}\mathrm{V}$. Since A is a $(\mathrm{V}+\Sigma, \mathcal{C})$-algebra,

  $$!^A_n \theta^{\sharp}_0(\ell) \ >_{A(n)} \ !^A_n \theta^{\sharp}_0(r).$$

- Case (Fun). Suppose that

  $$\triangleright n \vdash \ f(\ \ldots, n{+}1 \cdots n{+}i_j.t_j, \ldots) \Rightarrow_{\mathcal{C}} f(\ \ldots, n{+}1 \cdots n{+}i_j.t'_j, \ldots)$$

  is derived from

  $$f : \langle i_1, \ldots, i_m \rangle \in \Sigma \qquad \triangleright \ n + i_j \vdash \ t_j \Rightarrow_{\mathcal{C}} t'_j \quad \text{(some single } j \text{ s.t. } 1 \leq j \leq m)$$

  By I.H.,

  $$\theta^{\sharp}_{n+i_j} \ t_j \ >_{A(n+i_j)} \ \theta^{\sharp}_{n+i_j} \ t'_j$$

  Since f^A is monotone,

  $$\begin{aligned} \theta^{\sharp} f(\ \ldots, n{+}1 \cdots n{+}i_j.t_j, \ldots) = f^A(\ \ldots, \theta^{\sharp} \ t_j, \ldots) \ &>_{A(n)} \ f^A(\ \ldots, \theta^{\sharp} \ t'_j, \ldots) \\ &= \theta^{\sharp} f(\ \ldots, n{+}1 \cdots n{+}i_j.t'_j, \ldots) \end{aligned}$$

□

We obtain a complete characterisation of terminating CSs.

Theorem 5.10. A CS $\mathcal{C}$ is terminating if and only if there is a well-founded $(\mathrm{V}+\Sigma, \mathcal{C}, Z)$-algebra.

Proof. ($\Leftarrow$): Let A be a well-founded $(\mathrm{V}+\Sigma, \mathcal{C}, Z)$-algebra. Assume that $\mathcal{C}$ is non-terminating, that is, there exists an infinite reduction sequence

$$n \vdash t_1 \to_{\mathcal{C}} t_2 \to_{\mathcal{C}} \cdots .$$

By Proposition 5.9, we have

$$!_{A(n)}(t_1) \ >_{A(n)} \ !_{A(n)}(t_2) \ >_{A(n)} \ \cdots .$$

This contradicts well-foundedness of $>_A$.
($\Rightarrow$): When a CS $\mathcal{C}$ is terminating, the $(\mathrm{V}+\Sigma, \mathcal{C}, Z)$-algebra $(\mathrm{T}_{\Sigma}\mathrm{V}, \to_{\mathcal{C}})$ is a desired well-founded algebra, because the binary relation $\to_{\mathcal{C}}$ is well-founded. □

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

5.2 Benefit of completeness

In summary, we have a complete algebraic characterisation of rewriting of CSs as an extension of the first-order case (Huet and Lankford, 1978; Zantema, 1994).

Note that a known model of higher-order rules, the functional interpretation (van de Pol, 1996), is *incomplete* as the following example shows.

Example 5.11. (Incompleteness of functional interpretation (van de Pol, 1996)) Suppose a signature $\Sigma = \{c : \langle 0 \rangle\}$. Consider a CS $(\Sigma, \mathcal{C}, \{F^{(1)}, X^{(1)}, Y^{(0)}\})$ consisting of the following rule only:

$$F^{(1)}, X^{(1)}, Y^{(0)} \triangleright 0 \vdash c(F[F[X[Y]]]) \Rightarrow F[X[Y]].$$

We try to prove termination of $\mathcal{C}$. This CS is terminating because with any rewrite step the number of c-symbols decreases. Nevertheless the existing interpretation method of higher-order rewriting based on the model of hereditary monotone functionals *cannot show termination of* $\mathcal{C}$ due to the incompleteness of the model (van de Pol, 1994, 1996).

In contrast to it, we *can show* the termination of $\mathcal{C}$ by using our algebraic semantics as follows. Now we take the monotone $(\mathrm{V}+\Sigma)$-algebra $(\mathrm{T}_\Sigma \mathrm{V}, \succ_{\mathrm{T}_\Sigma \mathrm{V}})$ where

- $s \succ_{\mathrm{T}_\Sigma \mathrm{V}(n)} t$ if the number of c-symbols in s and t decreases.

Notice that now any term in $\mathrm{T}_\Sigma \mathrm{V}(n)$ consists of c and variables $1, \ldots, n$. Hence, any assignment into $\delta^n \mathrm{T}_\Sigma \mathrm{V}$ is of the form $F \mapsto c^k(x),\ X \mapsto c^m(x),\ Y \mapsto c^l(x)$ (k, m, l-times c's). This gives a well-founded $(\mathrm{V}+\Sigma, \mathcal{C}, Z)$-algebra $(\mathrm{T}_\Sigma \mathrm{V}, \succ_{\mathrm{T}_\Sigma \mathrm{V}})$, which implies the termination of $\mathcal{C}$ by Theorem 5.10.

Theorem 5.10 gives a complete method to prove termination of a CS by finding a suitable $(\mathrm{V}+\Sigma)$-algebra A and checking the satisfiability of rules in $\mathcal{C}$. But this method is impractical. The notion of $(\mathrm{V}+\Sigma, \mathcal{C}, Z)$-algebra uses the satisfiability by examining all *term-generated assignments*, which imposes that assignments must assign terms (not just elements in a model A) to metavariables. Namely, one need to consider all closed (w.r.t. metavariables) instances of left- and right-hand sides of the rule (see (6)). Since a $(\mathrm{V}+\Sigma)$-algebra A does not equip with a multiplication, one does not know a way to interpret meta-applications using only A.

In the next section, we improve this situation. We give a method that entails termination.

6. Algebraic Semantics of Meta-Rewriting: Monotone Σ-monoids

We have formulated the rewrite steps of a CS as a binary relation on terms. In this section, we define rewriting on meta-terms, which we call *meta-rewriting*. Let $(\Sigma, \mathcal{C}, Z)$ be a CS. We define the meta-rewriting relation $\Rightarrow_{\mathcal{C}}$ by the inference rules defined in Figure 4. Therefore, we consider the following notion of *meta-termination*.

Definition 6.1. We call a CS $(\Sigma, \mathcal{C}, Z)$ *meta-terminating* if the rewrite relation $\Rightarrow_{\mathcal{C}}$ is well-founded.

In this section, we give algebraic semantics of meta-rewriting. We follow basically the line of semantics given in Section 5, but we use Σ-monoids instead of Σ-algebras for the semantic structure.

6.1 Semantics

Definition 6.2. A *monotone* Σ*-monoid* $(A, >_A)$ is a Σ-monoid (A, ν^A, μ^A), where A is equipped with a binary relation $>_A$, such that every operation for Σ is monotone. Moreover, if $>_{A(n)}$ is a well-founded relation for each $n \in \mathbb{N}$, A is called *well-founded*.

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

$$\text{(Rule)}\ \frac{Z \rhd n + i_j \vdash\ s_j \quad (1 \le j \le m) \qquad \theta = [M_1 \mapsto s_1, \ldots, M_m \mapsto s_m] \\ (M_1^{(i_1)}, \ldots, M_m^{(i_m)}) \rhd 0 \vdash\ \ell \Rightarrow r) \in \mathcal{C}}{Z \rhd n \vdash\ \theta^\sharp(\ell) \Rightarrow_{\mathcal{C}} \theta^\sharp(r)}$$

$$\text{(Fun)}\ \frac{f : \langle i_1, \ldots, i_m \rangle \in \Sigma \quad Z \rhd n + i_j \vdash\ t_j \Rightarrow_{\mathcal{C}} t'_j \quad \text{(some single } j \text{ s.t. } 1 \le j \le m)}{Z \rhd n \vdash\ f(\ldots, n{+}1 \cdots n{+}i_j.t_j, \ldots) \Rightarrow_{\mathcal{C}} f(\ldots, n{+}1 \cdots n{+}i_j.t'_j, \ldots)}$$

$$\text{(Meta)}\ \frac{M \in Z(m) \quad Z \rhd n \vdash\ t_j \Rightarrow_{\mathcal{C}} t'_j \text{ for some single } j \\ t_i = t'_i \text{ for all } i \ne j \quad (1 \le i, j \le m)}{Z \rhd n \vdash\ M[t_1, \ldots, t_m] \Rightarrow_{\mathcal{C}} M[t'_1, \ldots, t'_m]}$$

Figure 4. Meta-rewriting (one-step).

Monotonicity of an operation corresponds to the (Fun) rule in Figure 4. Notice that the unit ν^A and the multiplication μ^A need not be monotone. The reasons are

- On the unit ν^A: there is no variable rewrite such as $x \Rightarrow y$.
- On the multiplication μ^A: (Meta) does not simply correspond to the monotonicity of μ^A. Rather, it should be modelled by admissible assignments, which we will give in Definition 6.4.

Definition 6.3. Let $(\Sigma, \mathcal{C}, Z)$ be a CS. A monotone Σ-monoid $(A, >_A)$ *satisfies* a rewrite rule $Z \rhd 0 \vdash\ \ell \Rightarrow r \in \mathcal{C}$ if

$$\phi_0^\sharp(\ell) >_{A(n)} \phi_0^\sharp(r)$$

for all assignments $\phi : Z \to \delta^n A$.

Definition 6.4. Let (A, ν, μ) be a monotone Σ-monoid, and $\phi : Z \to A$ an assignment. Define the map $\sigma : \underline{Z} \bullet A \longrightarrow A$ by the composite

$$\underline{Z} \bullet A \xrightarrow{\phi \bullet \mathrm{id}_A} A \bullet A \xrightarrow{\mu} A$$

The assignment ϕ is called *admissible* if σ is monotone.

The condition 'σ is monotone' means that each $\sigma_n^{(m)} : \underline{Z}(m) \times A(n)^m \longrightarrow A(n)$ is monotone, that is,

for all $m \in \mathbb{F}$, for all $(\xi, M) \in \underline{Z}(m)$ with $\xi : m' \to m$,
for all $n \in \mathbb{F}$, for all $a_1, b_1 \in A_1(n), \ldots, a_m, b_m \in A_m(n)$,
if there exists a single $k \in \{1, \ldots, m\}$ such that $(a_k >_{A(n)} b_k$ and for all $j \ne k$, $a_j = b_j)$,
then $\mu_n^A(A(\xi)(\phi_{m'}(M)); a_1, \ldots, a_m) >_{A(n)} \mu_n^A(A(\xi)(\phi_{m'}(M)); b_1, \ldots, b_m)$.

The notion of admissible assignments is a necessary ingredient of interpretation of meta-rewriting. Arbitrary assignments are not suitable to interpret meta-rewriting. An arbitrary assignment may not preserve the rewrite relation as the following example shows.

Example 6.5. Suppose the constants $\Sigma = \{a, b, c : \langle\rangle\}$, the metavariable set $Z = \{M^{(1)}\}$ and the CS

$$\mathcal{C} = \{ \rhd 0 \vdash\ a \Rightarrow b \}.$$

Then we have a meta-rewriting $M[a] \Rightarrow_{\mathcal{C}} M[b]$. We interpret this rewrite step in a monotone Σ-monoid $(\mathrm{M}_\Sigma \underline{Z}, [a^{\mathrm{M}_\Sigma}, b^{\mathrm{M}_\Sigma}, c^{\mathrm{M}_\Sigma}], \nu, \mu), \Rightarrow_{\mathcal{C}})$ (cf. Theorem 4.13). It satisfies the rule $a \Rightarrow_{\mathcal{C}} b$. Take

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

an assignment $\phi : M \longmapsto c$. Then, the interpretation using ϕ does not preserve the relation:

$$\phi^\sharp(M[a]) = c \not\Rightarrow_{\mathcal{C}} c = \phi^\sharp(M[b]). \tag{7}$$

We need a 'monotonic' interpretation of meta-rewriting to establish an algebraic termination proof method. The idea of admissible assignments is motivated by prohibiting this kind of 'non-monotonic' interpretation of a rewrite step.

This problem has already been recognised by van de Pol (1994, Example 5), (1996, Section 4.3). The notion of admissible assignments is analogous to his notion of strict valuations.

Definition 6.6. A $(\Sigma, \mathcal{C}, Z)$-*monoid* is a monotone Σ-monoid $(A, >_A)$ such that

(i) A satisfies all rules of a CS $(\Sigma, \mathcal{C}, Z)$, and

(ii) there is an admissible assignment $Z \longrightarrow A$.

Lemma 6.7. If $(A, >_A)$ is a $(\Sigma, \mathcal{C}, Z)$-monoid, so is $(\delta^n A, >_{\delta^n A})$.

Proof. Let $(A, >_A)$ be a $(\Sigma, \mathcal{C}, Z)$-monoid having an admissible assignment $\phi : Z \to A$. Then $(\delta^n A, >_{\delta^n A})$ is also a monotone Σ-monoid, where $a >_{\delta^n A(k)} b \overset{def}{\iff} a >_{A(n+k)} b$. Let $\ell \Rightarrow r \in \mathcal{C}$. Since A satisfies the rule, for all $\psi : Z \longrightarrow \delta^n A$,

$$\psi_0^\sharp(\ell) >_{A(n)} \psi_0^\sharp(r)$$

holds. Therefore, $\delta^n A$ satisfies the rule. There is an admissible assignment defined by

$$Z \xrightarrow{\phi} A \xrightarrow{\iota} \delta^n A$$

where $\iota_k = A(\mathsf{up}_k^{k+n}) : A(k) \to A(k+n)$. Hence $(\delta^n A, >_{\delta^n A})$ is a $(\Sigma, \mathcal{C}, Z)$-monoid. □

We will clarify a sufficient condition ensuring the existence of an admissible assignment in Section 8. A $(\Sigma, \mathcal{C}, Z)$-monoid is a model for meta-rewriting as a Definition 5.6 for rewriting. For $n \in \mathbb{N}$, we define a $\mathbb{N}$-indexed relation $\Rightarrow_{\mathcal{C}(n)} \overset{def}{=} \{(s,t) \mid Z \triangleright n \vdash s \Rightarrow_{\mathcal{C}} t\}$. An important example of $(\Sigma, \mathcal{C}, Z)$-monoids is as follows.

Proposition 6.8. $(\mathrm{M}_\Sigma \underline{Z}, \Rightarrow_{\mathcal{C}})$ is a $(\Sigma, \mathcal{C}, Z)$-monoid.

Proof. By Theorem 4.13, $\mathrm{M}_\Sigma \underline{Z}$ is a Σ-monoid, and so is $\mathrm{M}_\Sigma \underline{Z}$. Because of the (Fun)-rule, every operation $f^{\mathrm{M}_\Sigma \underline{Z}}$ is monotone. We check the conditions of Definition 6.6.

(i) By (Rule), for every rule $\ell \Rightarrow r$ in $\mathcal{C}$ and assignment $\theta : Z \to \delta^n \mathrm{M}_\Sigma \underline{Z}$, we have $\theta_0^\sharp(\ell) \Rightarrow_{\mathcal{C}} \theta_0^\sharp(r)$. Hence $\mathrm{M}_\Sigma \underline{Z}$ satisfies all rules in $\mathcal{C}$.

(ii) There is an admissible assignment $\theta : Z \longrightarrow \mathrm{M}_\Sigma \underline{Z}$ defined by $M \mapsto M[1, \ldots, n]$ for each $M \in Z(n)$.

□

Lemma 6.9. Let $\phi : \underline{Z} \to A$ be a map of $\mathbf{Set}^{\mathbb{F}}$ and $\psi : A \to B$ a Σ-monoid morphism. We have $(\psi \circ \phi)^\sharp = \psi \circ \phi^\sharp$.

Proof. By freeness of $\mathrm{M}_\Sigma \underline{Z}$, the unique Σ-monoid morphism that extends ϕ followed by a Σ-monoid morphism is represented by the unique Σ-monoid morphism that extends $\psi \circ \phi$. □

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

Proposition 6.10. Let $(\Sigma, \mathcal{C}, Z)$ be a CS and $(A, >_A)$ a $(\Sigma, \mathcal{C}, Z)$-monoid. For any admissible assignment $\phi : Z \to A$,

$$Z \triangleright n \vdash s \Rightarrow_{\mathcal{C}} t \;\Rightarrow\; \phi_n^\sharp(s) >_{A(n)} \phi_n^\sharp(t)$$

Proof. By induction on the proof of $s \Rightarrow_{\mathcal{C}} t$.

- Case (Rule). Suppose that

  $$Z \triangleright n \vdash \theta_0^\sharp(\ell) \Rightarrow_{\mathcal{C}} \theta_0^\sharp(r)$$

  is derived from a rule $(Z \triangleright 0 \vdash \ell \Rightarrow r) \in \mathcal{C}$ with $\theta : Z \to \delta^n \mathrm{M}_\Sigma \underline{Z}$. Let $\phi : Z \to A$ be an admissible assignment. Take an assignment ψ as

  $$\psi = \underline{Z} \xrightarrow{\theta} \delta^n \mathrm{M}_\Sigma \underline{Z} \xrightarrow{\delta^n \phi^\sharp} \delta^n A$$

  By Lemma 6.9, $\psi^\sharp = (\delta^n \phi)^\sharp \circ \theta^\sharp$. By Lemma 6.7, since A is a $(\Sigma, \mathcal{C}, Z)$-monoid, so is $\delta^n A$. Therefore,

  $$\phi_n^\sharp \theta_0^\sharp(\ell) = \psi_0^\sharp(\ell) >_{A(n)} \psi_0^\sharp(r) = \phi_n^\sharp \theta^\sharp(r)$$

- Case (Fun). Similarity to the case (Fun) in the proof of Proposition 5.9.
- Case (Meta) Suppose that $Z \triangleright n \vdash M[t_1, \ldots, t_l] \Rightarrow_{\mathcal{C}} M[t'_1, \ldots, t'_l]$ is derived from

  $$Z \triangleright n \vdash t_j \Rightarrow_{\mathcal{C}} t'_j \text{ for some single } j, \text{ and } t_i = t'_i \text{ for all } i \neq j \quad (1 \leq i, j \leq l)$$

  By I.H., $\phi^\sharp(t_j) >_{A(n)} \phi^\sharp(t'_j)$. Since ϕ is admissible,

  $$\begin{aligned} \phi_n^\sharp(M[\ldots, t_j, \ldots]) = \mu_n^A(\phi_l(M); \ldots, \phi_n^\sharp(t_j), \ldots) &>_{A(n)} \mu_n^A(\phi_l(M); \ldots, \phi_n^\sharp(t'_j), \ldots) \\ &= \phi_n^\sharp(M[\ldots, t'_j, \ldots]) \end{aligned}$$

□

Theorem 6.11. A CS $(\Sigma, \mathcal{C}, Z)$ is meta-terminating if and only if there is a well-founded $(\Sigma, \mathcal{C}, Z)$-monoid.

Proof. ($\Leftarrow$): Let A be a well-founded $(\Sigma, \mathcal{C}, Z)$-monoid. Assume that $\mathcal{C}$ is not meta-terminating, that is, there exists an infinite meta-rewriting sequence

$$Z \triangleright n \vdash t_1 \Rightarrow_{\mathcal{C}} t_2 \Rightarrow_{\mathcal{C}} t_3 \Rightarrow_{\mathcal{C}} \cdots.$$

By Proposition 6.10, for any admissible assignment $\phi : Z \to A$,

$$\phi_n^\sharp(t_1) >_{A(n)} \phi_n^\sharp(t_2) >_{A(n)} \phi_n^\sharp(t_3) >_{A(n)} \cdots.$$

This contradicts well-foundedness of $>_A$.
($\Rightarrow$): When a CS $\mathcal{C}$ is meta-terminating, the $(\Sigma, \mathcal{C}, Z)$-monoid $(\mathrm{M}_\Sigma \underline{Z}, \Rightarrow_{\mathcal{C}})$ by Proposition 6.8 is a desired well-founded one because the relation $\Rightarrow_{\mathcal{C}}$ is well-founded. □

Remark 6.12. Notice again that the notion of admissible assignments is only used to interpret a meta-rewrite sequence in a Σ-monoid A in the above proof of the soundness. We can allow non-admissible assignments in meta-rewriting steps. For example, in the case of the CS $\mathcal{C}$ of the untyped λ-calculus, we have

$$\frac{\begin{array}{l}\theta = [M \mapsto c,\ N \mapsto a] \\ M^{(1)}, N^{(0)} \triangleright 0 \vdash \lambda(x.M)@N \Rightarrow M[N] \in \mathcal{C}\end{array}}{\lambda(x.c)@a \Rightarrow_{\mathcal{C}} c}\ (\text{Rule})$$

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

where θ is not admissible. Nevertheless, it is a correct meta-rewriting step and a valid reduction in the λ-calculus.

Theorem 6.11 also serves as a method to prove termination of a CS. In practice, applying Theorem 6.11 to a termination problem is easier than applying Theorem 5.10. Theorem 5.10 requires to check the interpretations of *all instances* of rules by term-generated assignments. Considering all instances means that we need to use induction or case analysis on terms, which is tedious.

On the other hand, Theorem 6.11 uses merely *all* admissible assignments into a Σ-monoid for interpretation, which is simpler to consider.

7. Hereditary Monotone Functionals as a Σ-monoid

By Theorem 6.11, we can show the meta-termination of a given CS $\mathcal{C}$ by finding a well-founded monotone $(\Sigma, \mathcal{C}, Z)$-monoid. It can be regarded as a second-order extension of Theorem 5.1 for the termination of first-order TRSs.

How to find such a well-founded $(\Sigma, \mathcal{C}, Z)$-monoid for termination is an important problem in practice. The first issue is to find *an example of* Σ*-monoid*. Trying to define a Σ-algebra structure is usually not problematic, but finding a suitable compatible multiplication μ on it is generally hard. Fortunately, there is a guideline. There are two typical classes of Σ-monoids:

(i) A Σ-monoid of a *syntactic term structure*, where the syntactic substitution as the multiplication.

(ii) A Σ-monoid of (suitably restricted) *function spaces*, where the composition of functions as the multiplication.

As an example in the class (i), we have considered the Σ-monoid $\mathrm{T}_\Sigma \mathrm{V}$ of terms, and the Σ-monoid $\mathrm{M}_\Sigma \underline{Z}$ of meta-terms. In this class of Σ-monoids, the multiplication is compatible with the algebra structure (i.e. satisfying the Σ-monoid law), because the syntactic substitution is recursively defined on the term structure.

An example in the class (ii) is the structure called *clone* (short for closed sets of operations (Cohn, 1965)) known in universal algebra. A clone is a family of functions with all projections closed under composition. Therefore, the composition can be used as the multiplication of a Σ-monoid of clone.

For the above-mentioned class (ii) of clones, to define a suitable order structure on a clone, we need to analyse function spaces. Following Gandy's early work on functional interpretation of strong normalisation of typed λ-calculus (Gandy, 1980), (van de Pol, 1994, 1996) has defined the order structure on function spaces called *hereditary monotone functionals* for termination of higher-order rewrite systems in the format of Mayr and Nipkow (1998). In this section, using hereditary monotone functionals, we show that we can construct a monotone Σ-monoid.

7.1 Hereditary monotone functionals

Given a set B of base types, we define the set $\mathbb{T}$ of simple types as the least set satisfying

$$\mathbb{T} = B \cup \{\sigma \to \tau \mid \sigma, \tau \in \mathbb{T}\} \cup \{\sigma \times \tau \mid \sigma, \tau \in \mathbb{T}\}$$

Below we define a set D_τ for each $\tau \in \mathbb{T}$ by the set of all hereditary monotone functionals, with the monotone order ${}_{\mathrm{mon}}>$ and the strictly monotone order ${}_{\mathrm{str}}>$, parameterised by types (van de Pol, 1994).

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

Definition 7.1. **((van de Pol, 1994) [Definition 11])** Let $(A, >_A)$ be a non-empty set A with a strict partial order $>_A$ on it. A $\mathbb{T}$-indexed set of *hereditary monotone functionals* with families of (strict) partial orders

$$(\mathrm{D}, \{{}_{\mathrm{mon}}{>}_\tau\}_{\tau\in\mathbb{T}}, \{{}_{\mathrm{mon}}{\geq}_\tau\}_{\tau\in\mathbb{T}})$$

is defined by

(i) $\mathrm{D}_\iota = A$, with ${}_{\mathrm{mon}}{>}_\iota \stackrel{def}{=} >_A$, and ${}_{\mathrm{mon}}{\geq}_\iota \stackrel{def}{=} {}_{\mathrm{mon}}{>}_\iota \cup \mathrm{id}$ for base types ι.

(ii) $\mathrm{D}_{\sigma\times\tau} = \mathrm{D}_\sigma \times \mathrm{D}_\tau$, and $(x_1, x_2)\ {}_{\mathrm{mon}}{>}_{\sigma\times\tau}\ (y_1, y_2)$
$\stackrel{def}{\Longleftrightarrow} (x_1 = y_1 \text{ and } x_2\ {}_{\mathrm{mon}}{>}_\tau\ y_2)$ or $(x_1\ {}_{\mathrm{mon}}{>}_\sigma\ y_1 \text{ and } x_2\ {}_{\mathrm{mon}}{\geq}_\tau\ y_2)$,
$(x_1, x_2)\ {}_{\mathrm{mon}}{\geq}_{\sigma\times\tau}\ (y_1, y_2) \stackrel{def}{\Longleftrightarrow} (x_1\ {}_{\mathrm{mon}}{\geq}_\sigma\ y_1 \text{ and } x_2\ {}_{\mathrm{mon}}{\geq}_\tau\ y_2)$.

(iii) $\mathrm{D}_{\sigma\to\tau} = \{f : \mathrm{D}_\sigma \to \mathrm{D}_\tau \mid \text{for all } x, y \in \mathrm{D}_\sigma,\ x\ {}_{\mathrm{mon}}{\geq}_\sigma\ y \ \Rightarrow\ f(x)\ {}_{\mathrm{mon}}{\geq}_\tau\ f(y)\}$.
$f\ {}_{\mathrm{mon}}{>}_{\sigma\to\tau}\ g \stackrel{def}{\Longleftrightarrow}$ for all $x \in \mathrm{D}_\sigma$, $f(x)\ {}_{\mathrm{mon}}{>}_\tau\ g(x)$ in D_τ.
${}_{\mathrm{mon}}{\geq}_{\sigma\to\tau} \stackrel{def}{=} {}_{\mathrm{mon}}{>}_{\sigma\to\tau} \cup \mathrm{id}$, where $\sigma, \tau \in \mathbb{T}$.

We call a function in $\mathrm{D}_{\sigma\to\tau}$ a *hereditary monotone functional*. In van de Pol (1994, 1996), it was called a weakly monotone functional.

7.2 Presheaf with strict partial orders

We define the *presheaf of hereditary monotone functionals* $\mathrm{H} \in \mathbf{Set}^{\mathbb{F}}$ as follows. We assume the only base type ι.

$$\mathrm{H}(0) = \mathrm{D}_\iota$$
$$\mathrm{H}(n) = \mathrm{D}_{\iota^n\to\iota} \quad (\text{for } n > 0) \qquad \mathrm{H}(\rho)(f) = f \circ \langle \pi_{\rho 1}, \ldots, \pi_{\rho m}\rangle$$

where $\rho : m \to n$, and the projections $\pi_i(d_1, \ldots, d_n) = d_i$, and the pairing $\langle g_1, \ldots, g_n\rangle(a) = (g_1(a), \ldots, g_n(a))$. It is equipped with the strict partial orders defined by

$$>_{\mathrm{H}(0)} \stackrel{def}{=} {}_{\mathrm{mon}}{>}_\iota \qquad , \qquad >_{\mathrm{H}(n)} \stackrel{def}{=} {}_{\mathrm{mon}}{>}_{\iota^n\to\iota}$$

7.3 Monoid

The presheaf H forms a monoid in $\mathbf{Set}^{\mathbb{F}}$. For the unit $\nu_n^{\mathrm{H}} : \mathrm{V}(n) \to \mathrm{H}(n)$, we take

$$\nu_n^{\mathrm{H}}(i) = \pi_i$$

The multiplication $\mu_n^{\mathrm{H}(m)} : \mathrm{H}(m)\times\mathrm{H}(n)^m \longrightarrow \mathrm{H}(n)$ is defined by the composition:

$$f \in \mathrm{H}(m)\,; g_1, \ldots, g_m \in \mathrm{H}(n) \longmapsto f \circ \langle g_1, \ldots, g_m\rangle.$$

Then these satisfy the monoid laws (i)–(iii) given in Definition 4.5.

7.4 Σ-monoid

Suppose that given a signature Σ, we could define a monotone Σ-algebra

$$(\mathrm{H}, [f^{\mathrm{H}}]_{f\in\Sigma}, >^{\mathrm{H}}).$$

Each operation in the context 0 is a hereditary monotone functional. Next we must verify that H forms a Σ-monoid. This is not automatic and depends on the chosen hereditary monotone

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

functionals. If we took suitable ones, then the Σ-monoid laws hold. 'Second-order polynomials' are such suitable ones, which we will see next.

7.5 Example of termination proof using a Σ-monoid of hereditary monotone functionals

Consider the following CS $\mathcal{C}$ for case expressions (van de Pol, 1994, 1996) for sums with η-reduction. Let $\Sigma = \{\mathsf{inl}, \mathsf{inr} : \langle 0\rangle, \mathsf{case} : \langle 0, 1, 1\rangle\}$.

$$\begin{array}{lllll}
\text{(C1)} & X^{(0)}, F^{(1)}, G^{(1)} & \triangleright\ 0 & \vdash\ \mathsf{case}(\mathsf{inl}(X), F, G) & \Rightarrow F[X] \\
\text{(C2)} & Y^{(0)}, F^{(1)}, G^{(1)} & \triangleright\ 0 & \vdash\ \mathsf{case}(\mathsf{inr}(Y), F, G) & \Rightarrow G[Y] \\
\text{(C3)} & E^{(0)}, H^{(1)} & \triangleright\ 0 & \vdash\ \mathsf{case}(E, x.H[\mathsf{inl}(x)], y.H[\mathsf{inr}(y)]) & \Rightarrow H[E]
\end{array}$$

It is known that the known syntactic criteria of termination, such as the General Schema (Blanqui, 2000, 2016) criterion, cannot deal with the termination of this example because the lhs of the final rule is not a higher-order pattern (Miller, 1991). We show the termination of $\mathcal{C}$ by using the Σ-monoid H of hereditary monotone functionals. Now we take a monotone Σ-monoid $(\mathrm{H}, >_{\mathrm{H}})$ generated by $(\mathbb{N}, >)$ where $>$ is the usual order on the natural numbers $\mathbb{N}$. So, $\mathrm{H}(0) = \mathbb{N}$. We take operations as follows.

$$\begin{array}{l}
\nu^{\mathrm{H}} : \mathrm{V}(n) \to \mathrm{H}(n); \qquad \nu^{\mathrm{H}}_n(i) = \pi_i \\
\mathsf{inl}^{\mathrm{H}}, \mathsf{inr}^{\mathrm{H}} : \mathrm{H}(n) \to \mathrm{H}(n); \quad \mathsf{inl}^{\mathrm{H}}_n(x) = \mathsf{inr}^{\mathrm{H}}_n(x) = x \\
\mathsf{case}^{\mathrm{H}}_n : \mathrm{H}(n) \times \mathrm{H}(n+1) \times \mathrm{H}(n+1) \to \mathrm{H}(n) \\
\mathsf{case}^{\mathrm{H}}_n(z, f, g) = f \circ \langle \mathsf{Id}, z\rangle + g \circ \langle \mathsf{Id}, z\rangle + z + \mathbf{1}
\end{array}$$

where the constant function $\mathbf{1}(u) = 1$ and the function pairing is defined by $\langle f, g\rangle(x) = (f(x), g(x))$. The operations are indeed monotone. Then the Σ-monoid law holds. For instance, the Σ-monoid law (iv) for $\mathsf{case}^{\mathrm{H}}$ at $n = 1$:

$$\mu^{(1)}_1(\mathsf{case}^{\mathrm{H}}_1(z, f, g); b) \;=\; \mathsf{case}^{\mathrm{H}}_1(\mu^{(1)}_1(z, b), \mu^2_{(2)}(f; \mathsf{up}^2_1(b), \nu_2(2)), \mu^2_{(2)}(g; \mathsf{up}^2_1(b), \nu_2(2)))$$

holds because

$$\begin{array}{l}
\mathit{lhs} = (f \circ \langle \mathsf{Id}, z\rangle + g \circ \langle \mathsf{Id}, z\rangle + z + \mathbf{1}) \circ b \\
\quad = f \circ \langle b, z \circ b\rangle + g \circ \langle b, z \circ b\rangle + z \circ b + \mathbf{1} \\
\mathit{rhs} = f \circ \langle \mathsf{up}^2_1(b), \pi_2\rangle \circ \langle \mathsf{Id}, z \circ b\rangle + g \circ \langle \mathsf{up}^2_1(b), \pi_2\rangle \circ \langle \mathsf{Id}, z \circ b\rangle + z \circ b + \mathbf{1} \\
\quad = f \circ \langle b \circ \pi_1 \circ \langle \mathsf{Id}, z \circ b\rangle,\ \pi_2 \circ \langle \mathsf{Id}, z \circ b\rangle\rangle + g \circ \langle b \circ \pi_1 \circ \langle \mathsf{Id}, z \circ b\rangle,\ \pi_2 \circ \langle \mathsf{Id}, z \circ b\rangle\rangle + z \circ b + \mathbf{1} \\
\quad = \mathit{lhs}
\end{array}$$

The general case of other operations is proved similarly.

The Σ-monoid H satisfies the rules. The interpretations of both sides of all the rules are decreasing: for all $x, y, e \in \mathbb{N} = \mathrm{H}(0)$, and for all $f, g, h \in \mathbb{N} \to \mathbb{N} \subseteq \mathrm{H}(1)$,

$$\begin{array}{ll}
\text{(C1)} & \mathit{lhs} = f(x) + g(x) + x + 1 \;>\; f(x) = \mathit{rhs} \\
\text{(C2)} & \mathit{lhs} = f(y) + g(y) + y + 1 \;>\; g(y) = \mathit{rhs} \\
\text{(C3)} & \mathit{lhs} = h(e) + h(e) + e + 1 \;>\; h(e) = \mathit{rhs}
\end{array}$$

There is an admissible assignment $\phi : Z \longrightarrow \mathrm{H}$ defined by

$$X^{(0)}, Y^{(0)}, E^{(0)} \mapsto 1, \quad F^{(1)}, G^{(1)}, H^{(1)} \mapsto \lambda x.x \tag{8}$$

where the $\lambda\lambda$-notation denotes a meta-level function. Thus, $((\mathrm{H}, \alpha, \nu, \mu), >_{\mathrm{H}})$ forms a well-founded $(\Sigma, \mathcal{C}, Z)$-monoid. By Theorem 6.11, the CS $\mathcal{C}$ is meta-terminating.

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

8. On the Existence of Admissible Assignments

To give a $(\Sigma, \mathcal{C}, Z)$-monoid, we have required the existence of an admissible assignment in Definition 6.6 (ii). In this section, we discuss the following natural questions:

(1) Does an admissible assignment always exist for a given CS $(\Sigma, \mathcal{C}, Z)$ and a candidate monotone Σ-monoid?

(2) If not, when does an admissible assignment exist?

8.1 Analysis

For the question (1), the answer is No. Consider a CS $(\{a, b : \langle\rangle\}, \{\triangleright 0 \vdash\ a \Rightarrow b\}, Z)$ where $Z = \{N^{(2)}\}$. The monotone Σ-monoid $(\mathrm{T}_\Sigma \mathrm{V}, \to_{\mathcal{C}})$ satisfies the rule $a \Rightarrow b$. Consider an assignment $\phi : Z \longrightarrow \mathrm{T}_\Sigma \mathrm{V}$. Since now only the 2-ary metavariable $N \in Z(2)$ exists, possible mappings of $\phi_2 : \underline{Z}(2) \longrightarrow \mathrm{T}_\Sigma \mathrm{V}(2)$ are the four cases:

$$\phi_2 : N \mapsto a, \quad \phi_2 : N \mapsto b, \quad \phi_2 : N \mapsto 1 \quad \phi_2 : N \mapsto 2.$$

But none of these is admissible. The first two mappings are not admissible as in (7). For $\phi_2 : N \mapsto 1$, which assigns the variable 1 to N, interpreting a meta-rewriting $N[b, a] \Rightarrow_{\mathcal{C}} N[b, b]$, we have

$$\phi^\sharp(N[b, a]) = b \not\to_{\mathcal{C}} b = \phi^\sharp(N[b, b]).$$

For $\phi_2 : N \mapsto 2$, similarly interpreting a meta-rewriting $N[a, a] \Rightarrow_{\mathcal{C}} N[b, a]$, we have

$$\phi^\sharp(N[a, a]) = a \not\to_{\mathcal{C}} a = \phi^\sharp(N[b, a]).$$

These failures are due to the fact that the monotone Σ-monoid $(\mathrm{T}_\Sigma \mathrm{V}, \to_{\mathcal{C}})$ lacks 'an operation, which is monotonic in each argument'.

We next examine a successful case. We take the monotone Σ-monoid of hereditary monotone functionals $(\mathrm{H}, >_{\mathrm{H}})$ generated by $(\mathbb{N}, >)$ as in Section 7.5, where we take operations $a_n^{\mathrm{H}} = \mathbf{9}$, $b_n^{\mathrm{H}} = \mathbf{8}$, that is, constant functions returning 9 and 8. Since $9 > 8$, this satisfies the rule $a \Rightarrow b$. Consider an assignment $\phi_2 : \underline{Z}(2) \longrightarrow \mathrm{H}(2)$ defined by

$$\phi_2 : N \mapsto \lambda xy.x + y.$$

It is admissible because applying ϕ, a rewrite sequence

$$0 \vdash N[a, a] \ \Rightarrow_{\mathcal{C}}\ N[b, a] \ \Rightarrow_{\mathcal{C}}\ N[b, b]$$

is interpreted as

$$9 + 9 \ >\ 8 + 9 \ >\ 8 + 8.$$

In contrast to this, $(\mathrm{T}_\Sigma \mathrm{V}, \to_{\mathcal{C}})$ lacked such a '+'-like operation.

In general, rather than a binary '+', what we need is a "sum"-like operation that combines n-elements in a model. So, we call it sum in the next proposition.

Proposition 8.1. Let $((A, \nu^A, \mu^A), >_A)$ be a monotone Σ-monoid, and Z a metavariable set. Suppose that for each $n \geq 2$ with $Z(n) \neq \varnothing$, there exists a monotone morphism $\mathsf{sum}^n : A^n \to A$ of $\mathbf{Set}^{\mathbb{F}}$ such that

$$\mu^A(\mathsf{sum}^n(a_1, \ldots, a_n); \overrightarrow{b}) = \mathsf{sum}^n(\mu^A(a_1; \overrightarrow{b}), \cdots, \mu(a_n; \overrightarrow{b})).$$

Then an admissible assignment $\phi : Z \longrightarrow A$ exists and is given by

$$\phi_n(M) = \mathsf{sum}^n(\nu^A(1), \ldots, \nu^A(n)) \qquad \text{for each } M \in Z(n), n \geq 2.$$

Proof. Using the compatibility of sum^n with the multiplication, we see that the morphism σ in Definition 6.4 is monotone, hence ϕ is admissible. □

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

For a 0- or 1-ary metavariable M, we do not need such a sum, because mapping $M^{(0)}$ to an element in A is no problem, and we can always take the mapping $M^{(1)}$ to the variable 1, which is admissible.

8.2 Examples

We list examples of monotone morphisms $\mathsf{sum}^n : A^n \to A$ for known monotone Σ-monoids.

- Case $\mathrm{M}_\Sigma\underline{Z}$. $\mathsf{sum}^n : a_1, \ldots, a_n \longmapsto M[a_1, \ldots, a_n]$. It has been implicitly used in the proof of Proposition 6.8, that is, the admissible assignment $\phi_n : M \mapsto M[1, \ldots, n]$ was obtained from this sum^n.
- Case H of the Σ-monoid of hereditary monotone functionals defined in Section 7. $\mathsf{sum}^n : a_1, \ldots, a_n \longmapsto a_1 + \cdots + a_n$. Note that van de Pol has considered a similar functional S of 'summing up measures' (van de Pol, 1996)[Definition 4.3.5, Section 4.4] in his hereditary functional model in order to show the existence of strict functionals.
- Case $\mathrm{T}_\Sigma\mathrm{V}$. There is no canonical choice. One possible example is
  - If there is $f : \langle 0, 0\rangle \in \Sigma$, then $f^{\mathrm{T}_\Sigma\mathrm{V}} : \mathrm{T}_\Sigma\mathrm{V} \times \mathrm{T}_\Sigma\mathrm{V} \to \mathrm{T}_\Sigma\mathrm{V}$ is monotone, and compatible with the multiplication. Thus, we can take

$$\mathsf{sum}^n : a_1, \ldots, a_{n-1}, a_n \longmapsto f^{\mathrm{T}_\Sigma\mathrm{V}}(a_1, \ldots, f^{\mathrm{T}_\Sigma\mathrm{V}}(a_{n-1}, a_n)).$$

9. Termination of Binding CSs

Let $(\Sigma, \mathcal{C}, Z)$ be a CS such that every meta-application occurring in rules of $\mathcal{C}$ is of the form $M^l[x_1, \ldots, x_l]$, where every x_i is a variable. We call such a CS a *binding CS* because it is essentially meta-application free (see also a similar notion of binding TRSs (Hamana, 2003)). To interpret a rule and meta-rewriting in a binding CS $\mathcal{C}$, we do not need the monoid structure of Σ-monoids, that is, the multiplication μ is not used. In this section, we investigate the termination of binding CSs.

For example, interpreting a meta-term $M[1, 2]$ (for a metavariable $M^{(2)}$) in a rule by an assignment $\phi : Z \to A$ into a Σ-monoid (A, ν, μ), we have

$$\phi_2^\sharp(M[1, 2]) = \mu_2^{(2)}(\phi_2(M); \nu_2(1), \nu_2(2)) = \phi_2(M).$$

The second equation is due to the right unit law of the Σ-monoid A (Definition 4.5 (ii)). So, to interpret a meta-term like $M[1, 2]$, we need merely an assignment ϕ. Similarly, when the arguments of a meta-application are arbitrary variables $n_1, \ldots, n_l \in \mathbb{N}$, we have

$$\phi^\sharp(M[n_1, \ldots, n_l]) = \mu(\phi(M); \nu(n_1), \ldots, \nu(n_l)) = \mu(\phi(A(\xi)(M)); \nu(1), \ldots, \nu(l)) = \phi(M^\xi)$$

where $M \in Z(l); \xi : l \to n$, $\xi(i) = n_i$ $(1 \le i \le l)$, and $(\xi, M) \in \underline{Z}(n)$, because $A \bullet V$ is defined as a quotient making the above middle equation equal.

We call a meta-term a *binding meta-term* when any meta-application in it is of the form $M[n_1, \ldots, n_l]$, where $n_i \in \mathbb{N}$.

Binding meta-terms are meta-terms generated by a signature Σ, variables V and metavariables Z, where metavariables are substitutable syntactic objects. Hence, we characterise it as a free $(\mathrm{V}+\Sigma)$-algebra over $\underline{Z}$ defined by $\mathrm{B}_\Sigma\underline{Z} \stackrel{def}{=} \mathrm{T}_{\mathrm{V}+\Sigma}(\underline{Z})$. Syntactically, $\mathrm{B}_\Sigma\underline{Z}$ is constructed by a construction rule

$$(\mathrm{BMeta})\frac{M \in Z(l) \quad 1 \le n_i \le n\ (i = 1, \ldots, l)}{n \vdash M[n_1, \ldots, n_l] \in \mathrm{B}_\Sigma\underline{Z}(n)}$$

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

and the construction rules for variables and function terms in Figure 1. The free $(\mathrm{V}+\Sigma)$-algebra $\mathrm{B}_\Sigma \underline{Z}$ over $\underline{Z}$ is constructed by an initial $(\mathrm{V}+\Sigma)+\underline{Z}$-algebra.

We repeat the discussion of interpretation of rewriting.

Definition 9.1. The interpretation of a meta-term $Z \rhd n \vdash t$ in an $(\mathrm{V}+\Sigma)$-algebra $(A, [\nu, \alpha])$ by an assignment $\phi : Z \to A$ is denoted by $\phi^\sharp_n(t)$, which is an element of $A(n)$, defined by

$$\begin{aligned} \phi^\sharp_n(i) &= \nu_n(i) \\ \phi^\sharp_n(f(n + \overrightarrow{i_1}.t_1, \ldots, n + \overrightarrow{i_l}.t_l)) &= f^A_n(\phi^\sharp_{n+i_1}(t_1), \ldots, \phi^\sharp_{n+i_l}(t_l)) \\ \phi^\sharp_n(M[n_1, \ldots, n_m]) &= A(\xi)(\phi_m(M)), \end{aligned}$$

where $M \in Z(m)$, a map $\xi : m \to n$ in $\mathbb{F}$ is defined by $\xi(i) \stackrel{def}{=} n_i$.

Definition 9.2. Let $(\Sigma, \mathcal{C}, Z)$ be a binding CS. A monotone $(\mathrm{V}+\Sigma)$-algebra $(A, >_A)$ *satisfies* a rewrite rule $Z \rhd 0 \vdash \ell \Rightarrow r$ if

$$\phi^\sharp_0(\ell) >_{A(n)} \phi^\sharp_0(r)$$

for all assignments $\phi : Z \longrightarrow \delta^n A$.

Definition 9.3. A $(\Sigma, \mathcal{C}, Z)$-*algebra* A is a monotone $(\mathrm{V}+\Sigma)$-algebra A that satisfies all rules of a CS $(\Sigma, \mathcal{C}, Z)$.

Notice that a binding CS is a CS built only from binding meta-terms. We define the meta-rewriting on binding meta-terms by the inference system consisting of

$$\text{(Rule)}\ \frac{\begin{array}{l} s_j \in \mathrm{B}_\Sigma \underline{Z}(n + i_j) \quad (1 \le j \le m) \qquad \theta = [M_1 \mapsto s_1, \ldots, M_m \mapsto s_m] \\ (M_1^{(i_1)}, \ldots, M_m^{(i_m)}) \rhd 0 \vdash \ell \Rightarrow r) \in \mathcal{C} \end{array}}{Z \rhd n \vdash \theta^\sharp(\ell) \rightsquigarrow_{\mathcal{C}} \theta^\sharp(r)}$$

and (Fun) of Figure 4, where $\Rightarrow_{\mathcal{C}}$ is replaced with $\rightsquigarrow_{\mathcal{C}}$. It satisfies $\rightsquigarrow_{\mathcal{C}} = \Rightarrow_{\mathcal{C}} \cap \bigcup_{n \in \mathbb{N}} (\mathrm{B}_\Sigma \underline{Z} \times \mathrm{B}_\Sigma \underline{Z})(n)$.

Proposition 9.4. Let $(\Sigma, \mathcal{C}, Z)$ be a binding CS. For any $(\Sigma, \mathcal{C}, Z)$-algebra (A, α) and any assignment $\phi : Z \to A$,

$$Z \rhd n \vdash s \rightsquigarrow_{\mathcal{C}} t \ \Rightarrow\ \phi^\sharp_n(s) >_{A(n)} \phi^\sharp_n(t)$$

Proof. By induction on the proof of $s \rightsquigarrow_{\mathcal{C}} t$.

- Case (Rule). Suppose that

  $$Z \rhd n \vdash \theta^\sharp_0(\ell) \rightsquigarrow_{\mathcal{C}} \theta^\sharp_0(r)$$

  is derived from a rule $(Z \rhd 0 \vdash \ell \Rightarrow r) \in \mathcal{C}$ with $\theta : Z \to \delta^n \mathrm{B}_\Sigma \underline{Z}$. Let $\phi : Z \to A$ be an assignment. Take an assignment ψ as

  $$\psi = \underline{Z} \xrightarrow{\theta} \delta^n \mathrm{B}_\Sigma \underline{Z} \xrightarrow{\delta^n \phi^\sharp} \delta^n A$$

  By Lemma 6.9, $\psi^\sharp = (\delta^n \phi)^\sharp \circ \theta^\sharp$. Since A is a $(\Sigma, \mathcal{C}, Z)$-algebra, so is $\delta^n A$. Therefore,

  $$\phi^\sharp_n \theta^\sharp_0(\ell) = \psi^\sharp_0(\ell) >_{A(n)} \psi^\sharp_0(r) = \phi^\sharp_n \theta^\sharp(r)$$

- Case (Fun). Similarity to the case (Fun) in the proof of Proposition 5.9.

□

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

Theorem 9.5. A binding CS $(\Sigma, \mathcal{C}, Z)$ is meta-terminating on all binding meta-terms if and only if there is a well-founded $(\Sigma, \mathcal{C}, Z)$-algebra.

Proof. ($\Leftarrow$): Let A be a well-founded $(\Sigma, \mathcal{C}, Z)$-algebra. Assume that $\mathcal{C}$ is not meta-terminating, that is, there exists an infinite meta-rewriting sequence

$$Z \triangleright n \vdash t_1 \rightsquigarrow_{\mathcal{C}} t_2 \rightsquigarrow_{\mathcal{C}} t_3 \rightsquigarrow_{\mathcal{C}} \cdots .$$

By Proposition 9.4, for any admissible assignment $\phi : Z \to A$,

$$\phi_n^\sharp(t_1) >_{A(n)} \phi_n^\sharp(t_2) >_{A(n)} \phi_n^\sharp(t_3) >_{A(n)} \cdots .$$

This contradicts well-foundedness of $>_A$.
($\Rightarrow$): When a CS $\mathcal{C}$ is meta-terminating, the $(\Sigma, \mathcal{C}, Z)$-algebra $(\mathrm{B}_\Sigma \underline{Z}, \rightsquigarrow_{\mathcal{C}})$ is a desired well-founded one because the relation $\rightsquigarrow_{\mathcal{C}}$ is well-founded. □

Corollary 9.6. Let $(\Sigma, \mathcal{C}, Z)$ be a binding CS. If there is a well-founded $(\Sigma, \mathcal{C}, Z)$-algebra, then $\mathcal{C}$ is terminating.

Proof. It is clear that the meta-termination of $\mathcal{C}$ on binding meta-terms implies the termination of $\mathcal{C}$ on terms because all terms are binding meta-terms. □

Hence, in the case of binding CSs this becomes an interesting termination proof method by interpretation because we do not need a monoid structure.

Example 9.7. (Prenex normal forms) We show the termination of the binding CS $(\Sigma, \mathcal{C}, Z)$ for conversion into prenex normal forms given in the introduction. Formally, it is given by the signature $\Sigma = \{\forall, \exists : \langle 1\rangle, \wedge, \vee : \langle 0, 0\rangle, \neg : \langle 0\rangle\}$ and the metavariable set $Z = \{P^{(0)}, Q^{(1)}\}$. The set $\mathcal{C}$ of rules in de Bruijn levels is obtained by just replacing the variable x with 1.

$$Z \triangleright 0 \vdash P \wedge \forall(1.Q[1]) \Rightarrow \forall(1.P \wedge Q[1]) \qquad Z \triangleright 0 \vdash \neg\forall(1.Q[1]) \Rightarrow \exists(1.\neg(Q[1]))$$
$$Z \triangleright 0 \vdash \forall(1.Q[1]) \wedge P \Rightarrow \forall(1.P \wedge Q[1]) \qquad Z \triangleright 0 \vdash \neg\exists(1.Q[1]) \Rightarrow \forall(1.\neg(Q[1]))$$

We use Theorem 9.5 to show termination. Take a $(\mathrm{V}+\Sigma)$-algebra K such that the carrier is $K(n) = \mathbb{N}$ with $>_{K(n)}$ by the standard order $>$ on $\mathbb{N}$ for all $n \in \mathbb{N}$, and the operations are given by

$$\nu_n^K(i) = 0 \qquad \wedge_n^K(x, y) = \vee_n^K(x, y) = 2x + 2y$$
$$\neg_n^K(x) = 2x \qquad \forall_n^K(x) = \exists_n^K(x) = x + 1.$$

All operations are monotone. We show that K satisfies the rules: take an assignment $\phi : X \to \delta^n K$ by $P \mapsto x \in \mathbb{N}$ and $Q \mapsto y \in \mathbb{N}$, then

$$\phi_0^\sharp(P \wedge \forall(1.Q[1])) = 2x + 2(y+1) >_{K(0)} (2x + 2y) + 1 = \phi_0^\sharp(\forall(1.P \wedge Q[1]))$$
$$\phi_0^\sharp(\neg\exists(1.Q[1])) = 2(y+1) >_{K(0)} 2y + 1 = \phi_0^\sharp(\forall(1.\neg(Q[1]))).$$

The interpretations of other rules are similarly calculated. Since $>_{K(n)} = >$ is well-founded, this shows K forms a well-founded $(\Sigma, \mathcal{C}, Z)$-algebra. Thus, the binding CS $\mathcal{C}$ is terminating on all terms by Corollary 9.6.

This interpretation is *much simpler* than the Σ-monoid of hereditary monotone functionals in Section 7. We merely use ordinary polynomials and did not need to use second-order polynomials as in Section 7.5. Namely, if a CS is a binding CS, we do not need functionals to interpret second-order function symbols such as $\forall, \exists, \overline{\lambda}, \lambda$.

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

Example 9.8. (CPS translation) The binding CS $\mathcal{S}$ for a CPS translation in Example 3.3

$$\begin{aligned}
\mathsf{CPS}(E_0) &\Rightarrow \lambda k.(\!|E_0|\!)\,\bar{}\,(\bar{\lambda} m.km)\\
(\!|V|\!) &\Rightarrow \bar{\lambda} k.k\,\bar{}\,V\\
(\!|\lambda x.E[x]|\!) &\Rightarrow \bar{\lambda} k.k\,\bar{}\,(\lambda x.\lambda k.(\!|E[x]|\!)\,\bar{}\,(\bar{\lambda} m.km))\\
(\!|E_0E_1|\!) &\Rightarrow \bar{\lambda} k.(\!|E_0|\!)\,\bar{}\,(\bar{\lambda} m.(\!|E_1|\!)\,\bar{}\,(\bar{\lambda} n.mn(\lambda a.k\,\bar{}\,a)))
\end{aligned}$$

is shown to be terminating by the following polynomial interpretation: take a (V+Σ)-algebra K where the carrier $K(n) = \mathbb{N}$, the unit $\nu_n^K(x) = 0$, and the operations:

$$\mathsf{CPS}_n^K(e) = 5e + 5 \quad (\!|e|\!)_n^K = 5e + 1 \quad \bar{\lambda}_n^K(e) = e \quad \lambda_n^K(e) = e + 1$$
$$(e_0\,\bar{}\,e_1)_n^K = e_0 + e_1 \quad (e_0\,e_1)_n^K = e_0 + e_1 + 1.$$

The (V+Σ)-algebra K satisfies the rules. We take an assignment $\phi : Z \to \delta^n K$ by $E \mapsto e \in \mathbb{N}$, $V \mapsto v \in \mathbb{N}$, $E_0 \mapsto e_0 \in \mathbb{N}$, $E_0 \mapsto e_0 \in \mathbb{N}$. Then all rules are decreasing. For instance, the interpretation of the last rule is decreasing as

$$\begin{aligned}
\phi_0^\sharp((\!|E_0E_1|\!)) &= 5(e_0 + e_1 + 1) + 1 > 5e_0 + 1 + (5e_1 + 1 + 3)\\
&= \phi_0^\sharp(\bar{\lambda} k.(\!|E_0|\!)\,\bar{}\,(\bar{\lambda} m.(\!|E_1|\!)\,\bar{}\,(\bar{\lambda} n.mn(\lambda a.k\,\bar{}\,a)))).
\end{aligned}$$

Hence $\mathcal{C}$ is terminating by Corollary 9.6.

Example 9.9. (A theory of π-calculus) As the final example, we consider an example taken from a theory of π-calculus given by Stark. The π-calculus of Milner is one of the most fundamental concurrent calculi (Milner, 1999). Stark gave a free algebra model of π-calculus (Stark, 2008). A theory of π-calculus consists of 12 axioms. In Hamana (2019), we have analysed that the theory should be partitioned into rewrite rules and equations. A reason of doing this is that commutativity axioms for the sum and the new name generation operators cannot be oriented. In this example, we consider the termination of the rule part of the theory of π-calculus. We omit writing contexts. The signature Σ is

$$\mathsf{nil} : \langle\rangle \qquad \mathsf{in} : \langle 0, 1\rangle \qquad \mathsf{tau} : \langle 0\rangle \qquad \mathsf{sum} : \langle 0, 0\rangle \qquad \mathsf{out} : \langle 0, 0, 0\rangle \qquad \mathsf{new} : \langle 1\rangle$$

and the set $\mathcal{C}$ of rules is given by

$$\begin{aligned}
&\mathsf{new}(a.X) && \Rightarrow X\\
&\mathsf{sum}(\mathsf{nil}, X) && \Rightarrow X\\
&\mathsf{new}(a.\mathsf{sum}(X[a], Y[a])) && \Rightarrow \mathsf{sum}(\mathsf{new}(a.X[a]), \mathsf{new}(a.Y[a]))\\
&\mathsf{new}(a.\mathsf{out}(a, B, X[a])) && \Rightarrow \mathsf{nil}\\
&\mathsf{new}(a.\mathsf{out}(B, C, X[a])) && \Rightarrow \mathsf{out}(B, C, \mathsf{new}(a.X[a]))\\
&\mathsf{new}(a.\mathsf{in}(B, c.X[a, c])) && \Rightarrow \mathsf{in}(B, c.\mathsf{new}(a.X[a, c]))\\
&\mathsf{new}(a.\mathsf{tau}(X[a])) && \Rightarrow \mathsf{tau}(\mathsf{new}(a.X[a]))\\
&\mathsf{new}(a.\mathsf{in}(a, b.X[a, b])) && \Rightarrow \mathsf{nil}
\end{aligned}$$

This is a binding CS. Now we take the (V+Σ)-algebra K where the carrier $K(n) = \mathbb{N}$ and the operations:

$$\nu_n^K(x) = 0 \quad \mathsf{new}(x) = 2x + 1 \quad \mathsf{out}(x, y, z) = 2x + 2y + 2z + 2$$
$$\mathsf{tau}(x) = 2x + 2 \quad \mathsf{in}(x, y) = 2x + 2y + 2 \quad \mathsf{sum}(x, y) = 2x + 2y + 4 \quad \mathsf{nil} = 1$$

The (V+Σ)-algebra K satisfies the rules. We take an assignment $\phi : Z \to \delta^n K$ that assigns $x, y, b, c \in \mathbb{N}$ to metavariables X, Y, B, C, respectively. Then all rules are decreasing. For instance,

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

the interpretation of the third rule is decreasing as

$$\phi^{\sharp}_n(\mathsf{new}(a.\mathsf{sum}(X[a], Y[a]))) = 2(2x + 2y + 4) + 1 \;>\; 2(2x + 1) + 2(2y + 1) + 4 = \phi^{\sharp}_n(\mathsf{sum}(\mathsf{new}(a.X[a]), \mathsf{new}(a.Y[a])))$$

Hence $\mathcal{C}$ is terminating by Corollary 9.6.

10. Summary

Using the algebraic structures in a presheaf category over finite sets by Fiore, Plotkin and Turi, we have developed sound and complete models of second-order rewriting systems called second-order CSs. Restricting the algebraic structures to those equipped with well-founded relations, we have obtained complete characterisations of terminating CSs. We have also extended the characterisation to rewriting on meta-terms using the notion of Σ-monoids. A known model of higher-order rewrite rules given by hereditary monotone functionals has been shown to be an instance of Σ-monoids. Moreover, we have also shown that binding CSs have been modelled using simpler algebraic structures, which also simplified the model-based termination proof.

Acknowledgements. I am grateful to Gordon Plotkin, John Power, Neil Ghani and Marcelo Fiore for encouragement, suggestions and discussions around related topics. I also thank the reviewers for their positive and constructive comments. This work was supported in part by JSPS KAKENHI Grant Number 20H04164.

Notes

1 It is finitary, hence the initial algebra exists (Adamek, 1974).

2 A signature functor has a canonical strength $st : \Sigma(A) \bullet A \to \Sigma(A \bullet A)$ (Tanaka and Power, 2006, Theorem 6.3) (Fiore, 2008, Corollary 7).

References

Aczel, P. (1978). *A General Church-Rosser Theorem*. Technical report, University of Manchester.

Adamek, J. (1974). Free algebras and automata realizations in the language of categories. *Commentationes Mathematicae Universitatis Carolinae* **15** (589602).

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

Blanqui, F. (2016). Termination of rewrite relations on λ-terms based on Girard's notion of reducibility. *Theoretical Computer Science* **611** 50–86.

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

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

Cohn, P. (1965). *Universal Algebra*. Harper & Row.

Danvy, O. and Rose, K. (1998). Higher-order rewriting and partial evaluation. In: *Rewriting Techniques and Applications, 9th International Conference, (RTA'98)*, LNCS, vol. 1379.

de Bruijn, N. (1972). Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae* **34** 381–391.

Despeyroux, J., Felty, A. and Hirschowitz, A. (1995). Higher-order abstract syntax in Coq. In: *Typed Lambda Calculi and Applications*, LNCS, vol. 902, 124–138.

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

Fiore, M. (2008). Second-order and dependently-sorted abstract syntax. In: *LICS'08*, 57–68.

Fiore, M. and Hamana, M. (2013). Multiversal polymorphic algebraic theories: Syntax, semantics, translations, and equational logic. In: *28th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS, vol. 2013, 520–529.

Fiore, M. and Hur, C.-K. (2010). Second-order equational logic. In: *CSL'10*, LNCS, vol. 6247, 320–335.

Fiore, M. and Mahmoud, O. (2010). Second-order algebraic theories. In: *MFCS'10*, LNCS, vol. 6281, 368–380.

Fiore, M., Plotkin, G. and Turi, D. (1999). Abstract syntax and variable binding. In: *Proceedings of 14th Annual Symposium on Logic in Computer Science*, 193–202.

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press

Gandy, R. (1980). Proofs of strong normalization. In: Seldin, J. P. and Hindley, J. R. (eds.) *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press Limited.
Hamana, M. (2003). Term rewriting with variable binding: An initial algebra approach. In: *Fifth ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP'03)*, 148–159.
Hamana, M. (2004). Free Σ-monoids: A higher-order syntax with metavariables. In: *Asian Symposium on Programming Languages and Systems (APLAS 2004)*, LNCS, vol. 3302, 348–363.
Hamana, M. (2005). Universal algebra for termination of higher-order rewriting. In: *RTA'05*, LNCS, vol. 3467, 135–149.
Hamana, M. (2007). Higher-order semantic labelling for inductive datatype systems. In: *PPDP'07*, 97–108.
Hamana, M. (2011). Polymorphic abstract syntax via Grothendieck construction. In *FoSSaCS'11*, LNCS, vol. 3467, 381–395.
Hamana, M. (2017). How to prove your calculus is decidable: Practical applications of second-order algebraic theories and computation. *Proceedings of the ACM on Programming Languages* **1** (22) 1–28.
Hamana, M. (2019). How to prove decidability of equational theories with second-order computation analyser SOL. *Journal of Functional Programming* **29** (e20).
Hamana, M., Abe, T. and Kikuchi, K. (2020). Polymorphic computation systems: Theory and practice of confluence with call-by-value. *Science of Computer Programming* **187** (102322).
Huet, G. and Lankford, D. S. (1978). On the uniform halting problem for term rewriting systems. Technical Report Rapport Laboria 283, INRIA.
Jouannaud, J.-P. and Rubio, A. (2001). Higher-order recursive path orderings à la carte. In: *International Workshop on Rewriting in Proof and Computation (RPC'01)*, 161–175.
Klop, J. (1980). *Combinatory Reduction Systems*. PhD thesis, CWI, Amsterdam, vol. 127. Mathematical Centre Tracts.
Lescanne, P. and Rouyer-Degli, J. (1995). Explicit substitutions with de Bruijn's levels. In: *Rewriting Techniques and Applications, 6th International Conference (RTA-95)*, LNCS, vol. 914. Springer, 294–308.
Libal, T. and Miller, D. (2016). Functions-as-constructors higher-order unification. In: *Proceedings of FSCD 2016*, vol. 52. *LIPIcs*, 26:1–26:17.
Mac Lane, S. (1971). *Categories for the Working Mathematician*, vol. 5. Graduate Texts in Mathematics. Springer-Verlag, New York.
Mayr, R. and Nipkow, T. (1998). Higher-order rewrite systems and their confluence. *Theoretical Computer Science* **192** (1) 3–29.
Miculan, M. and Scagnetto, I. (2003). A framework for typed HOAS and semantics. In: *Proceedings of PPDP'03*. ACM Press, 184–194.
Miller, D. (1991). A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation* **1** (4) 497–536.
Milner, R. (1999). *Communicating and Mobile Systems - The π-Calculus*. CUP.
Pfenning, F. and Elliott, C. (1988). Higher-order abstract syntax. In: *Proceedings of the ACM SIGPLAN '88 Symposium on Language Design and Implementation*, 199–208.
Plotkin, G. (1998). Binding algebras: A step between universal algebra and type theory (invited talk). In: *Rewriting Techniques and Applications, 9th International Conference, RTA'98, Tsukuba, Japan*.
Stark, I. (2008). Free-algebra models for the π-calculus. *Theoretical Computer Science* **390** (2–3) 248–270.
Tanaka, M. and Power, J. (2006). A unified category-theoretic semantics for binding signatures in substructural logics. *Journal of Logic and Computation* **16** (1) 5–25.
Taylor, W. (1993). Abstract clone theory. *Algebras and Orders* **389** 507–530. NATO ASI Series C.
Turi, D. and Plotkin, G. (1997). Towards a mathematical operational semantics. In: *Proceedings of the Twelfth Annual IEEE Symposium on Logic in Computer Science, LICS '97*, 280–291.
van de Pol, J. (1994). Termination proofs for higher-order rewrite systems. In: *The First International Workshop on Higher-Order Algebra, Logic and Term Rewriting (HOA'93)*, LNCS, vol. 816, 305–325.
van de Pol, J. (1996). *Termination of Higher-order Rewrite Systems*. PhD thesis, Universiteit Utrecht. https://cs.au.dk/jaco/papers/thesis.pdf
Yokoyama, T., Hu, Z. and Takeichi, M. (2003). Deterministic higher-order patterns for program transformation. In: *Logic Based Program Synthesis and Transformation, 13th International Symposium LOPSTR 2003, Uppsala, Sweden, August 25–27, 2003, Revised Selected Papers*, 128–142.
Yokoyama, T., Hu, Z. and Takeichi, M. (2004). Deterministic second-order patterns. *Information Processing Letters* **89** (6) 309–314.
Zantema, H. (1994). Termination of term rewriting: Interpretation and type elimination. *Journal of Symbolic Computation* **17** 23–50.

Cite this article: Hamana M (2022). Complete algebraic semantics for second-order rewriting systems based on abstract syntax with variable binding. *Mathematical Structures in Computer Science* **32**, 542–573. https://doi.org/10.1017/S0960129522000287

https://doi.org/10.1017/S0960129522000287 Published online by Cambridge University Press