NON-DETERMINISTIC ALGEBRAIC REWRITING AS ADJUNCTION

RĂZVAN DIACONESCU

ABSTRACT. We develop a general model theoretic semantics to rewriting beyond the usual confluence and termination assumptions. This is based on preordered algebra which is a model theory that extends many sorted algebra. In this framework we characterise rewriting in arbitrary algebras rather than term algebras (called algebraic rewriting) as a persistent adjunction and use this result, on the one hand for proving the soundness and the completeness of an abstract computational model of rewriting that underlies the non-deterministic programming with Maude and CafeOBJ, and on the other hand for developing a compositionality result for algebraic rewriting in the context of the pushout-based modularisation technique.

1. INTRODUCTION

Term rewriting is a computational paradigm that plays a major role in functional programming, theorem proving or formal verification. We may distinguish between two kinds of term rewriting:

- Type-theoretic term rewriting that applies to terms of some higher-order logical system, such as \( \lambda \)-calculus. A number of functional programming languages and computer systems that implement some form of higher-order logic or type theory implement this kind of term rewriting, such as Haskell \([24]\), ML \([27]\), Coq \([9, 2]\), etc.

- Algebraic term rewriting that applies to terms in some form of model theory, traditionally some variant of universal algebra. This includes also particular important contexts such as group theory or ring theory where algebraic rewriting plays a crucial role in their computational sides. In computing science algebraic term rewriting constitutes the computational basis of the execution engines of several algebraic specification and programming systems such as OBJ \([21]\), ASF+SDF \([30]\), Elan \([3]\), CafeOBJ \([15]\), Maude \([5]\), etc. While algebraic term rewriting is commonly used as a decision procedure for equational logic (by assuming confluence and termination) \([1]\), languages and systems such as Maude or CafeOBJ also implement algebraic term rewriting in a wider non-deterministic sense, its semantic interpretation going beyond universal algebra and its associated equational logic.

Our work is concerned with the latter kind of term rewriting in its wider meaning where confluence and even termination are not assumed. The semantic implication of this is that rather than collapsing the intermediate states of the rewriting process to the final result (like in the case of the traditional approach to rewriting as a decision procedure for equational logic) we consider them as semantic entities with separate identities.

There is yet another dimension of generality to our approach to rewriting. In our work we consider rewriting in arbitrary algebras (hence the terminology ‘algebraic rewriting’),
ordinary term rewriting corresponding just to the particular case when the algebra under
corresponding just to the particular case when the algebra under
corresponding just to the particular case when the algebra under
corresponding just to the particular case when the algebra under
consideration is the term algebra. Apart of ordinary term rewriting there are several other
interpretations of (general) algebraic rewriting. One of them concerns the practically im-
portant case of rewriting modulo axioms, the corresponding algebra being the free algebra
of an equational theory. Another important application of algebraic rewriting is that of
computation with predefined types, when the corresponding algebra is not user specified
but is rather provided by the respective computational system.

Motivation. As a programming and specification paradigm, non-deterministic program-
ming is a powerful one even in a simple form that does not involve additional fancy fea-
tures. But like with any other advanced programming paradigms its effectiveness depends
on methodologies and on a proper understanding of the paradigm from the side of the
users. Semantics can play a crucial role in all these, but only when it is simple enough,
clean, and it enjoys those mathematical properties that support several important general
techniques such as a good modular development. Our work is motivated by this and builds
on the logic\(^1\) of preordered algebra (abbreviated \(POA\)) \([16, 13, 12, 7, 8]\) that has been
established as the logical semantics of non-deterministic rewriting in CafeOBJ \([16]\) and
Maude \([7, 8]\). Here we refine the definition of \(POA\) such that it captures more accurately
programming language aspects and develop a series of new concepts and results regard-
ing non-deterministic rewriting in \(POA\) in order to refine the mathematical foundations of
CafeOBJ and Maude supporting a deeper understanding of how to use them. Moreover,
some novelties in our work may open the door for new methodologies for non-deterministic
rewriting. We hope this will increase the attractiveness of this paradigm both for education
and software development.

1.1. Our main contributions.

(1) A model theoretic semantics to rewriting that has the following characteristics (not nec-
essarily ordered by their importance):

- Mathematical effectiveness. It is rigorously based on a refinement of the \(POA\) of
[16, 13, 12] which considers two types of sorts (rather than a single one) because
this fits more accurately the applications. This upgraded version of \(POA\) remains
an institution \([18, 12]\), and this has important consequences, such as the availability
of a highly developed model theory at the level of abstract institutions \([12]\) and of
a matured specification and programming theory that has become a foundational
standard when defining formal specification and logic-based languages.
- The relationship between the model theoretic semantics and the computational model
that we establish here is a \textit{direct} one, without the mediation of a system of logical
deduction (equational logic for deterministic term rewriting or rewriting logic in the
case of non-deterministic term rewriting).
- The models are considered \textit{loose} rather than tight, term models being just one ex-
ample of our theory. All interpretations of algebraic rewriting, including those men-
tioned above, can be thus treated in an uniform and unitary way.

The latter two characteristics represent important novelties with respect to previous
\(POA\) semantics to rewriting.

\(^1\)This means mathematically defined syntax, semantics and proof theory, as put forward in \([28]\).
A categorical adjunction characterisation of algebraic rewriting, which can be considered as an axiomatic view on rewriting. This core result is instrumental for subsequent developments in the paper.

An abstract rewriting-based computational model\(^2\) that allows for the integration of deterministic with non-deterministic rewriting at different layers.

General results about the soundness and the completeness of algebraic rewriting with respect to the proposed model theoretic semantics. These properties are best explained as an isomorphism relationship between the models of the denotational and of the operational semantics. An immediate application of these general results is a genuine model theoretic argument for the correctness of the executions of Maude and CafeOBJ system modules.

A theorem on the compositionality of algebraic rewriting. This result provides a model theoretic foundations for the structuring of non-deterministic algebraic rewriting through the pushout-based technique.

1.2. The structure of the paper.

In a section on preliminaries we review some basic notations and concepts about sets, relations and provide a presentation of the basic concepts from many-sorted algebra.

We introduce our model theoretic framework for rewriting, the (many-sorted) preorder algebra in the enhanced form mentioned above.

We develop the result that shows that the algebraic rewriting relations can be defined through an adjunction between the category of many sorted algebras and the category of preordered algebras satisfying a set of Horn sentences.

We define the abstract operational model for algebraic rewriting and we provide sufficient conditions for its soundness and completeness with respect to the denotational model provided by the adjunction result.

The theorem on the compositionality of algebraic rewriting.

2. Preliminaries

2.1. Relations. A binary relation is a subset \( R \subseteq A \times B \), where \( A \) and \( B \) are sets. When \( A = B \) we say that \( R \) is a binary relation on \( A \). Sometimes \( (a, b) \in R \) may be denoted as \( a \; R \; b \). A binary relation \( R \) on a set \( A \) is

- reflexive, when for each \( a \in A \), \( (a, a) \in R \),
- symmetric, when for each \( a, b \in A \), \( (a, b) \in R \) implies \( (b, a) \in R \),
- anti-symmetric, when for each \( a, b \in A \), \( (a, b), (b, a) \in R \) implies \( a = b \),
- transitive, when for each \( a, b, c \in A \), \( (a, b), (b, c) \in R \) implies \( (a, c) \in R \),
- preorder, when it is reflexive and transitive,
- partial order, when it is anti-symmetric preorder,
- equivalence, when it is symmetric and preorder.

Given an equivalence relation \( \sim \) on a set \( A \), for each \( a \in A \) by \( a/\sim \) we denote the equivalence class of \( a \) which is the set \( \{ b \in A \mid a \sim b \} \). Let \( A/\sim = \{ a/\sim \mid a \in A \} \) be the set of all equivalence classes in \( A \) determined by \( \sim \).

\(^2\)In the sense of model theory.
Given a relation $R$ let $R^*$ be its reflexive-transitive closure, which is the least reflexive and transitive relation containing $R$. Then $R^* = \bigcup_{n\in\omega} R^n$ where $R^n = \{(a_0, a_n) \mid \exists(a_i, a_{i+1}) \in R, \ 0 \leq i \leq n - 1\}$. For a binary relation $\rightarrow$ by $\rightarrow^*$ we mean $\rightarrow^*$.

2.2. Many-sorted Algebra. Many-sorted algebra (MSA) has traditionally played a major role in the semantics of computation. For instance MSA is the core formalism in the area of algebraic specification. Since OBJ [21], due to algebraic term rewriting, MSA has been turned into a functional programming paradigm (called equational programming). Since the model theoretic framework of our work is built on top of MSA, in what follows we provide an overview of the main MSA concepts.

2.2.1. Signatures. We let $S^*$ denote the set of all finite sequences of elements from $S$, with $[]$ the empty sequence. A(n S-sorted) signature $(S, F)$ is an $S^* \times S$-indexed set $F = \{F_w \mid w \in S^*, \ s \in S\}$ of operation symbols. The sets $F_w$ in the definition above stand for the sets of symbols with arity $w$ and sort $s$. We may denote $F_w$ simply as $F_w$. Note that this definition permits overloading, in that the sets $F_w$ need not be disjoint. When this leads to parsing ambiguities, the qualification of the operation symbols by their rank is a good solution. Then $\sigma \in F_w$ will be denoted as $\sigma : w \rightarrow s$.

2.2.2. Terms. An $(S, F)$-term $t$ of sort $s \in S$, is an expression of the form $\sigma(t_1, \ldots, t_n)$, where $\sigma \in F_{s_1, \ldots, s_n}$, and $t_1, \ldots, t_n$ are $(S, F)$-terms of sorts $s_1, \ldots, s_n$, respectively. The set of the $(S, F)$-terms of sorts $s$ is denoted by $(T_{(S,F)})_s$ while $T_{(S,F)}$ denotes the set of all $(S, F)$-terms.

2.2.3. Variables. Let $(S, F)$ be a signature. A variable for $(S, F)$ is a pair $(x, s)$ where $x$ is the name of the variable and $s \in S$ is the sort of the variable, such that $(x, s) \notin F_w$. The sort is an essential qualification for a variable. However, when this is clear, we may simply refer to a variable by its name only. For example, if $X$ is a set of variables for $(S, F)$, then $(x, s) \in X$ may be denoted $x \in X$. For this to make sense, but also in order to avoid other kinds of clashes, we make the basic assumption valid all over our material, that when considering sets of variables, any two different variables have different names.

For any signature $(S, F)$ and any set $X$ of variables for $(S, F)$, the signature $(S, F \cup X)$ denotes the extension of $(S, F)$ with $X$ as new constants that respects the sorts of the variables. This means $(F \cup X)_{w \rightarrow s} = F_w \cup \{(x, s) \mid (x, s) \in X\}$.4

2.2.4. Substitutions. Given sets $X$ and $Y$ of variables for a signature $(S, F)$, an $(S, F)$-substitution $\theta$ from $X$ to $Y$ is a function $\theta : X \rightarrow T_{(S,F \cup Y)}$ that respects the sorts, i.e. if $x$ has sort $s$ then $\theta(x) \in (T_{(S,F \cup Y)})_s$. The existence of substitutions from $X$ to $Y$ requires that whenever there is a variable in $X$ of sort $s$ then $(T_{(S,F \cup Y)})_s$ is non-empty. In general, this condition can be met if we assume that the signatures contain at least one constant for each sort. Any substitution $\theta : X \rightarrow T_{(S,F \cup Y)}$ extends to a function

3This is also the solution adopted in the implementation of algebraic specification languages.

4Note that this union is always disjoint because $X$ are new constants.
\[ \theta^t : T_{(S, F, \cup X)} \to T_{(S, F, \cup Y)} \] defined by

\[
\theta^t(t) = \begin{cases} \theta(x) & \text{when } t = x \text{ for } x \in X \\ \sigma(\theta^t(t_1), \ldots, \theta^t(t_n)) & \text{when } t = \sigma(t_1, \ldots, t_n) \text{ with } \sigma \in F. \end{cases}
\]

When there is no danger of notational confusion we may omit ‘\(\star\)’ from the notation and write simply \(\theta(t)\) instead of \(\theta^t(t)\).

2.2.5. Sentences. Given a signature \((S, F)\), an atomic equation is a pair \((t, t')\) of \((S, F)\)-terms of the same sort. Often we write atomic equations as \(t = t'\) rather than \((t, t')\). The set of the \((S, F)\)-sentences is the least set such that:

- Each atomic \((S, F)\)-equation is an \((S, F)\)-sentence.
- If \(\rho_1\) and \(\rho_2\) are \((S, F)\)-sentences then \(\rho_1 \land \rho_2\) (conjunction), \(\rho_1 \lor \rho_2\) (disjunction), \(\rho_1 \Rightarrow \rho_2\) (implication) and \(\neg \rho_1\) (negation) are also \((S, F)\)-sentences.
- If \(X\) is a set of variables for \((S, F)\), then \(\forall X \cdot \rho\) and \(\exists X \cdot \rho\) are \((S, F)\)-sentences whenever \(\rho\) is an \((S, F \cup X)\)-sentence.

The sentences that do not involve any quantifications are called quantifier-free sentences.

A conditional equation is a sentence of the form
\[ \forall X \cdot H \Rightarrow C \]
where \(H\) is a finite conjunction of (atomic) equations and \(C\) is a single (atomic) equation.

When \(H\) is empty the respective conditional equation is usually written simply as \(\forall X \cdot C\) and is called unconditional equation. In this paper ‘conditional equations’ will be called simply ‘equations’ and when the equations are unconditional we will explicitly say ‘unconditional equations’.

2.2.6. Algebras. Given a set of sort symbols \(S\), an \(S\)-indexed (or sorted) set \(A\) is a family \((A_s)_{s \in S}\) of sets indexed by the elements of \(S\); in this context, \(a \in A\) means that \(a \in A_s\) for some \(s \in S\). Given an \(S\)-indexed set \(A\) and \(w = s_1 \ldots s_n \in S^n\), we let \(A_w = A_{s_1} \times \cdots \times A_{s_n}\); in particular, we let \(A_[] = \{\ast\}\), some one point set.

Given a signature \((S, F)\), a \((S, F)\)-algebra \(A\) consists of

- an \(S\)-indexed set \(A\) (the set \(A_s\) is called the carrier of \(A\) of sort \(s\)), and
- a function \(A_{\sigma,w \rightarrow s} : A_w \rightarrow A_s\) for each \(\sigma \in F_{w \rightarrow s}\).

When there is no danger of ambiguity (because of overloading of \(\sigma\)) we may simplify the notation \(A_{\sigma,w \rightarrow s}\) to \(A_{\sigma}\). If \(\sigma \in F_{w \rightarrow s}\) then \(A_{\sigma}\) determines an element in \(A_s\) which may also be denoted \(A_{\sigma}\).

Any \((S, F)\)-term \(t = \sigma(t_1, \ldots, t_n)\), where \(\sigma\) is an operation symbol of \((S, F)\), gets interpreted as an element \(A_t \in A_s\) in a \((S, F)\)-algebra \(A\) defined by
\[ A_t = A_{\sigma}(A_{t_1}, \ldots, A_{t_n}). \]

2.2.7. Term algebras. Given a signature \((S, F)\), the \((S, F)\)-term algebra, denoted \(0_{(S, F)}\), interprets any sort symbol \(s \in S\) as the set of the \((S, F)\)-terms of sort \(s\), and each operation symbol \(\sigma : w \rightarrow s\) as
\[ (0_{(S, F)})_{\sigma}(t_1, \ldots, t_n) = \sigma(t_1, \ldots, t_n). \]
2.2.8. Homomorphisms. An \( S \)-indexed (or sorted) function \( f: A \to B \) is a family \( \{ f_s: A_s \to B_s \mid s \in S \} \). Also, for an \( S \)-sorted function \( f: A \to B \), we let \( f_w: A_w \to B_w \) denote the function product mapping a tuple of elements \((a_1, \ldots, a_n)\) to the tuple \((f_{s_1}(a_1), \ldots, f_{s_n}(a_n))\).

Given \((S, F)\)-algebras \( A \) and \( B \), an \((S, F)\)-homomorphism from \( A \) to \( B \) is an \( S \)-indexed function \( h: A \to B \) such that

\[
h_s(A_\sigma(a)) = B_\sigma(h_w(a))
\]

for each \( \sigma \in F_{w \to s} \) and \( a \in A_w \). When there is no danger of confusion we may simply write \( h(a) \) instead of \( h_s(a) \).

Homomorphisms preserve the interpretations of terms, which is a useful technical property:

**Lemma 2.1.** Let \( h : A \to B \) be an \((S, F)\)-algebra homomorphism and let \( t \) be any \((S, F)\)-term. Then \( h(A_t) = B_t \).

Given \((S,F)\)-homomorphisms \( h : A \to B \) and \( g : B \to C \), their composition \( h; g \) is the algebra homomorphism \( A \to C \) defined by \((h; g)_s = g_w \circ h_s\) for each sort symbol \( s \in S \).

2.2.9. Isomorphisms. An \((S,F)\)-homomorphism \( h : A \to B \) is a \((S,F)\)-isomorphism when there exists another homomorphism \( h^{-1} : B \to A \) such that \( h; h^{-1} = 1_A \) and \( h^{-1}; h = 1_B \), where by \( 1_A : A \to A \) and \( 1_B : B \to B \) we denote the ‘identity’ homomorphisms that map each element to itself.

**Fact 1.** A \((S,F)\)-homomorphism \( h : A \to B \) is isomorphism if and only if each function \( h_s : A_s \to B_s \) is bijective (i.e., one-to-one and onto, in an older terminology).

2.2.10. Initial algebras. Given any class \( C \) of \((S,F)\)-algebras, an algebra \( A \) is initial for \( C \) when \( A \in C \) and for each algebra \( B \in C \) there exists an unique homomorphism \( f : A \to B \).

**Proposition 2.2.** If \( A \) and \( A' \) are both initial algebras for \( C \), then there exists an isomorphism \( h : A \to A' \).

The following is well known and also easy to establish.

**Proposition 2.3.** The term algebra \( 0_{(S,F)} \) is initial in the class of all \((S,F)\)-algebras.

2.2.11. Satisfaction. The satisfaction between \((S,F)\)-algebras and \((S,F)\)-sentences, denoted \( \models_{(S,F)} \) (or simply by \( \models \) when there is no danger of confusion), is defined inductively on the structure of the sentences as follows. This process can be regarded as an evaluation of the sentences to one of the truth values \( \text{true} \) or \( \text{false} \) and which is contingent on a given model/algebra.

Given a fixed arbitrary signature \((S,F)\) and a \((S,F)\)-algebra \( A \),

- \( A \models t = t' \) if and only if \( A_t = A_{t'} \) for atomic equations,
- \( A \models \rho_1 \land \rho_2 \) if and only if \( A \models \rho_1 \) and \( A \models \rho_2 \),
- \( A \models \rho_1 \lor \rho_2 \) if and only if \( A \models \rho_1 \) or \( A \models \rho_2 \),
- \( A \models \rho_1 \Rightarrow \rho_2 \) if and only if \( A \not\models \rho_1 \) or \( A \models \rho_2 \),
- \( A \models \neg \rho \) if and only if \( A \not\models \rho \),

\(^5\)This means \((h; g)(a) = g(h(a))\) for each \( a \in A_s \).
• for any set of variables $X$ for the signature $(S, F)$, and for any $(S, F \cup X)$-sentence $\rho$, $A \models_{(S, F)} \forall X \cdot \rho$ if and only if $A' \models_{(S, F \cup X)} \rho$ for each $(S, F \cup X)$-algebra $A'$ such that $A'_s = A_s$ for each $s \in S$ and $A'_o = A_o$ for each operation symbol $o$ of $F$.

• $A \models \exists X \cdot \rho$ if and only if $A \not\models \forall X \cdot \neg \rho$.

When $A \models \rho$ we say that $A$ satisfies $\rho$ or that $\rho$ holds in $A$.

2.2.12. Signature morphisms. Given two MSA signature $(S, F)$ and $(S', F')$, a signature morphism $\varphi : (S, F) \to (S', F')$ consists of a function $\varphi^\text{st} : S \to S'$ and a family of functions $\varphi^\text{op} = \{ \varphi^\text{op}_{w,s} : F_{w,s} \to F'_{\varphi^\text{st}(w),\varphi^\text{st}(s)} \mid w \in S^*, s \in S \}$. Signature morphisms compose component-wise; we skip the straightforward technical details here. The signature morphisms have all compositionality properties of the functions.

2.2.13. Sentence translations. Any MSA signature morphism $\varphi : (S, F) \to (S', F')$ induces a translation function from the set of the $(S, F)$-sentences to the set of the $(S', F')$-sentences. In brief, these translations just rename the sort and the function symbols according to the respective signature morphism. They can be formally defined by induction on the structure of the sentences. The details of this can be read from the literature (eg. [12]). Let us denote by $\varphi \rho$ the translation of an $(S, F)$-sentence $\rho$ along a signature morphism $\varphi$.

2.2.14. Model reducts. For each signature morphism $\varphi : (S, F) \to (S', F')$, the $\varphi$-reduct $\varphi A'$ of an $(S', F')$-algebra $A'$ is an $(S, F)$-algebra defined by $(\varphi A')_x = A'_\varphi x$ for each sort or function symbol $x$ from $(S, F)$. Conversely, $A'$ is called the $\varphi$-expansion of $A$. These concepts extend to model homomorphisms. Given an $(S', F')$-homomorphism $h' : A' \to B'$, its $\varphi$-reduct is the model homomorphism $h : \varphi A' \to \varphi B'$, denoted $\varphi h'$, defined for each sort $s \in S$ by $h_s = h'_\varphi s$.

2.2.15. Assignment evaluations. Given a signature $(S, F)$, a set $X$ of variables for $(S, F)$, an $(S, F)$-algebra $B$, and a mapping $\theta : X \to B$, for any $(S, F \cup X)$-term $t$ by $\theta t$ we designate $B'_t$ where $B'$ is the $(S, F \cup X)$-expansion of $B$ such that $B'_x = \theta x$ for each $x \in X$. When $B$ is the term algebra $0_{(S, F \cup Y)}$ then $\theta$ is a substitution $X \to Y$ and $\theta t$ is precisely the instance of $t$ by $\theta$ as defined above. Thus, in this way the mappings $\theta : X \to B$ can be regarded as a generalisation of the concept of substitution.

2.2.16. The Satisfaction Lemma. It says that:

**Theorem 2.4.** For each MSA signature morphism $\varphi : (S, F) \to (S', F')$, for each $(S, F)$-sentence $\rho$ and each $(S', F')$-algebra $A'$

$$A' \models_{(S', F')} \varphi \rho \text{ if and only if } \varphi A' \models_{(S, F)} \rho.$$ 

Proofs of this result can be found in several places in the literature, eg. [18, 12].

The capture of the MSA signatures and their morphisms as a category, of the collection of sets of sentences and their translations as a functor, of the collection of categories of models and their reducts as another functor, together with the Satisfaction Lemma, is what makes MSA an institution. As shown in the literature this has vast theoretical and practical consequences which we do not discuss here. In this work we do not involve any institution theory, however we want the reader to be aware in general of the importance of having model theoretic frameworks captured as institutions, and in particular of the importance of having the model-theoretic framework of our work captured as an institution.
3. Many-sorted Preordered Algebra

The structure of ‘preordered algebra’, has already been present in works regarding model theoretic semantics of transitions (e.g. [16, 13, 7, 8]) and has also been used frequently as one of the benchmark examples for institutional model theoretic developments [11, 10, 22, 6, 23]. Much of the model theory of POA can be found in [12]. Although it has its own specific computing science meaning, model theoretically POA is very much an extension of MSA. Here we upgrade and refine the established definitions of POA in order to make it more usable in the applications. The most important upgrade – that triggers also most of the other upgrades – is that we will make an explicit distinction between two kind of sorts, one for data types (like in MSA) and another kind for transitions. Only the latter sorts are interpreted as preorders. Although all developments of our paper can be done within the context of the simpler form of POA, only with the upgraded POA we will be able to match properly the mathematical theory to the specification examples.

3.1. POA definitions. They follow the structure of the MSA definitions and also include them. Therefore we will provide explicitly only the definitions of the parts that are specific to POA.

Definition 1 (POA signatures). A POA signature is a triple $(D, S, F)$ where $D, S$ are sets such that $D \cap S = \emptyset$ and $(D \cup S, F)$ is an MSA signature. The elements of $D$ are called data sorts while the elements of $S$ are called system sorts.

The POA concepts of term, variable, substitution, are all just the corresponding MSA concepts obtained by regarding the POA signatures $(D, S, F)$ as MSA signatures $(D \cup S, F)$.

Definition 2 (POA sentences). Given a POA signature $(D, S, F)$, the $(D, S, F)$-sentences are defined like MSA $(D \cup S, F)$-sentences with one important difference: instead of one kind of atoms like in MSA, in POA we have two kinds of atoms:

- MSA (atomic) equations $t = t'$, where $t, t'$ are $(D \cup S, F)$-terms of the same sort, and
- (atomic) transitions $t \rightarrow t'$ where $t, t'$ are $(D \cup S, F)$-terms of the same system sort.

A Horn $(D, S, F)$-sentence is a sentence of the form $\forall X \cdot H \Rightarrow C$ where $H$ is a finite conjunction of atoms and $C$ is a single atom. A $(D, S, F)$-equation is just a $(D \cup S, F)$-equation. A $(D, S, F)$-transition is any Horn $(D, S, F)$-sentence $\forall X \cdot H \Rightarrow (t \rightarrow t')$.

In some situations it is useful to be able to separate in the conditions $H$ the equations from the transitions.

Notation 1. Given a finite conjunction $H$ of atomic sentences we let

- $H_{=}$ be the set of the equations occurring in $H$, and
- $H_{\rightarrow}$ be the set of the transitions occurring in $H$.

Definition 3 (POA models). Given a POA signature $(D, S, F)$, a $(D, S, F)$-(preordered) algebra $(A, \preceq)$ consists of

- an MSA $(D \cup S, F)$-algebra $A$, and
- a family $\preceq = (\preceq_{s})_{s \in S}$ such that each $(A_{s}, \preceq_{s})$ is a preordered set such that for each $\sigma \in F_{w \rightarrow s}$ if $s \in S$ then $A_{\sigma}$ is monotone.
Definition 4 (POA model homomorphism). Given \((D, S, F)\)-algebras \((A, \leq)\) and \((B, \leq)\), a POA homomorphism \(h : (A, \leq) \to (B, \leq)\) is an MSA homomorphism \(h : A \to B\) that is monotone on the system sorts, i.e. \(a \leq a'\) implies \(ha \leq ha'\).

The POA concepts of isomorphism and initiality are like the respective MSA concepts. In fact both concepts are category-theoretic concepts, so their definition is independent of the actual model theoretic framework. However there is an important difference between MSA and POA with respect to model isomorphisms: in POA Fact 1 does not hold! The concept of POA model isomorphism is strictly stronger than that of bijective homomorphism, in other words a bijective POA homomorphism is not necessarily a POA isomorphism.

Example 1. Let us consider the signature with one sort and two constants \(a\) and \(b\), and on the one hand the POA algebra \(A\) consisting of two elements, \(A_a\) and \(A_b\) and an empty preorder relation, and on the other hand the POA algebra \(B\) consisting of two elements \(B_a\) and \(B_b\) such that \(B_a \leq B_b\). Then the unique homomorphism \(h : A \to B\) is bijective but its inverse as a function is not a POA homomorphism because it is not monotone.

Definition 5 (POA satisfaction). Given a POA signature \((D, S, F)\), the satisfaction between the \((D, S, F)\)-algebras and \((D, S, F)\)-sentences is obtained by extending the MSA satisfaction between \((D \cup S, F)\)-algebras and \((D \cup S, F)\)-sentences with the satisfaction of atomic transitions:

\[(A, \leq) |=_{(D, S, F)} t \to t' \text{ if and only if } A_t \leq A_{t'}\]

and by following the same inductive process on the structure of the sentences like in MSA.

We end this section with a couple of examples.

Example 2. The bubble sort algorithm for sorting lists of natural numbers admits a very compact and clear coding in POA as follows.\(^6\)

```maude
mod BUBBLE-SORT is
  protecting LIST{Nat} .
  vars m n : Nat .
  crl m n => n m if n < m .
endm
```

In this specification there is an import of data using the keyword \texttt{protecting}. The data is the lists of the natural numbers. This has two types:

1. The natural numbers \texttt{NAT}. There are two ways to consider this, either as user defined or as predefined. In the former approach \texttt{NAT} is an equational specification of the natural numbers as can be found in many places in the algebraic specification literature.

---

\(^6\)We use the Maude notations that are close to the common mathematical notations as follows. A module definition starts with its name (we use the keyword \texttt{mod} without respect of the semantic nature of the respective module), then we have data type import declarations, then sort symbols declarations (keyword \texttt{sort}), then operation symbols declarations (keyword \texttt{op}) that follow the usual functional notation from mathematics that was introduced by Euler, then variables declarations (keyword \texttt{var}) in which the sort of each variable is given. These give the specification of the signatures. By default convention a sort is a system sort if and only if there exist a transition with terms of that sort. The equations / transitions are specified by the keywords \texttt{eq/rcl} (or \texttt{ceq/crl} in the conditional case). \texttt{=>} is used for transitions. The conditions are given as Boolean terms after the keyword \texttt{if}. 

Although this is a nice example to illustrate concepts from the theory of the initial data type specification, the real world way is to consider \textit{NAT} as a predefined type. Practically this means that \textit{NAT} is already made available by the system. In the case of Maude and CafeOBJ it actually comes from the underlying implementation language, which is C ultimately. In terms of the mathematical semantics this means that we have a signature that consists of one sort \textit{Nat} and some usual operations on the natural numbers.\footnote{We take here a minimalist approach that does not consider order sorted structures.}

For the purpose of this example we only need the predicate “less than” which is encoded as a Boolean valued operation. In Maude notation we can write this as

\begin{verbatim}
sort Nat .
op _<_ : Nat Nat -> Bool .
\end{verbatim}

Next we consider a standard interpretation of the natural numbers as an \textit{MSA} algebra, where \textit{Nat} is interpreted as \(\omega\) (the set of the natural numbers) and \(\preceq\) is interpreted as the “less then” relation on \(\omega\).

(2) The list of the natural numbers \texttt{LIST\{Nat\}}. In Maude and CafeOBJ this is also a predefined module but in a different way than \texttt{NAT} is. It does not come from C, but it is rather defined as an equational specification. It is predefined in the sense that is already available in the system. Here we give it the simplest possible treatment:

\begin{verbatim}
sort List\{Nat\} .
op _ : Nat -> List\{Nat\} .
op nil : -> List\{Nat\} .
op __ : List\{Nat\} List\{Nat\} -> List\{Nat\} [assoc] .
\end{verbatim}

The first operation provides the generators for the lists, which are the natural numbers regarded as lists of size 1. In most algebraic specification languages this is handled more elegantly by declaring \textit{Nat} as a \textit{subsort} of \textit{List\{Nat\}}. However this requires \textit{order sorted algebra} \cite{19} which is a refinement of \textit{MSA} that is outside the scope of our discussion here. The constant \texttt{nil} represents the empty list and the binary operation \texttt{__} represent the concatenation (written in mixfix syntax). The word \texttt{assoc} abbreviates the equation specifying the associativity of the binary operation. The considered \textit{MSA} algebra interprets as the set of lists of natural numbers (denoted \(\omega^*\)) and \texttt{__} as lists concatenation.

These are \textit{common to all models of BUBBLE-SORT}; this is the meaning of \textit{protecting}.

From now on we can have various \textit{POA} models that satisfy the transition specified in the module. But before presenting some of them it is important to clarify the \(D\) and the \(S\) from the definition of the \textit{POA} signatures. \(D = \{\text{Nat,Bool}\}\) and \(S = \{\text{List\{Nat\}}\}\) which means that the \textit{POA} models do not consider a preorder on \(\omega\), the interpretation of \textit{Nat}. \(F\) consists of all operations of \textit{Nat}, whatever they may be, \(\preceq\) and the two \textit{LIST} operations.

What happens is that we start with a \textit{MSA} algebras, the set of sorts being \(D \cup S\), and then we consider \((D, S, F)\)-algebras by adding appropriate preorders. Let us describe some of the \((D, S, F)\)-algebras thus obtained.

(1) The preorder on \(\omega^*\) is the total preorder, i.e. \(l \preceq l'\) for all \(l, l' \in \omega^*\).
(2) The preorder on \(\omega^*\) is given by the reflexive-transitive closure of the relation

\[(l \preceq m \preceq l') \rightarrow (l \preceq n \preceq m \preceq l')\]
where \( l, l' \in \omega^* \), \( m, n \in \omega \).

(3) The preorder \( \preceq \) on \( \omega^* \) is given by
\[
l \preceq l' \quad \text{if and only if} \quad \text{inv}(l) \leq \text{inv}(l')
\]
where \( \text{inv}(l) \) represents the number of the inversions in \( l \).

If we match the transition specified by \( \text{BUBBLE-SORT} \) to the general definition \( \forall X : H \Rightarrow t \rightarrow t' \) we have that \( X = \{ m, n \} \), \( H \) is the equation \( (n < m) = \text{true} \), \( t = (m \ n) \) and \( t' \) is \( (n \ m) \). The first two models satisfy the transition of \( \text{BUBBLE-SORT} \) while the third model does not satisfy it.\(^8\)

Example 3. Let us consider the following simple specification:

\[
\begin{array}{l}
\text{mod SOLUTIONS is} \\
\text{sort s .} \\
\text{op f : s -> s .} \\
\text{var x : s .} \\
\text{rl f(x) => x .} \\
\text{endm}
\end{array}
\]

From the wide class of the POA models for the signature of this specification we consider the following ones:

\begin{itemize}
\item (1) The model \((A, \preceq_A)\): \( A_s = \mathbb{Z} \) (the set of the integers), \( A_f(n) = n^2 \), \( m \preceq_A n \) if and only if \( n^2 = m \) for some \( k \in \omega \).
\item (2) The model \((B, \preceq_B)\): \( B_s = \mathbb{Z} \), \( B_f(n) = n^2 \), \( m \preceq_B n \) if and only if \( |n| \leq |m| \).
\item (3) The model \((C, \preceq_C)\): \( C_s = \mathbb{Z} \), \( C_f(n) = n^2 \), \( m \preceq_C n \) if and only if \( m \) divides \( n \).
\end{itemize}

While the first two models do satisfy the transition of \text{SOLUTIONS} the third one does not.\(^9\)

3.2. **POA as institution.**

**Definition 6.** A POA signature morphism \( \varphi : (D, S, F) \rightarrow (D', S', F') \) is just an MSA signature morphism \( \varphi : (D \cup S, F) \rightarrow (D' \cup S', F') \) such that \( \varphi^{st} D \subseteq D' \) and \( \varphi^{st} S \subseteq S' \). Moreover
\begin{itemize}
\item Each POA \((D, S, F)\)-sentence \( \rho \) gets translated to a \((D', S', F')\)-sentence \( \varphi \rho \) by following the MSA translation of sentences that correspond to \( \varphi \) regarded as an MSA signature morphism \((D \cup S, F) \rightarrow (D' \cup S', F') \). The atomic transitions are translated in the same way as the atomic equations are translated.
\item Each POA \((D', S', F')\)-algebra \((A', \preceq)\) gets reduced to the \((D, S, F)\)-algebra \( \varphi(A', \preceq) = (\varphi A', \preceq) \). The reducts of POA \((D', S', F')\)-homomorphisms are defined as the reducts of the underlying MSA \((D' \cup S', F')\)-homomorphisms.
\end{itemize}

With all these the important ‘Satisfaction Lemma’ of Theorem 2.4 gets extended to POA in a straightforward way.

**Theorem 3.1.** For each POA signature morphism \( \varphi : (D, S, F) \rightarrow (D', S', F') \), for each \((D, S, F)\)-sentence \( \rho \) and each \((D', S', F')\)-algebra \((A', \preceq)\)
\[
(A', \preceq) \models_{(D', S', F')} \varphi \rho \quad \text{if and only if} \quad \varphi(A', \preceq) \models_{(D, S, F)} \rho.
\]

\(^8\)Since for instance \((2 1) \not\subseteq (1 2) \) which means that for \( m = 2, n = 1 \) the condition of the transition holds while the transition itself does not hold.

\(^9\)Since for instance \( C_f(3) = 9 \not\subseteq_C 3 \).
A comparison between POA and other semantic approaches to non-deterministic rewriting. In [25] the ‘rewriting logic’ (abbreviated RWL) was introduced as a logical foundations for non-deterministic rewriting. Then this was apparently implemented in the form of the Maude language [5]. However right from its inception RWL has suffered a number of mathematical shortcomings that propagated into a series of problems at the more practical levels. These culminated to the conclusion put forward by [7, 8] that RWL cannot be the logic / model theory of Maude, and instead the only such possibility is provided by POA. But due to the historically important role played by RWL it is worth to look into the (mathematical) relationship and differences between RWL and POA.

(1) While RWL is mathematically complicated as it involves sophisticated category-theoretic structures (e.g. sub-equalisers, etc.), POA achieves mathematical simplicity. In itself this may not be that important if it did not relate to other aspects discussed below.

(2) While RWL fails the Satisfaction Lemma of institution theory, all versions of POA are institutions. In other words, while POA belongs to the wide family of specification and programming logics that are captured as institutions [18, 12, 29], RWL does not. The advantages of being an institution are rather vast. On the practical side, the design of any language that implements POA rigorously would benefit directly from the rich institution theoretic modularisation theory. On the model theoretic side, POA benefits directly from the in-depth axiomatic developments provided by institution theory. Moreover, there is a strong interdependency relationship between the two sides.

(3) In the seminal work on RWL [25] and also in subsequent works it is presented and discussed a concept of projection from the categories of models of RWL to categories of models of POA that essentially collapses parallel transitions (arrows) to a preorder relation. In this way a model of RWL gets mapped to a preordered algebra. But as often happens in mathematics, simplifying structure does not necessarily lead to a theory that is a special case of the former, on the contrary it may lead to stronger and more useful properties. This projection has lead to a widespread mistaken belief that as a model theory POA is a collapsed form of RWL and consequently concepts and results from the former can be automatically transfered to the latter. There are rigorous arguments that refute such views such as in [7, 8] where, for instance, it is shown that the consequence relation in POA is different (i.e. larger) that what is obtained by collapsing RWL. This situation strengthen claims of novelty regarding results obtained directly in POA. The mathematical explanation for this important discrepancy is as follows. The proper way to establish the relationship between the two model theories is through the mathematics of institution (co)morphisms [18, 20, 12], but in the case of POA and RWL such an attempt fails immediately due to the latter formalism not being an institution. Naturally, this discrepancy between RWL and POA propagates to computational aspects.

(4) From the side of the logical languages, in some sense POA is significantly richer than RWL because while POA supports a fully fledged first-order language, RWL sentences are restricted to a proper sub-class of Horn sentences. For instance Horn sentences such

---

This failure is common knowledge among experts of the area. However as a statement it appears rather scarcely in the literature. But it can be found explicitly in [12] and implicitly in [7, 8].

A relevant monograph containing application to the model theory of POA in its abridged form is [12]. Since its publication the institution-theoretic approach to model theory has developed further.
as $\forall X \cdot (t \to t') \Rightarrow (t_1 = t_2)$ are not supported in $RWL$ because of the hierarchical built of $RWL$ in which the logic of transitions is built on top of equational logic. But these sentences can be very important to have in the applications. For instance when verifying properties of algorithms one needs to express that a transition has a certain property, for instance that a certain (semi-)invariant holds. Such properties are logically encoded as equations conditioned by transitions. As an example we may consider the termination of $BUBBLE\text{-}SORT$. If for each string $s$ we let $i(s)$ denote the number of its inversions, then in the logical language we should be able to express

$$\forall s, s' \cdot (s \to s') \Rightarrow (i(s) > i(s') = true).$$

It should be noted that while such sentences are supported in CafeOBJ they are not supported in Maude due to the latter’s design commitment to $RWL$.

In the recent paper [26] another semantic framework for non-deterministic rewriting is introduced, namely transition systems. As mathematical structures these are significantly weaker than preordered algebras as they are just algebras enhanced with binary relations without preorder and monotonicity requirements. So, in a sense we can say that $POA$ represents a middle ground between $RWL$ and the semantics put forward in [26].

4. Rewriting as Adjunction

In this section we do the following:

(1) We define on any algebra $B$ the rewriting relation $\mapsto_{\Gamma/B}$ determined by a set $\Gamma$ of transitions.

(2) We prove that $(B, \mapsto_{\Gamma/B})$ is a preordered algebra that satisfies $\Gamma$.

(3) We prove that the preordered algebras $(B, \mapsto_{\Gamma/B})$ arise as a left adjoint functor, a property that can be regarded as an implicit denotational definition of rewriting.

In this section $\Gamma$ denotes a set of $(D, S, F)$-transitions.

4.1. The explicit definition of algebraic rewriting. The following technical concept lies at the heart of rewriting.

**Definition 7** (Rewriting contexts). Given an MSA signature $(S, F)$, an $(S, F \cup \{z\})$-term $c$ over the signature extended with a new variable $z$ is a (rewriting) $(S, F)$-context if

- $c = z$, or
- $c = \sigma(c_1, \ldots, c_n)$ such that $\sigma \in F_{w \to s}$ is an operation symbol and there exists exactly one $k \in \{1, \ldots, n\}$ such that $c_k$ is context, with $c_i$ being just $(S, F)$-terms for $i \neq k$.

Then $c_k$ is called the immediate sub-context of $c$.

**Notation 2.** We denote contexts by $c[z]$ where $c$ is the actual context as a term and $z$ is the variable of the context.

**Definition 8.** Let $c[z]$ be a context of sort $s'$ such that the variable $z$ is of sort $s$, and let $B$ be any MSA algebra for the respective signature. The $B_c : B_s \to B_{s'}$ is the function defined as follows:

- If $c[z] = z$ then $B_c$ is an identity function.
Let \( \sigma = \sigma(c_1, \ldots, c_k[z], \ldots, c_n) \) where \( \sigma \) is an operation symbol and \( c_k[z] \) is the immediate sub-context of \( c[z] \) then
\[
B_c(b) = B_\sigma(B_{c_1}, \ldots, B_{c_k}(b), \ldots, B_{c_n}).
\]

**Notation 3.** For any set \( B \), we let \( \Delta_B \) denote its diagonal, i.e. \( \Delta_B = \{(b, b) \mid b \in B\} \). This notation extends to algebras by considering their underlying sets.

**Definition 9.** Let \( B \) be a \((D, S, F)\)-algebra and let \( \Omega \subseteq B \times B \). By \( \Gamma(\Omega) \) we denote the set
\[
\{(B_c(\theta t), B_c(\theta t')) \mid c[z] \text{ applicable context}\}
\]
of a system sort, \( \forall X \cdot H \Rightarrow (t \rightarrow t') \in \Gamma \), \( \theta : X \rightarrow B \), \( \theta H \subseteq \Delta_B, \theta H \subseteq \Omega \).

In what follows the role of \( \Omega \) will be played only by relations representing bounded applications of transitions on \( B \).

The following provides the definition of a rewriting relation on \( B \) determined by a set \( \Gamma \) of transitions.

**Definition 10.** Let \{\( \Gamma_{B,k} \mid k \in \omega \)\} be the following inductively defined sequence of binary relations on \( B \):
\[
\begin{align*}
\Gamma_{B,0} &= \Delta_B, \\
\Gamma_{B,k+1} &= \Gamma_{B,k}^* \cup \Gamma(\Gamma_{B,k}^*).
\end{align*}
\]
For any \( b, b' \in B \) we let \( b \rightarrow_{\Gamma/B} b' \) when there exists \( k \) such that \((b, b') \in \Gamma(\Gamma_{B,k}^*)\). Then \( \rightarrow_{\Gamma/B} \) denotes the reflexive-transitive closure of \( \rightarrow_{\Gamma/B} \).

**Proposition 4.1.** \( b \rightarrow_{\Gamma/B} b' \) if and only if \((b, b') \in \bigcup_{k \in \omega} \Gamma_{B,k} \).

**Proof.** For the implication from the left to the right we consider the chain
\[
b = b_1 \rightarrow_{\Gamma/B} b_2 \rightarrow_{\Gamma/B} \ldots \rightarrow_{\Gamma/B} b_n = b'.
\]
For each \( i \in [n - 1] \) there exists \( k_i \) such that \((b_i, b_{i+1}) \in \Gamma_{B,k_i} \). Let \( k = \max_{i \in [n-1]} k_i \).

Since \( \Gamma_{B,j} \) is increasing it follows that for each \( i \in [n - 1] \), \((b_i, b_{i+1}) \in \Gamma_{B,k} \), hence \((b, b') \in \Gamma_{B,k} \subseteq \Gamma_{B,k+1} \).

For the implication from the right to the left we prove by induction on \( k \in \omega \) that \((b, b') \in \Gamma_{B,k} \) implies \( b \rightarrow_{\Gamma/B} b' \). For \( k = 0 \) the conclusion follows by the reflexivity of \( \rightarrow_{\Gamma} \). Now let \( k > 0 \).

- When \((b, b') \in \Gamma_{B,k}^* \) the conclusion follows from the induction hypothesis by the transitivity of \( \rightarrow_{\Gamma/B} \).
- When \((b, b') \in \Gamma(\Gamma_{B,k}^*) \) it means that \( b \rightarrow_{\Gamma/B} b' \) hence \( b \rightarrow_{\Gamma/B} b' \).

The traditional implementations of term rewriting require restrictions on the occurrences of the variables in the rewrite rules: that all variables in the right-hand side term and in the conditions occur also in the left-hand side term. These also have a computational significance, beyond just implementation issues, which relates to termination. But for the

\[^{12}\text{The sort } z \text{ is the same with the sort of } t \text{ and } t'. \]
purpose of developing the results in this paper such conditions are not required. Moreover several implementations of rewriting go beyond such restrictions; these include languages / systems such as ASF+SDF, ELAN, Maude, which support matching conditions.

4.2. Model theoretic properties of algebraic rewriting.

Proposition 4.2. \((B, \rightarrow_{\text{algebra}})\) is a preordered \((D, S, F)\)-algebra.

Proof. Consider an operation \(\sigma\) of the respective signature \((D, S, F)\) such that \(\sigma \in F^{s_1\ldots s_n \rightarrow s}\) where \(s \in S\). For each \(k \in [n]\) let \(b_k, b'_k \in B_{s_k}\) such that

- \(b_k = b'_k\) if \(s_k \in D\), and
- \(b_k \rightarrow_{\Gamma/B} b'_k\) if \(s_k \in S\).

We have to prove that

\[B_\sigma(b_1, \ldots, b_n) \rightarrow_{\Gamma/B} B_\sigma(b'_1, \ldots, b'_n).\]

In order to simplify the presentation of the argument we consider the case \(n = 2\) and we assume that \(s_1, s_2 \in S\). For \(n > 2\) and some of the \(s_k\) are data sorts the argument is similar to the reduced case.

We prove by induction on \(k \in \omega\) that

\((b_1, b'_1) \in \Gamma_{B,k}\) implies \((B_\sigma(b_1, b_2), B_\sigma(b'_1, b_2)) \in \Gamma_{B,k}\).

For \(k = 0\) this is obvious. For the induction step we assume that this holds for \(k\) and prove it for \(k + 1\). So let us consider \((b_1, b'_1) \in \Gamma_{B,k+1}\). We distinguish two cases:

- When \((b_1, b'_1) \in \Gamma_{B,k}\). Then there exists \(b_1 = b^1_1, \ldots, b^1_n = b'_1\) such that \((b^1_k, b^{i+1}_k) \in \Gamma_{B,k}\). By the induction hypothesis it follows that \((B_\sigma(b_1, b_2), B_\sigma(b'_1, b_2)) \in \Gamma_{B,k}\) hence \((B_\sigma(b_1, b_2), B_\sigma(b'_1, b_2)) \in \Gamma_{B,k} \subseteq \Gamma_{B,k+1}\).
- When there exists \(\forall X : H \Rightarrow (t \rightarrow t') \in \Gamma\) and \(\theta : X \rightarrow B\) and context \(c[z]\) such that \(\theta H = \Gamma_{B,0}\) and \(\theta H_\rightarrow \subseteq \Gamma_{B,k}\), \(b_1 = B_c(\theta t)\) and \(b'_1 = B_c(\theta t')\). By considering the context \(c'[z] = \sigma(c[z], t_2)\) we obtain that \((B_\sigma(b_1, b_2), B_\sigma(b'_1, b_2)) \in \Gamma_{B,k+1}\).

By considering the identity context \(c[z] = z\) we obtain that \((\theta t, \theta t') \in \Gamma_{B,k+1}\). Hence \(\theta t \rightarrow_{\Gamma/B} \theta t'\) which means \(B'_1 \rightarrow_{\Gamma/B} B'_2\) which means \((B'_1, B'_2) \rightarrow_{\Gamma/B} \Gamma\).

Proposition 4.3. \((B, \rightarrow_{\Gamma/B}) \models \Gamma\).

Proof. Let \(\forall X : H \Rightarrow (t \rightarrow t')\) be any sentence in \(\Gamma\). Let \(B'\) be any \((D, S, F \cup X)\)-expansion of \(B\) such that \((B', \rightarrow_{\Gamma/B}) \models H\). Let \(\theta : X \rightarrow B\) such that \(\theta(x) = B'_x\) for each \(x \in X\). Then

- \(\theta H = \Gamma_{B,0}\) and
- \(\theta H_\rightarrow \subseteq \rightarrow_{\Gamma/B}\) which means that there exists \(k \in \omega\) such that \(\theta H_\rightarrow \subseteq \Gamma_{B,k}\).

By considering the identity context \(c[z] = z\) we obtain that \((\theta t, \theta t') \in \Gamma_{B,k+1}\). Hence \(\theta t \rightarrow_{\Gamma/B} \theta t'\) which means \(B'_1 \rightarrow_{\Gamma/B} B'_2\) which means \((B'_1, B'_2) \rightarrow_{\Gamma/B} \Gamma\).

4.3. The adjunction property of algebraic rewriting.
Notation 4. For any MSA signature \((S, F)\) let \(\text{Mod}(S, F)\) denote the category of the MSA \((S, F)\)-algebras and their homomorphisms. For any POA signature \((D, S, F)\) we let \(\text{Mod}(D, S, F)\) denote the category of the POA \((D, S, F)\)-models and their homomorphisms, and \(\text{Mod}(D, S, F, \Gamma)\) denote the full category of \(\text{Mod}(D, S, F)\) defined by those preordered algebras that satisfy \(\Gamma\).

The following theorem shows that any set \(\Gamma\) of transitions endows any MSA algebra with a minimal preorder structure that yields a POA algebra that satisfies \(\Gamma\). Moreover the original MSA algebra is preserved. Technically it is convenient to present this result as a categorical adjunction. The uniqueness property from this result indicates that this can be taken as an implicit definition of non-deterministic algebraic rewriting.

Theorem 4.4. The forgetful functor \(U: \text{Mod}(D, S, F, \Gamma) \rightarrow \text{Mod}(D \cup S, F)\) has a left-adjoint left inverse which maps any \((D \cup S, F)\)-algebra \(B\) to the preordered algebra \((B, \rightarrow_{\Gamma/B})\).

Proof. It is enough if we proved that:

For any preordered \((D, S, F)\)-algebra \((A, \leq)\) such that \((A, \leq) \models \Gamma\) and for any \(h: B \rightarrow A\) a \((D \cup S, F)\)-homomorphism, \(h\) is a homomorphism of POA algebras \((B, \rightarrow_{\Gamma/B}) \rightarrow (A, \leq)\) too.

\[
\begin{array}{cccc}
B & \xrightarrow{=} & U(B, \rightarrow_{\Gamma/B}) & \xrightarrow{=} & (B, \rightarrow_{\Gamma/B}) \\
\downarrow h & & \downarrow h & & \downarrow h \\
U(A, \leq) & & (A, \leq) & & \end{array}
\]

Thus we only have to prove that \(b \rightarrow_{\Gamma/B} b'\) implies \(h(b) \leq h(b')\). By induction on \(k \in \omega\) we prove that \((b, b') \in \Gamma_{B,k}\) implies \(h(b) \leq h(b')\).

For \(k = 0\) the conclusion follows by the reflexivity of \(\leq\).

For the induction step we assume \((b, b') \in (\Gamma_{B,k+1})^*\). We have two cases:

1. \((b, b') \in (\Gamma_{B,k})^*\), or
2. \((b, b') \in \Gamma((\Gamma_{B,k})^*)\), i.e. \(b = B_x(\theta t)\) and \(b' = B_x(\theta t')\) where \(c[z]\) is a context, \(\forall X \cdot H \Rightarrow (t \rightarrow t')\) belongs to \(\Gamma\), \(\theta : X \rightarrow B, \theta H = \subseteq \Gamma_{B,0}\) and \(\theta H = \subseteq \Gamma_{B,k}\).

In the former case, by the induction hypothesis \(h(\Gamma_{B,k}) \subseteq \leq\). It follows that \(h(\Gamma_{B,k}) = (h(\Gamma_{B,k}))^* \subseteq \leq^* = \leq\) (the last equality follows by the transitivity of \(\leq\)).

In the latter case we have \(h(b) = h(B_x(\theta t)) = A_h(\theta(\theta t))\) and likewise \(h(b') = A_h(\theta(\theta t'))\). Thus, by the monotonicity of the interpretations of the operations in \(A\) it is enough to prove that \(h(\theta t) \leq h(\theta t')\).

Let us consider \(A'\) the \((D, S, F \cup X)\)-expansion of \(A\) such that \(A'_{t} = A_h(\theta z)\) for each \(x \in X\). We establish that \((A', \leq) \models H\). Let \(B'\) be the \((D, S, F \cup X)\)-expansion of \(B\) such that \(B'_{t} = \theta x\) for each \(x \in X\). Note that \(h\) is also a \((D, S, F \cup X)\)-homomorphism \(B' \rightarrow A'\).

- Let \((t_1 = t_2) \in H_x\). Then \((A'_{t_1}, A'_{t_2}) = (h(B'_{t_1}), h(B'_{t_2})) = (h(\theta t_1), h(\theta t_2)) \in h(\theta H_x) \subseteq h(\Gamma_{B,0})\). Hence \(A'_{t_1} = A'_{t_2}\) which means \(A' \models (t_1 = t_2)\).
• Let \((t_1 \rightarrow t_2) \in H_-\). Then, similarly to above we get that \((A'_t, A''_t) \in h(\theta H_-) \subseteq h(\Gamma^+_B)\). From the proof at the former case of the induction step we know that \(h(\Gamma^+_B) \subseteq \leq\). Hence \(A'_t \subseteq A''_t\) which shows that \(A' \models (t_1 \rightarrow t_2)\).

Finally, since \((A, \leq) \models \forall X \cdot H \Rightarrow (t \rightarrow t')\) and \((A', \leq) \models H\) it follows that \((A', \leq) \models (t \rightarrow t')\) which means \(A'_t \subseteq A'_{t'}\) which means \(h(\theta t) \leq h(\theta t')\).

The unit given by the adjunction given by this theorem is the identity. In the literature, eg. [18, 12] such adjunctions are called strongly persistent adjunctions. They are stronger than the ordinary adjunctions but weaker than the equivalences of categories.

**Example 4.** We continue Example 2. Recall that from the three POA \((D, S, F)\)-models presented there only the first two satisfy the transition of the specification. From these two the second one is the free \((D, S, F, \Gamma)\)-model over the underlying \((D \cup S, F)\)-algebra of the lists of natural numbers.

**Example 5.** We continue Example 3. There \(D = \emptyset\). From the three POA models presented in Example 3, all of them sharing the same underlying MSA \((S, F)\)-algebra, only \(A\) and \(B\) satisfy the transition of SOLUTIONS. From these two \(A\) is the free model over the underlying MSA algebra.

This example displays a kind of non-determinism that is not available in the term based approaches to rewriting, neither in RWL nor on POA. For instance we have that \(4 \leq_A 2\) and \(4 \leq_A -2\) because the equation \(x^2 = 4\) has 2 and \(-2\) as solutions. In the approaches that deal only with terms, a single transition can be applied only in one way to a term because term rewriting matching is deterministic. In our example this matching corresponds to solving quadratic equations \(x^2 = a\), which may have two solutions. Implementation of algebraic rewriting over predefined types may replace the matching algorithms by constraint solvers.

**Notation 5.** It is well known that any equational theory \(E\) admits an initial algebra. Let us denote it by \(0_E\). Given a set \(\Gamma\) of transitions we abbreviate by \(\rightarrow_{\Gamma/E}\) the rewriting relation \(\rightarrow_{\Gamma, 0_E}\) on \(0_E\) determined by \(\Gamma\).

We have the following practically important consequence of Theorem 4.4.

**Corollary 4.5.** \((0_E, \rightarrow_{\Gamma/E})\) is the initial preordered algebra that satisfies \(E \cup \Gamma\).

**Proof.** \(0_E \models E\) and moreover \((0_E, \rightarrow_{\Gamma/E}) \models \Gamma\) by Proposition 4.3. Let \((A, \leq)\) be any preordered \((D, S, F)\)-algebra such that \((A, \leq) \models E \cup \Gamma\). Then \(A \models E\). By the initiality property of \(0_E\) there exists an unique \((D \cup S, F)\)-homomorphism \(h : 0_E \rightarrow A\). By Theorem 4.4 we have that \(h\) is also a \((D, S, F)\)-homomorphism \((0_E, \rightarrow_{\Gamma/E}) \rightarrow (A, \leq)\).

Modulo RWL restrictions of the occurrences of variables, the result of Corollary 4.5 corresponds to a related initiality result in [25]. However due to the collapse of RWL to POA being illusory, as discussed in Section 3.2, the derivation of Corollary 4.5 from the initiality result of [25] appears as problematic. Moreover, the source of this corollary, namely Theorem 4.4, does not have any correspondent in the RWL literature. An important class of examples of this refer to rewriting on predefined types, something which is unavoidable
in any proper programming language, Maude and CafeOBJ included. For instance both Examples 4 and 5 fall outside the scope of the initial semantics result of Corollary 4.5.

Example 6. The RWL literature abounds of examples that fall within the scope of the initial semantics result of [25], and many of those can be made examples for our Corollary 4.5. However let us provide here a concrete example based on BUBBLE-SORT. For this we have to replace the predefined NAT and BOOL by user defined NAT and BOOL with initial equational semantics. We can get minimalist about that and consider the following specification.

```plaintext
mod NAT is
  sorts Bool Nat .
  ops true false : -> Bool .
  op 0 : -> Nat .
  op s_ : Nat -> Nat .
  vars n m : Nat .
  eq (0 < s n) = true .
  eq (n < 0) = false .
  eq (s m < s n) = (m < n) .
endm
```

Then $E$ of Corollary 4.5 consists of the three equations of NAT above plus the associativity equation of lists concatenation. $0_E$ is precisely the MSA algebra of the lists of naturals. Consequently $\rightarrow_{\Gamma/E}$ corresponds to the preorder of the second model presented in Example 2.

5. Computing in preordered algebras

Deterministic term rewriting comes with some clear and practically relevant sufficient conditions (i.e. confluence and termination) for the completeness of computations. Under these conditions the completeness is explained in various different ways that suit different backgrounds. The highest and the mathematically most advanced such explanation is the isomorphism between, on the one hand the initial algebra of the corresponding equational theory, and on the other hand the algebra of the normal forms of the rewriting relation on terms. This applies equally to plain term rewriting and to term rewriting modulo a theory. The former model can be considered as the denotational model and the latter as the computational model.\(^\text{13}\)

This approach brings a novelty to the theory of non-deterministic rewriting. Most of the RWL literature around the language Maude (e.g. [5]) defines a computational method for non-deterministic rewriting on initial algebras modulo equational theories, defines its completeness in proof theoretic terms, and provides sufficient conditions for this completeness. Moreover all these are presented in a colloquial style [5] lacking mathematical rigor. However in [26] the so-called “coherence problem” is defined in terms that are similar to our model homomorphisms approach, but modulo the crucial and rather ample difference between the model theoretic structures involved. Our aim in this section is to fill this gap for this computational method, first by providing a computational model in a model theoretic sense within the framework of POA, and then by specifying a set of sufficient

\(^{13}\text{Note that here the term “model” is used in the precise mathematical sense given by model theory rather than in the informal sense given by the concept of “modelling”.}\)
conditions that guarantee its existence and its soundness and completeness. Because the 
computational method refers to non-deterministic rewriting modulo equational theories our 
completeness result is built on top of the above mentioned model theoretic completeness 
of deterministic rewriting.

More specifically, in this section we do the following.

1. We present the method to compute with POA transitions modulo equations that is em-
ployed by languages such as Maude and CafeOBJ. This constitutes the main application 
for the theory developed in this section.

2. We define an abstract POA model that serves as a generic “computational” model.

3. Under some conditions that match very well the practice of programming in Maude and 
CafeOBJ we prove the isomorphism between the computational model and the denota-
tional model given by the adjointness result of Theorem 4.4. This isomorphism result 
explains the soundness and the completeness of the respective computational method.

5.1. The computational method. In this section we present the method employed by 
Maude and CafeOBJ for hybrid computing with equations and transitions. However our 
presentation is tailored to the framework of this paper.

For a program consisting of a set $E$ of equations and a set $\Gamma$ of transitions, in principle 
there are two different levels of computing modulo axioms.

1. An “upper” level where the rewriting relation is determined by $\Gamma$ on the initial algebra 
$0_E$. This is rewriting by $\Gamma$ modulo $E$.

2. A “lower” level where $E$ is partitioned as $E = E_0 \cup E_1$, with $E_1$ being used as rewrite 
rules modulo $E_0$. Thus $0_E$ is obtained as the algebra of the normal forms of rewriting 
modulo $E_0$ by $E_1$, or otherwise said the normal forms of rewriting by $E_1$ in $0_{E_0}$. At 
this level, with respect to $E_0$ Maude and CafeOBJ implement only associativity, com-
mutativity, identity of user defined binary operations.

Thus with a program consisting of a set $E$ of equations and a set $\Gamma$ of transitions, computa-
tions involve two rewriting systems, one for $E_1$ (modulo $E_0$), and one for $\Gamma$ (also 
modulo $E_0$). Here there is the basic assumption that 

the rewrite system modulo $E_0$ determined by $E_1$ is confluent and terminating,

so each term $t$ modulo $E_0$, i.e., $t_{E_0} \in 0_{E_0}$, has an unique normal form $[t_{E_0}]$ with respect 
to $\xrightarrow{*}_{E_1/E_0}$.

Example 7. Let us consider BUBBLE-SORT with user defined data types (like in Example 
6). Then $E_0, E_1, \Gamma$ are as follows:

- $E_0$ consists of the associativity of list concatenation.
- $E_1$ consists of the three equations of NAT as specified in Example 6.
- $\Gamma$ consists of the single transition.

Then $E_1$ is confluent and terminating on the sorts Nat and Bool, the normal forms of sort 
Nat being the natural numbers in the form $(s \ s \ldots s \ 0)$, and of sort Bool being just 
the two constants. This extends immediately to the confluence and termination modulo 
$E_0$, that takes place on the sort List{Nat}, because in the terms the list concatenation 
operation _ never occurs below an operation from NAT.

---

14The congruence class of $t$ determined by $E_0$ on the term algebra.
Then a computation process that starts with a term \( t \) is governed by the following non-deterministic algorithm:

0. The current term \( T \) is set to \( t \).
1. \( T \) is reduced to its normal form \([T] \) under \( \rightarrow_{E_1/E_0} \).
2. The new value of \( T \) is set to a term obtained from \([T] \) by applying one rewrite step by \( \Gamma \). Then we move to step 1.

This can be considered as a computational modelling of the initial preordered algebra \((0_E, \rightarrow_{\Gamma/E})\) (Corollary 4.5). In what follows we provide an argument for this.

According to the theory of equational rewriting, in the presence of the assumption of termination and confluence, the initial algebra \(0_E\) and the algebra \(N_{E_1/E_0}\) of the normal forms of the rewriting relation \( \rightarrow_{E_1/E_0} \) on \(0_E\) are isomorphic. The mathematical modelling of the algorithm above that defines computation in POA is given by the reflexive-transitive closure of the following relation on \(N_{E_1/E_0}\):

\[
[t_{E_0}] \rightarrow_{\Gamma/\emptyset} [t'_{E_0}] \quad \text{iff there exists } u \text{ such that } [t_{E_0}] \rightarrow_{\Gamma/E_0} u_{E_0} \text{ and } [u_{E_0}] = [t'_{E_0}].
\]

Thus our goal is

- first, to establish that \((N_{E_1/E_0}, \rightarrow_{\Gamma/\emptyset})\) is a preordered algebra, and
- second, to establish that the isomorphism \( h : 0_E \rightarrow N_{E_1/E_0}\) gives an isomorphism of preordered algebras

\[
(0_E, \rightarrow_{\Gamma/E}) \cong (N_{E_1/E_0}, \rightarrow_{\Gamma/\emptyset}).
\]

In order to establish (1) it is enough to prove two things:

(2) that \( h(\rightarrow_{\Gamma/E}) \subseteq \rightarrow_{\Gamma/\emptyset} \), and

(3) that \( h^{-1}(\rightarrow_{\Gamma/\emptyset}) \subseteq \rightarrow_{\Gamma/E} \).

The latter fact, which is the easier one, represents a sort of soundness property, because it says that through our algorithm, from each state \( t_E \) we can reach only states \( t'_E \) such that \( t_E \rightarrow_{\Gamma/E} t'_E \). The former fact, more difficult to establish, represents a sort of completeness aspect: if \( t_E \rightarrow_{\Gamma/E} t'_E \) then \( t'_E \) can be computed from \( t_E \) by our algorithm.

5.2. The abstract computational POA model. In order to study the properties (2) and (3) it is mathematically convenient to abstract \( 0_{E_0}, 0_E \) and \( N_{E_1/E_0} \) to arbitrary algebras; in this way we achieve simpler notations, we get rid of some concrete redundant details (such as terms and congruence classes of terms) and by those we are able to unhide the essential causalities. Moreover this abstract framework can be also applied to other particular situations of interest such as rewriting-based computations with predefined types. In fact all these are well known general benefits of axiomatic treatment of problems. In this case the mathematical object thus obtained will be an abstract POA model that represents the computational method.
In what follows we will often switch between MSA and POA algebras and back. This is based on an implicit assumption of a POA signature \((D, S, F)\) and of its associated MSA signature \((D \cup S, F)\).

The following concept captures abstractly the essence of the normal forms with respect to rewriting relations which has two aspects: as a mapping, the computation of normal forms is homomorphic and idempotent.

**Definition 11.** An MSA homomorphism \(\beta : B \rightarrow N\) is an nf-homomorphism when \(N \subseteq B\) and \(N\) is invariant with respect to \(\beta\), i.e., \(\beta n = n\) for each \(n \in N\).

**Example 8.** [Normal forms of common term rewriting] We let \(B = 0_{(S,F)}\) be the term algebra for an MSA signature \((S, F)\) and \(E\) be a set of \((S, F)\)-equations which we use for rewriting. If the rewriting relation \(\rightarrow_{E/0_{(S,F)}}\) is confluent and terminating then the normal forms \([t]\) of the terms \(t\) form an algebra \(N_E\) (this is \(N\) of the Definition 11) that is actually isomorphic to the initial algebra \(0_E\). The mapping of the terms to their normal forms, i.e. \(t \mapsto [t]\) is a homomorphism \(0_{(S,F)} \rightarrow N_E\), so \(\beta\) is \([\cdot]\).

**Example 9.** [Normal forms of term rewriting modulo axioms] We let \(B = 0_{E_0}\) be the initial algebra of an equational theory \(E_0\) for an MSA signature \((S, F)\). Let \(E_1\) be a set of \((S, F)\)-equations used for rewriting modulo \(E_0\). If the rewriting relation \(\rightarrow_{E_1/E_0}\) (on \(0_{E_0}\)) is confluent and terminating then the normal forms \([t_{E_0}]\) of the terms \(t_{E_0}\) modulo \(E_0\) form the algebra that we denoted \(N_{E_1/E_0}\) (this is \(N\) of Definition 11). Then the mapping \([\cdot] : 0_{E_0} \rightarrow N_{E_1/E_0}\) is an MSA homomorphism.

The following gives an abstract mathematical definition of the above non-deterministic algorithm that combines equational with transition-based rewriting. What it essentially says is that the rewriting relation projected on the normal forms should be obtained through a finite sequence of steps of the algorithm defining the computation method. The notation \(\beta(\rightarrow_{\Gamma/B})\) stands for this projection, i.e. the relation \(\{(\beta b_1, \beta b_2) \mid b_1 \rightarrow_{\Gamma/B} b_2\}\).

**Definition 12.** Let \(\Gamma\) be a set of transitions, \(\beta : B \rightarrow N\) be an nf-homomorphism and \(\rightarrow_{\Gamma/B}\) be the following binary relation on \(N:\)

\[n_1 \rightarrow_{\Gamma/B} n_2\quad \text{if and only if there exists } b_2 \text{ such that } n_1 \rightarrow_{\Gamma/B} b_2 \text{ and } \beta b_2 = n_2.\]

We say that \(\Gamma\) is \(\beta\)-coherent when

\[\beta(\rightarrow_{\Gamma/B}) \subseteq \rightarrow_{\Gamma/B}.\]

**Example 10.** When \(B = 0_{E_0}\), \(N = N_{E_1/E_0}\), \(\beta = [\cdot]\), that \(\Gamma\) is \([\cdot]\)-coherent means that whenever \(t_{E_0} \rightarrow_{\Gamma/E_0} t'_{E_0}\) there exists \(u\) such that \([t_{E_0}] \rightarrow_{\Gamma/E_0} u_{E_0}\) and \([u_{E_0}] = [t'_{E_0}]\).

With the next two examples we get more concrete about the \(\beta\)-coherence condition.

**Example 11.** Within the context of the Examples 7 and 10 combined we have that \(B = 0_{E_0}\) is the algebra of the lists of \(\Sigma_{\text{nat}}\)-terms, where \(\Sigma_{\text{nat}}\) is the signature \(\text{Nat}\) of Example 6. In \(B\), on the sort \(\text{Bool}\), besides \(\text{true}\) and \(\text{false}\) we have also the potential inequalities between natural numbers. \(N = N_{E_1/E_0}\) is the algebra of the lists of the natural numbers, that share with \(B\) the same interpretations of the sorts \(\text{Nat}\) and \(\text{List}\{\text{Nat}\}\), but on \(\text{Bool}\)
we have only the two constants, true and false. The homomorphism $\beta$ essentially reduces any potential inequality between natural numbers to any of the two constants. The $\beta$-coherence conditions holds trivially true because $\Gamma/B$ (aka $\Gamma, E_0$) is empty as the transition of $\Gamma$ cannot be applied because the equations responsible for the evaluations of the Boolean condition belong to $E_1$ rather than $E_0$.

However we can twist this example such that the $\beta$-coherence condition is not trivial anymore. First we move the three equations of $\text{NAT}$ from $E_1$ to $E_0$. If we did just that then the $\beta$-coherence condition would still be trivial but in a different way than before; now $\beta$ would be an identity homomorphism because $E_1$ gets emptied. However we can still escape this triviality by adding more operations on the natural numbers, such as an addition operation which is specified as follows:

\[
\begin{align*}
op \_+&_\_ & : \text{Nat Nat} \rightarrow \text{Nat} . \\
eq m + 0 &= m . \\
eq m + s n &= s(m + n) .
\end{align*}
\]

Then $E_1$ would consist of the two equations above. Now $\beta$ essentially evaluates additions. Let us consider the $\beta$-coherence condition as given in Example 10. Then $t_{E_0} \rightarrow_{\Gamma/E_0} t'_{E_0}$ means that two adjacent natural numbers from $t_{E_0}$, in their normal form, are swapped. The rest of the elements of $t_{E_0}$ may not necessarily be free of additions. In this case $[t_{E_0}]$ is the list of the natural numbers obtained by evaluating all additions in $t_{E_0}$ and $u_{E_0}$ is the same with respect to $t'_{E_0}$. Obviously $[u_{E_0}] = u_{E_0}$ and moreover $[t_{E_0}] \rightarrow_{\Gamma/E_0} u_{E_0}$.

The next example illustrates a failure of the $\beta$-condition. It is adapted after an example from [5].

**Example 12.** Let us consider the following specification:

```
mod BETA-FAIL is
  sorts U B M .
  ops a b c : -> U .
  ops 0 1 : -> B .
  op [__] : U B -> M [assoc comm] .
  eq [a 0] [b 1] [c 0] = [a 0] [b 1] .
  eq [a 1] [b 0] [c 1] = [a 1] [b 0] .
  rl [b 0] => [b 1] .
  rl [b 1] => [b 0] .
  rl [a 0] [b 0] => [a 1] [b 1] .
  rl [a 1] [b 1] => [a 0] [b 0] .
endm
```

We set $E_0$ to the associativity and the commutativity of the union of multisets, $E_1$ to the set of the two explicit equations, and $\Gamma$ to the set of the four transitions. Let $t_{E_0}$ be $[a 0] [b 1] [c 0]$ and $t'_{E_0}$ be $[a 0] [b 0] [c 0]$. Then $t_{E_0} \rightarrow_{\Gamma/E_0} t'_{E_0}$ through the application of the second transition. We also have $[t_{E_0}] = [a 0] [b 1]$ and $[t'_{E_0}] = t'_{E_0}$. But any $u_{E_0}$ such that $[t_{E_0}] \rightarrow_{\Gamma/E_0} u_{E_0}$ does not contain $c$, so we cannot possibly have $[u_{E_0}] = [t'_{E_0}] (= t'_{E_0})$.

The $POA$ model $(N, \rightarrow_{\Gamma/\beta})$ obtained through Proposition 5.1 below is an abstraction of the envisaged computational model $(N_{E_1/E_0}, \rightarrow_{\Gamma/\beta})$. 
Proposition 5.1. Let \( \beta : B \rightarrow N \) be an nf-homomorphism and \( \Gamma \) be a \( \beta \)-coherent set of transitions. Then \( (N, \rightarrow_{\Gamma/\beta}) \) is a preordered algebra and moreover \( \beta : (B, \rightarrow_{\Gamma/B}) \rightarrow (N, \rightarrow_{\Gamma/\beta}) \) is a POA homomorphism.

Proof. We have to prove that the interpretation of the operation symbols by \( N \) are monotone with respect to \( \rightarrow_{\Gamma/\beta} \). Let us consider an operation symbol \( \sigma \) of a system sort. Without any loss of generality, in order to simplify the presentation, we assume that \( \sigma \) takes precisely two arguments. We first prove that:

\[
(4) \quad n_1 \rightarrow_{\Gamma/\beta} n_2 \quad \text{implies} \quad N_\sigma(n_1, n) \rightarrow_{\Gamma/\beta} N_\sigma(n_2, n).
\]

By hypothesis there exists \( m_2 \) such that \( n_1 \rightarrow_{\Gamma/B} m_2 \) and \( \beta m_2 = n_2 \). Then

\[
\begin{align*}
1 & \quad B_\sigma(n_1, n) \rightarrow_{\Gamma/B} B_\sigma(m_2, n) & \quad \text{\( B_\sigma \) monotone with respect to \( \rightarrow_{\Gamma/B} \)} \\
2 & \quad \beta(B_\sigma(n_1, n)) \rightarrow_{\Gamma/\beta} \beta(B_\sigma(m_2, n)) & \quad \text{from 1 since \( \beta(\rightarrow_{\Gamma/B}) \subseteq \rightarrow_{\Gamma/\beta} \) because \( \Gamma \) is \( \beta \)-coherent} \\
3 & \quad N_\sigma(\beta n_1, \beta n) \rightarrow_{\Gamma/\beta} N_\sigma(\beta m_2, \beta n) & \quad \text{from 2 since \( \beta \) is MSA homomorphism} \\
4 & \quad N_\sigma(n_1, n) \rightarrow_{\Gamma/\beta} N_\sigma(n_2, n) & \quad \text{from 3 since \( \beta m_2 = n_2 \) and \( N \) is invariant wrt \( \beta \)}.
\end{align*}
\]

By transitivity we extend (4) to

\[
(5) \quad n_1 \rightarrow_{\Gamma/\beta} n_2 \quad \text{implies} \quad N_\sigma(n_1, n) \rightarrow_{\Gamma/\beta} N_\sigma(n_2, n).
\]

In a similar way we may prove that

\[
(6) \quad n_1 \rightarrow_{\Gamma/\beta} n_2 \quad \text{implies} \quad N_\sigma(n_1, n_1) \rightarrow_{\Gamma/\beta} N_\sigma(n_2, n_2).
\]

Now let us consider \( n_1, n_1', n_2, n_2' \) arguments for \( N_\sigma \) such that \( n_1 \rightarrow_{\Gamma/\beta} n_2 \) and \( n_1' \rightarrow_{\Gamma/\beta} n_2' \). We have that:

\[
\begin{align*}
1 & \quad N_\sigma(n_1, n_1') \rightarrow_{\Gamma/\beta} N_\sigma(n_2, n_1') & \quad \text{from (5)} \\
2 & \quad N_\sigma(n_2, n_1') \rightarrow_{\Gamma/\beta} N_\sigma(n_2, n_2') & \quad \text{from (6)} \\
3 & \quad N_\sigma(n_1, n_1') \rightarrow_{\Gamma/\beta} N_\sigma(n_2, n_2') & \quad \text{from 1 and 2 by the transitivity of \( \rightarrow_{\Gamma/\beta} \)}.
\end{align*}
\]

Thus \( (N, \rightarrow_{\Gamma/\beta}) \) is a preordered algebra. In addition to that, from the \( \beta \)-coherence assumption it follows that \( (B, \rightarrow_{\Gamma/B}) \rightarrow (N, \rightarrow_{\Gamma/\beta}) \) is a POA homomorphism. \( \square \)

Corollary 5.2. If \( \Gamma \) is \( [\_] \)-coherent then \( (N_{E_1/E_0}, \rightarrow_{\Gamma/\Gamma}) \) is a preordered algebra and \( [\_] : (0_{E_0}, \rightarrow_{\Gamma/B}) \rightarrow (N_{E_1/E_0}, \rightarrow_{\Gamma/\Gamma}) \) is a POA homomorphism.

The role played by the \( \beta \)-coherence is to establish \( (N, \rightarrow_{\Gamma/\beta}) \) as a POA model. The following example shows that in its absence this may fail to be a POA model.

Example 13. Let us consider the example of the failure of \( \beta \)-coherence given by BETA–FAIL of Example 12. Then we have that

\[
(7) \quad \begin{bmatrix} a & 0 \\ b & 0 \\ c & 0 \end{bmatrix} \rightarrow_{\Gamma/\beta} \begin{bmatrix} a & 0 \\ b & 1 \end{bmatrix}.
\]

because \( \begin{bmatrix} a & 0 \\ b & 0 \\ c & 0 \end{bmatrix} \rightarrow_{\Gamma/B} \begin{bmatrix} a & 0 \\ b & 1 \\ c & 0 \end{bmatrix} \) and
\( \begin{bmatrix} a & 0 \\ b & 1 \\ c & 0 \end{bmatrix} = [a, 0] [b, 1]. \) Let us add \([a, 1] [c, 1]\) to each of the two multisets of (7). If the union of the multisets were monotone with respect to \( \rightarrow_{\Gamma/\beta} \) then we would have that

\[
\begin{bmatrix} a & 0 \\ a & 1 \\ b & 0 \\ c & 0 \\ c & 1 \end{bmatrix} \rightarrow_{\Gamma/\beta} \begin{bmatrix} a & 0 \\ a & 1 \\ b & 1 \\ c & 1 \end{bmatrix}
\]
which means
\[
[a\ 0] [a\ 1] [b\ 0] [c\ 0] \xrightarrow{}_{\Gamma/\beta} [a\ 0] [a\ 1] [b\ 1] [c\ 1].
\]
But this is not possible because there is no way to transform \([c\ 0]\) into \([c\ 1]\). Therefore the union of the multisets is not monotone with respect to \(\xrightarrow{}_{\Gamma/\beta}\).

5.3. The soundness of the operational semantics.

**Proposition 5.3.** Under the hypotheses of Proposition 5.1 if \(\beta' : N \rightarrow B'\) is an MSA homomorphism then \(\beta' : (N, \xrightarrow{}_{\Gamma/\beta}) \rightarrow (B', \xrightarrow{}_{\Gamma/\beta'})\) is a POA homomorphism.

**Proof.** In this case we have only to prove that

\[
\beta'([\beta n]) \subseteq \xrightarrow{}_{\Gamma/\beta'} \beta'([\beta b_2]).
\]

Let \(n_1 \xrightarrow{}_{\Gamma/\beta} n_2\) and let \(b_2 \in B\) such that \(n_1 \xrightarrow{}_{\Gamma/\beta} b_2\) and \(\beta b_2 = n_2\). Since \(\beta' \circ \beta : B \rightarrow B'\) is an MSA homomorphism, by virtue of Theorem 4.4 we have that

\[
\beta' \circ \beta : (B, \xrightarrow{}_{\Gamma/\beta}) \rightarrow (B', \xrightarrow{}_{\Gamma/\beta'})
\]

is a POA homomorphism. Hence

\[
\beta'([\beta n_1]) \subseteq \xrightarrow{}_{\Gamma/\beta'} \beta'([\beta b_2]).
\]

which by the invariance assumption and since \(\beta b_2 = n_2\) means

\[
\beta'([\beta n_1]) \subseteq \xrightarrow{}_{\Gamma/\beta'} \beta'([\beta n_2]).
\]

Hence \(\beta'([\beta n]) \subseteq \xrightarrow{}_{\Gamma/\beta'} \beta'([\beta b_2])\). This can be extended to (8) by using the transitivity of \(\xrightarrow{}_{\Gamma/\beta'}\).

The property (3) now follows immediately:

**Corollary 5.4** (The concrete soundness of the computational method). If \(\Gamma\) is a \(\llbracket\cdot\rrbracket\)-coherent then \(h^{-1}([\xrightarrow{}_{\Gamma/\Gamma}]) \subseteq \xrightarrow{}_{\Gamma/\Gamma'}\).

**Proof.** In Proposition 5.3 we set \(B = 0_{E_0}, B' = 0_{E}, N = N_{E_1/E_0}, \beta' = h^{-1}\) (where \(h\) is the isomorphism \(0_{E} \rightarrow N_{E_1/E_0}\)).

5.4. The completeness of the operational semantics. It is common in model theory that the completeness properties are more difficult to establish than the soundness properties. This is also the case with the completeness of the POA operational semantics with respect to the denotational semantics, which needs some additional conditions.

**Proposition 5.5.** Let \(\beta : B \rightarrow N\) be an nf-homomorphism and let \(\Gamma\) be a \(\beta\)-coherent set of transitions that in addition satisfy the following conditions:

- For each transition \(\forall X \cdot H \Rightarrow (t \rightarrow t')\) in \(\Gamma\), \(H\) is a finite conjunction of equations of the form \(t_i = c_i\) where \(c_i\) is a constant such that \(B_{c_i} \subseteq N\).
- For each sort \(s\) of an equation \(t_i = c_i\) that occurs in a condition of a transition in \(\Gamma\) (like above):

\[
B \models \forall x \cdot \bigvee \{x = c \mid c \in F_{\rightarrow s}, B_c \in N\} \quad \text{(no junk)}
\]

\[
B \models \bigwedge \{c \neq c' \mid c \neq c' \in F_{\rightarrow s}, B_c, B_{c'} \in N\} \quad \text{(no confusion)}
\]

Then \((N, \xrightarrow{}_{\Gamma/\beta}) \models \Gamma\).
Proof. Let $\forall X \cdot H \Rightarrow (t \rightarrow t') \in \Gamma$ and $\overline{N}$ be an expansion of $N$ such that $\overline{N} \models H$. Let $(t_i = c_i) \in \overline{\Gamma}$. Then $\overline{N} \models (t_i = c_i)$. Let $\overline{B}$ be the expansion of $B$ such that for each variable $x \in X$ we have $\overline{B}_x = \overline{N}_x$. This determines also an expansion $\overline{\beta}$ of $\beta$. Let us prove that $\overline{B} \models (t_i = c_i)$.

According to the conditions there exists a constant $c$ such that $B_c \in N$ and $\overline{B}_{t_i} = B_c$. By Reductio ad Absurdum suppose that $c \neq c_i$. Since $\overline{B}$ is a homomorphism we have that

$$\overline{N}_{t_i} = \overline{\beta}(\overline{B}_{t_i}) = \beta(B_c) = N_c = \overline{N}_c$$

hence $N_c = N_{c_i}$. It follows that $B_c = N_c = N_{c_i} = B_{c_i}$ which contradicts the conditions of the proposition. Thus $c = c_i$ which means $\overline{B}_{t_i} = B_{c_i}$ which means $\overline{B} \models (t_i = c_i)$.

Thus $\overline{B} \models H$. Since $(B, \xrightarrow{}_{\Gamma/B}) \models \Gamma$ it follows that $(\overline{B}, \xrightarrow{}_{\overline{\Gamma}/\overline{B}}) \models (t \rightarrow t')$ which means $\overline{B}_{t} \xrightarrow{}_{\overline{\Gamma}/\overline{B}} \overline{B}_{t'}$.

By Proposition 5.1 it follows that $\overline{\beta}(\overline{B}_t) \xrightarrow{}_{\overline{\Gamma}/\overline{B}} \overline{\beta}(\overline{B}_{t'})$. By the homomorphism property of $\overline{\beta}$ we have that $\overline{\beta}(\overline{B}_t) = \overline{N}_t$ and $\overline{\beta}(\overline{B}_{t'}) = \overline{N}_{t'}$ hence $\overline{N}_t \xrightarrow{}_{\overline{\Gamma}/\overline{B}} \overline{N}_{t'}$.

\[ \text{Corollary 5.6.} \text{ Let } q : B \rightarrow B' \text{ and } h : B' \rightarrow N \text{ be MSA homomorphisms. Let us assume that } \beta = h \circ q \text{ is an nf-homomorphism. Let } \Gamma \text{ be a set of transitions satisfying the conditions of Proposition 5.5. Then } h : (B', \xrightarrow{}_{\overline{\Gamma}/\overline{B'}}) \rightarrow (N, \xrightarrow{}_{\overline{\Gamma}/\overline{B}}) \text{ is a POA homomorphism.} \]

\[ \text{Proof.} \text{ By Proposition 5.5 we have that } (N, \xrightarrow{}_{\overline{\Gamma}/\overline{B}}) \models \Gamma. \text{ The conclusion follows by Theorem 4.4.} \]

\[ \text{Corollary 5.7 (The completeness of the concrete computational method). We assume the following conditions on } \Gamma: \]

- $\Gamma$ is $[\bot]$-coherent.
- For each sentence $\forall X \cdot H \Rightarrow (t \rightarrow t') \in \Gamma$, $H$ is a finite conjunction of equations of the form $t_i = c_i$ where $c_i$ is a constant such that $(c_i)_{E_0}$ is a normal form.
- For each sort $s$ of an equation $t_i = c_i$ that occurs in a condition of a transition in $\Gamma$ (like above):

$$0_{E_0} \models \forall x \cdot \bigvee \{x = c \mid c \in F_{\rightarrow s}, e_{E_0} = [c_{E_0}]\} \quad \text{(no junk)}$$

$$0_{E_0} \models \bigwedge \{c \neq c' \mid c \neq c' \in F_{\rightarrow s}, e_{E_0} = [c_{E_0}], e'_{E_0} = [c'_{E_0}]\} \quad \text{(no confusion)}$$

then $h(\xrightarrow{}_{\overline{\Gamma}/\overline{B}}) \subseteq \overline{\Gamma}/\overline{L}$.

\[ \text{Proof.} \text{ In Corollary 5.6, we set } B = 0_{E_0}, B' = 0_E, N = N_{E_1/E_0}, \beta = [\bot]. \]

Corollaries 5.4 and 5.7 together give the desired isomorphism (1). On the conditions underlying completeness. The situation of the specific conditions of Proposition 5.5, also propagated as conditions of Corollaries 5.6 and 5.7, remind us of the so-called minimax principle present in many areas of mathematics. From the mathematical side they appear quite restrictive, but from the applications side they appear quite permissive. Their great applicability comes from the treatment of conditions as Boolean terms. This is the only treatment of conditions available in CafeOBJ, and is available also in Maude.\(^{15}\) As already mentioned this approach has much greater expressivity power than

\(^{15}\)In both cases this was inherited from OBJ3.
the standard approach to conditions as conjunctions of atoms. In that approach a condition $H$ is an equation of the Boolean sort of the form $\tau = \top$ (true) or $\tau = \bot$ (false), where $\tau$ is a term of the Boolean sort. Then the ‘no junk’ condition says that the interpretations of $\top$ and $\bot$ are the only elements of $\mathcal{E}$ and the ‘no confusion’ condition say that these interpretations are different. In fact because the Boolean data type is always imported in ‘protected’ mode, for all correct programs this situation holds in all models. That $\top$ and $\bot$ are in normal form is trivial because we never place them in the lefthand sides of equations that we write in our programs, in fact in the programs we a priori think of them as playing roles of normal forms. The following example shows how this works in practice.

**Example 14.** Let us consider the benchmark example of BUBBLE-SORT. The condition of the transition is $(n < m) = \text{true}$ which is indeed of the form $\tau = \top$. Moreover in all BUBBLE-SORTS variants discussed in this paper the data type of the Booleans is of course ‘protected’ by the ‘no junk’ and ‘no confusion’ conditions.

There is the question regarding the apparent absence of the $H_-$ component from the conditions. In the Boolean terms approach the atomic transitions can be present in a coded format by employing for each system sort a predicate for the reachability relation. For instance CafeOBJ implements these; they are denoted $==>$. But Maude does not do this, which means that its approach to conditions is weaker.

In general the theory around Maude does not consider transitions conditioned by Boolean terms under the model theoretic constraint that the data type of the Booleans is protected. From that literature the closest to this comes the “generalised rules” of [26] that allows conditions to be arbitrary quantifier-free sentences.

With respect to the coherence conditions, the literature around Maude puts forward conditions for the completeness of the reachability computations that bear some similarity to our $\beta$-coherence, but they play a different role than in our work, of a mere syntactic nature. While in our approach the $\beta$-coherence condition represents a cause for the existence of the computational model as preordered algebra, in [26] this is not the case due to the primitive structure of their models.

6. **Compositionality of Algebraic Rewriting**

Large programs are built efficiently from smaller ones by using modularisation techniques such as module imports or parameterised modules. One of the established mathematical foundations for these is the pushout technique [4, 18, 17, 12, 14, 29], etc. In this section we develop a result on the compositionality of the algebraic rewriting relation in the context of the pushout technique for modularisation. This modularisation technique is by far best defined and developed within the institution theoretic framework. It relies on two rather technical concepts: model amalgamation and theory morphisms. Below we recall them briefly without directly involving any institution theory. Unless specified explicitly all definitions and results below apply to both MSA and POA.
Model amalgamation. A commutative square of signature morphisms like below:

\[
\begin{array}{ccc}
\Sigma & \xrightarrow{\varphi_1} & \Sigma_1 \\
\downarrow{\varphi_2} & & \downarrow{\theta_1} \\
\Sigma_2 & \xrightarrow{\theta_2} & \Sigma'
\end{array}
\]

is a **model amalgamation** square when

- for any \(\Sigma_k\)-models \(M_k\), \(k = 1, 2\), such that \(\varphi_1 M_1 = \varphi_2 M_2\) there exists an unique \(\Sigma'\)-model \(M'\) such that \(\theta_1 M' = M_k\), \(k = 1, 2\), and

- the same property like above when ‘model’ is replaced by ‘model homomorphism’.

The **MSA** version of the following result appears in many places in the algebraic specification literature, but its **POA** version is less known because **POA** is less known. However the **POA** version can also be found in some places such as [12].

**Proposition 6.1.** Any pushout square of signature morphisms is a model amalgamation square.

The concept of model amalgamation has many variations in the literature. Sometimes it considers only the condition on the models and drops that on homomorphisms, other times it drops the uniqueness requirement. Model amalgamation is a property that holds naturally in many contexts. For instance the model \(M'\) just puts together the interpretations of sorts, operations, preorders, provided by \(M_1\) and \(M_2\). In order for this to be possible it is necessary that the two collections of interpretations are mutually consistent, i.e. they share the same interpretations of the common symbols.

**Theory morphisms.** A **theory** is a pair \((\Sigma, E)\) that consists of a signature \(\Sigma\) and a set \(E\) of \(\Sigma\)-sentences. A \((\Sigma, E)\)-**model** is a \(\Sigma\)-model that satisfies \(E\). A **theory morphism** \(\varphi: (\Sigma, E) \rightarrow (\Sigma', E')\) is a signature morphism \(\varphi: \Sigma \rightarrow \Sigma'\) such that for each \((\Sigma', E')\)-model its \(\varphi\)-reduct is a \((\Sigma, E)\)-model.

**Fact 2.** The composition of theory morphisms yields a theory morphism.

Pushouts and model amalgamation can be lifted from signature morphisms to theory morphisms.

**Proposition 6.2.** Any span of theory morphisms has a pushout that inherits a pushout of the underlying span of signature morphisms.

**Proposition 6.3.** Any pushout square of theory morphisms is a model amalgamation square.

The last proposition lifts the model amalgamation property from signature morphisms to theory morphisms. This works at an abstract general level, not only for **MSA** and **POA**, but that generality requires the mathematical machinery of institution theory.

\[^{16}\text{For the abridged version of the POA.}\]
Amalgamation of rewriting. The result of this section, developed below, can be read in two ways. From the perspective of model amalgamation theory it says that

the amalgamation of algebraic rewriting relations yields an algebraic rewriting relation.

In other words, the amalgamation of the free POA models (as given by Theorem 4.4) yields a free POA model. But from an operational perspective it says that any algebraic rewriting of the result of putting together two sets of transitions happens in one of the components, provided that the component algebraic rewriting relations are mutually consistent.

Theorem 6.4. Consider a pushout square of POA signature morphisms like in the diagram below:

\[
\begin{array}{c}
(S, D, F) \xrightarrow{\varphi_1} (S_1, D_1, F_1) \\
\downarrow \varphi_2 \quad \downarrow \theta_1 \\
(S_2, D_2, F_2) \xrightarrow{\theta_2} (S', D', F')
\end{array}
\]

Let \( B' \) be any POA \((S', D', F')\)-algebra and let \( \Gamma_1, \Gamma_2 \) be sets of transitions for the signatures \((S_1, D_1, F_1)\) and \((S_2, D_2, F_2)\), respectively. Let \( \Gamma' = \theta_1 \Gamma_1 \cup \theta_2 \Gamma_2 \). If for each \( b, b' \in \varphi_1(\theta_1 B') = \varphi_2(\theta_2 B') \)

\[ b \xrightarrow{\Gamma_1/\theta_1 B'} b' \quad \text{if and only if} \quad b \xrightarrow{\Gamma_2/\theta_2 B'} b' \]

then

\[ \xrightarrow{\Gamma'/B'} = \xrightarrow{\Gamma_1/\theta_1 B'} \cup \xrightarrow{\Gamma_2/\theta_2 B'} \]

Proof. We have that

\[
\begin{array}{c}
((S, D, F), \emptyset) \xrightarrow{\varphi_1} ((S_1, D_1, F_1), \Gamma_1) \\
\downarrow \varphi_2 \quad \downarrow \theta_1 \\
((S_2, D_2, F_2), \Gamma_2) \xrightarrow{\theta_2} ((S', D', F'), \Gamma')
\end{array}
\]

is a pushout square in the category of the POA theory morphisms. Then we can apply Proposition 6.3 for POA. Hence the square above is a model amalgamation square.

Let \( B = \varphi_1(\theta_1 B') = \varphi_2(\theta_2 B') \). Then \((B, \leq)\) is a preordered algebra where \( \leq \) is defined by \( b \leq b' \) if and only if \( b \xrightarrow{\Gamma_1/\theta_1 B'} b' \). Moreover

\[ (B, \leq) = \varphi_1(\theta_1 B', \xrightarrow{\Gamma_1/\theta_1 B'}) = \varphi_2(\theta_2 B', \xrightarrow{\Gamma_2/\theta_2 B'}) \]

By the model amalgamation property for POA theory morphisms (Proposition 6.3) there exists in \((S', D', F')\) an amalgamation of \((\theta_1 B', \xrightarrow{\Gamma_1/\theta_1 B'})\) and \((\theta_2 B', \xrightarrow{\Gamma_2/\theta_2 B'})\). By the uniqueness of model amalgamation in MSA (Proposition 6.1) it follows that the underlying MSA algebra of the model amalgamation in POA is \( B' \), hence the model amalgamation in POA gives a preordered algebra \((B', \leq)\).

By the Satisfaction Lemma in POA it follows that \((B', \leq) \models \theta_k \Gamma_k\) hence \((B', \leq) \models \Gamma'\). Now we prove more, that \((B', \leq)\) is the free over the MSA algebra \( B' \).

Let \((A', \leq)\) be any POA \((S', D', F')\)-algebra such that \((A', \leq) \models \Gamma'\) and let \( h' : B' \to A' \) be an MSA homomorphism. By the Satisfaction Lemma in POA, for each \( k \in \{1, 2\} \)
we have that \( \theta_k(A', \preceq) \models \Gamma_k \). Thus by Theorem 4.4 it follows that \( \theta_k h' \) is a POA homomorphism \( (\theta_k B', \xrightarrow{\Gamma_k/\theta_k B'}) \rightarrow (A', \preceq) \). By the uniqueness of the model homomorphisms amalgamation property in POA (Proposition 6.1) we obtain that \( h' \) is a POA homomorphism \( (B', \preceq) \rightarrow (A', \preceq) \), which proves the freeness property of \( (B', \preceq) \). From this freeness it follows that the identity MSA homomorphism \( 1_{B'} : B' \rightarrow B' \) is an isomorphism \( (B', \preceq) \rightarrow (B', \preceq) \) which implies that \( \preceq = \overset{\Gamma'/B'}{\rightarrow} \). Thus \( (B', \overset{\Gamma'/B'}{\rightarrow}) \) is the amalgamation of \( (\theta_1 B', \overset{\Gamma_1/\theta_1 B'}{\rightarrow}) \) and \( (\theta_2 B', \overset{\Gamma_2/\theta_2 B'}{\rightarrow}) \) which implies the conclusion of the theorem. \( \square \)

This was about amalgamation of denotational models of algebraic rewriting. What about the amalgamation of computational models in the sense of the development of Section 5? This is a mathematically interesting problem but without much applications because in practice we are interested only in complete computational models, and those coincide (modulo isomorphisms) with the denotational models.

7. Conclusions and Future Research

In this paper we did the following:

(1) We defined in arbitrary MSA algebras \( B \) the rewriting relation \( \overset{\Gamma/B}{\rightarrow} \) determined by a set of conditional transitions \( \Gamma \).

(2) We proved that \( (B, \overset{\Gamma/B}{\rightarrow}) \) is the free preordered algebra that satisfies \( \Gamma \).

(3) We defined an abstract operational model for non-deterministic rewriting-based computations that captures the integration between the transition-based and the equational rewriting, both of them eventually modulo axioms. This operational model provides model theoretic foundations for the execution engines of Maude and CafeOBJ.

(4) Under a common coherence condition the abstract operational model is proper POA models (a preordered algebra) that is sound and complete with respect to the denotational preordered algebra of the respective POA theory as provided by the adjunction result.

(5) The completeness result relies on a set of specific conditions that apply well to the conditions-as-Boolean-terms approach of Maude and CafeOBJ. In the case of the latter language this is the only approach to conditional axioms.

(6) Finally, we developed a compositionality result for the rewriting relations \( \overset{\Gamma/B}{\rightarrow} \) within the context of the pushout-based technique for modularisation, which may serve as a foundation for an institution theoretic comprehensive approach to the modularisation of non-deterministic rewriting.

As with other algebraic specification/programming frameworks, algebraic non-deterministic rewriting can be refined by adding new features. Such an example is order-sortedness. Such enhancements can be topics of further developments in the area. With our compositionality result of Section 6 we just touched the great theme of modularisation. This can be further developed for algebraic non-deterministic rewriting by employing the corresponding institution theory applied to POA.
Acknowledgements. This work was supported by a grant of the Romanian Ministry of Education and Research, CNCS – UEFISCDI, project number PN-III-P4-ID-PCE-2020-0446, within PNCDI III. The reviewers have made some comments that helped to improve some aspects of the presentation.

REFERENCES

[1] Franz Baader and Tobias Nipkow. Term Rewriting and All That. Cambridge University Press, 1998.
[2] Yves Bertot and Pierre Castéran. Interactive Theorem Proving and Program Development (Coq’Art). Springer, 2004.
[3] Peter Borovansky, Claude Kirchner, Hélène Kirchner, Pierre-Etienne Moreau, and Christophe Ringeissen. An overview of ELAN. Electronic Notes in Theoretical Computer Science, 15:55–70, 1998.
[4] Rod Burstall and Joseph Goguen. The semantics of Clear, a specification language. In Dines Bjørner, editor, 1979 Copenhagen Winter School on Abstract Software Specification, volume 86 of Lecture Notes in Computer Science, pages 292–332. Springer, 1980.
[5] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. All About Maude - A High-Performance Logical Framework, volume 4350 of Lecture Notes in Computer Science. Springer, 2007.
[6] Mihai Codescu and Daniel Găină. Birkhoff completeness in institutions. Logica Universalis, 2(2):277–309, 2008.
[7] Mihai Codescu, Till Mossakowski, Adrián Riesco, and Christian Maeder. Integrating Maude into Hets. In Michael Johnson and Dusko Pavlovic, editors, Algebraic Methodology and Software Technology - 13th International Conference, AMAST 2010, Lac-Beauport, QC, Canada, June 23-25, 2010. Revised Selected Papers, volume 6486 of Lecture Notes in Computer Science, pages 60–75. Springer, 2010.
[8] Mihai Codescu, Till Mossakowski, Adrián Riesco, and Christian Maeder. Integrating Maude into Hets. Technical Report 07/10, Departamento de Sistemas Informáticos y Computación Universidad Complutense de Madrid, 2011.
[9] Thierry Coquand and Gerard Huet. The calculus of constructions. Information and Computation, 76(2/3):95–120, 1988.
[10] Răzvan Diaconescu. Institution-independent ultraproducts. Fundamenta Informaticæ, 55(3-4):321–348, 2003.
[11] Răzvan Diaconescu. Elementary diagrams in institutions. Journal of Logic and Computation, 14(5):651–674, 2004.
[12] Răzvan Diaconescu. Institution-independent Model Theory. Birkhäuser, 2008.
[13] Răzvan Diaconescu. Coinduction for preordered algebras. Information and Computation, 209(2):108–117, 2011.
[14] Răzvan Diaconescu and Ionuţ Tuţu. On the algebra of structured specifications. Theoretical Computer Science, 412(28):3145–3174, 2011.
[15] Răzvan Diaconescu and Kokichi Futatsugi. CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification, volume 6 of AMAST Series in Computing. World Scientific, 1998.
[16] Răzvan Diaconescu and Kokichi Futatsugi. Logical foundations of CafeOBJ. Theoretical Computer Science, 285:289–318, 2002.
[17] Răzvan Diaconescu, Joseph Goguen, and Petros Stefaneas. Logical support for modularisation. In Gerard Huet and Gordon Plotkin, editors, Logical Environments, pages 83–130. Cambridge, 1993. Proceedings of a Workshop held in Edinburgh, Scotland, May 1991.
[18] Joseph Goguen and Rod Burstall. Institutions: Abstract model theory for specification and programming. Journal of the Association for Computing Machinery, 39(1):95–146, 1992.
[19] Joseph Goguen and Răzvan Diaconescu. An Oxford survey of order sorted algebra. Mathematical Structures in Computer Science, 4(4):363–392, 1994.
[20] Joseph Goguen and Grigore Roșu. Institution morphisms. Formal Aspects of Computing, 13:274–307, 2002.
[21] Joseph Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. Introducing OBJ. In Joseph Goguen and Grant Malcolm, editors, Software Engineering with OBJ: algebraic specification in action. Kluwer, 2000.

[22] Daniel Găină and Marius Petria. Completeness by forcing. Journal of Logic and Computation, 20(6):1165–1186, 2010.

[23] Daniel Găină and Andrei Popescu. An institution-independent proof of Robinson consistency theorem. Studia Logica, 85(1):41–73, 2007.

[24] Simon Marlow. Haskell 2010 language report. https://www.haskell.org/definition/haskell2010.pdf, 2010.

[25] José Meseguer. Conditional rewriting logic as a unified model of concurrency. Theoretical Computer Science, 96(1):73–155, 1992.

[26] José Meseguer. Generalized rewrite theories, coherence completion, and symbolic methods. Journal of Logical and Algebraic Methods in Programming, 110:100483, 2020.

[27] Robin Milner, Mads Tøfte, and Robert Harper. Definition of standard ML. MIT Press, 1990.

[28] Till Mossakowski, Joseph Goguen, Răzvan Diaconescu, and Andrzej Tarlecki. What is a logic? In Jean-Yves Béziau, editor, Logica Universalis, pages 113–133. Birkhäuser, 2005.

[29] Donald Sannella and Andrzej Tarlecki. Foundations of Algebraic Specifications and Formal Software Development. Springer, 2012.

[30] A. van Deursen, T. B. Dinesh, and E. A. van der Meulen. The ASF+SDF Meta-environment. In M. Nivat, C. Rattray, T. Ras, and G. Scollo, editors, Algebraic Methodology and Software Technology (AMAST’93), pages 411–412. Springer London, 1994.

Simion Stoilow Institute of Mathematics of the Romanian Academy
Email address: Razvan.Diaconescu@ymail.com