Reducibility of operation symbols in term rewriting systems and its application to behavioral specifications

著者	Nakamura Masaki, Ogata Kazuhiro, Futatsugi Kokichi
journal or publication title	Journal of Symbolic Computation
volume	45
number	5
page range	551-573
year	2010-01-01
URL	http://hdl.handle.net/2297/22582

doi: 10.1016/j.jsc.2010.01.008

Reducibility of operation symbols in term rewriting systems and its application to behavioral specifications

Masaki Nakamura

Kanazawa University, Kakuma, Kanazawa, 920-1192, Japan

Kazuhiro Ogata and Kokichi Futatsugi

Japan Advanced Institute of Science and Technology, 1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan

Abstract

In this paper, we propose the notion of reducibility of symbols in term rewriting systems (TRSs). For a given algebraic specification, operation symbols can be classified on the basis of their denotations: the operation symbols for functions and those for constructors. In a model, each term constructed by using only constructors should denote an element, and functions are defined on sets formed by these elements. A term rewriting system provides operational semantics to an algebraic specification. Given a TRS, a term is called reducible if some rewrite rule can be applied to it. An irreducible term can be regarded as an answer in a sense. In this paper, we define the reducibility of *operation symbols* as follows: an operation symbol is reducible if any term containing the operation symbol is reducible. Non trivial properties on context-sensitive rewriting, which is a simple restriction of rewriting, can be obtained by restricting the terms on the basis of variable occurrences, its sort, etc. We confirm the usefulness of the reducibility of operation symbols by applying them to behavioral specifications for proving the behavioral coherence property.

Key words: Term rewriting system, Algebraic specification, Behavioral specification, Behavioral coherence, Observational transition system.

1. Introduction

A term rewriting system (TRS) is a set of rewrite rules. A term is constructed by using operation symbols and variables. For a given TRS R, a term is said to be an R-normal

⋆ A preliminary version of a part of this article has appeared in (Nakamura et al., 2005) (in Japanese).

form if no rewrite rule applies to the term. A non-(R-)normal form is said to be reducible (w.r.t. R). In a model, operation symbols are partitioned into constructor and defined symbols. For example, in a TRS for natural numbers, `0` and `s_` are operation symbols for constructing natural numbers (constructor symbols) and `_+_` and `_*_` are those for functions on natural numbers (defined symbols). Defined symbols are expected to be defined for all terms constructed by using only constructor symbols. In other words, any defined symbol is not expected to be included in any normal form. In this paper, we propose the notion of reducibility of operation symbols. An operation symbol f is reducible if any term containing f is reducible, i.e., is not a normal form. Without any restriction, the reducibility of an operation symbol f is not useful since in such a case, the following properties become equivalent: (1) f is reducible, and (2) there exists a rewrite rule $f(\vec{x}) \to r$ in the TRS such that x_i is a variable that is distinct from any other x_j [1]. Each term containing f can be rewritten by $f(\vec{x}) \to r$ ($1 \Leftarrow 2$). If f is reducible, then $f(\vec{y})$ should be reducible, and only the rewrite rule whose left-hand side is $f(\vec{x})$ can be applied to such a term ($1 \Rightarrow 2$).

By restricting the variable occurrences in input terms, sorts of input terms, etc, we propose different kinds of reducible operation symbols and provide some nontrivial properties pertaining to restrictions of rewriting to obtain efficient reduction. Our results can be applied to algebraic specification languages such as CafeOBJ and Maude [2]. Consider the following CafeOBJ specification `ZERO`:

```
mod! ZERO{
  pr(NAT+)
  op zero : Nat  -> Bool
  vars M N : Nat
  eq zero(0) = true .  eq zero(s N) = false .
  eq zero(M + N) = zero(M) and-also zero(N) .
}
```

where the function `zero` is defined for natural numbers. The function `zero` checks whether an input natural number is zero or not. More explanations of this specification can be found in Example 2 in Section 2.2. Let $t =$ `zero(`t_0 `+` t_1 `+` t_2`)` where t_0, t_1, and t_2 are very large natural numbers such as `s s` $\cdots$ `0`. In the innermost strategy, the argument subterm t_0 `+` t_1 `+` t_2 is first reduced to `s s s` $\cdots$ `0` by many rewrite steps, and then `zero(s s s` $\cdots$ `0)` is rewritten to `false` by using the equation `zero(s N) = false`. However, when we assume that only Boolean terms, i.e., `zero(`t`)`, are to be reduced, we can restrict the reduction of the argument of `zero` for obtaining a normal form. CafeOBJ supports the E-strategy (Futatsugi et al., 1985), in which we can flexibly restrict the rewriting of arguments of operation symbols. When restricting the argument of `zero`, the term `zero(`t_0 `+` t_1 `+` t_2`)` is first reduced to `zero(`t_0`) and-also zero(`t_1`) and-also zero(`t_2`)` by the last equation in `ZERO`. Next, the term is reduced to `false and-also false and-also false` by the equation `zero(s N) = false`, and then it is reduced to `false`. without the evaluation of t_0 `+` t_1 `+` t_2.

It is well known that context-sensitive rewriting (CSR) (Lucas, 1998) is a useful method to formalize a restriction of rewriting. The E-strategy can be regarded as an

[1] $\vec{a}$ is an abbreviation of $a_1, \ldots, a_n$ for some n.

[2] CafeOBJ official home page: `http://www.ldl.jaist.ac.jp/cafeobj/`
The Maude system: `http://maude.cs.uiuc.edu/`

implementation of CSR. In CSR, the rewriting of some arguments is restricted by a replacement map μ. In general, owing to the restriction, CSR provides a more efficient and terminating rewrite relation than the ordinary rewrite relation. However, a reducible term, i.e., a term containing a redex, may not be rewritten by CSR; that is, CSR may return a reducible term as a normal form of CSR called the μ-normal form. One of the motivation to conduct this study was to obtain sufficient conditions under which each μ-normal form is also an R-normal form. We term the property as μ-correctness. In other words, when CSR satisfies μ-correctness, the CSR does not return a reducible term as its normal form. The existing results for μ-normal forms deal with general terms (with no restrictions as described above), restrict a given TRS to a left-linear TRS, and guarantee that all μ-normal forms are head-normal forms, which are considered to be weaker conditions than R-normal forms. When applying these results to algebraic specifications, the algebraic specifications need to be left-linear TRSs and an output of the E-strategy reduction may not be in the R-normal form. We provide a solution to this problem. Our results restrict the input terms but do not restrict a given TRS and guarantee that any μ-normal form is an R-normal form. When reducing a term, we can select a suitable replacement map μ by analyzing the input term according to our results of μ-correctness and reduce the term under the replacement map μ. There are no restrictions on the input TRSs, and it is guaranteed that each output of the E-strategy reduction is an R-normal form.

The main contributions of this study are as follows:

Context-sensitive rewriting and μ-correctness: We propose the notion of μ-ground reducibility of operation symbols, which is defined as follows: any term containing μ-ground reducible operation symbols is reducible if an input term to be reduced satisfies the restriction that no variable appears in the path of particular arguments of particular operation symbols. We obtain a sufficient condition of μ-correctness under the above restriction of input terms. We also propose the notion of sort reducibility of operation symbols. For a set of sorts S', any term containing S'-sort reducible operation symbols is reducible if the sort of an input term is that of S'. We also obtain a sufficient condition of μ-correctness under the above restriction of input terms. Moreover, we give the notion of μ-ground S'-sort reducibility of operation symbols and obtain a sufficient condition of μ-correctness.

Sufficient completeness: We obtain a novel sufficient condition under which a given TRS is sufficiently complete by applying the notion of reducibility of operation symbols. Sufficient completeness is one of the most important properties of algebraic specifications (Guttag, 1975; Guttag and Horning, 1978). Let C be a set of constructor symbols. A TRS is sufficiently complete if each ground term can be reduced to a term constructed by using only constructor symbols. Roughly speaking, sufficient completeness guarantees that defined symbols are completely defined. Although sufficient completeness is generally undecidable, some restricted cases have been proposed in which sufficient completeness is decidable. The notion of ground reducibility is considered to be useful to obtain the decidability results (Jouannaud and Kounalis, 1986; Kapur et al., 1987). Our reducibility of operation symbols provides a generalization of ground reducibility. By combining the termination results of CSR, we obtain a new sufficient condition under which a given TRS is sufficiently complete.

Behavioral specification and behavioral coherence: As another application of the reducibility of operation symbols, we obtain a sufficient condition for behavioral coherence in behavioral specifications. Behavioral coherence is one of the most important properties of behavioral specifications (Diaconescu and Futatsugi, 1998, 2000). We obtain the sufficient condition by analyzing sorts that are declared as tight and protected and by restricting the form of terms in the arguments belonging to the sorts[3]. Our result can be applied to more specifications than the existing sufficient condition for behavioral coherence (Bidoit and Hennicker, 1999) implemented in CafeOBJ.

In Section 2, we introduce the syntax and semantics of the algebraic specification language CafeOBJ and fundamentals of TRSs. In Section 3, we propose a method to achieve the reducibility of operation symbols and describe some properties of CSR. In Section 4, we confirm that the reducibility of operation symbols is useful to prove behavioral coherence in behavioral specifications. In Section 5, we discuss some related studies and conclude the paper Section 6.

2. Preliminaries

We assume the reader is familiar with algebraic specifications and term rewriting systems (Diaconescu and Futatsugi, 1998; Ohlebusch, 2002; Terese, 2003).

2.1. *Order-sorted Algebraic specification*

The set of all natural numbers is denoted by $\mathcal{N}$. The set of all finite sequences over a set A is denoted by A^*. We may write an element of A^* like $a_1a_2a_3$ or $a_1.a_2.a_3$ where $a_i \in A$ ($i = 0, 1, 2$). For a set S, an S-sorted set A is a family $\{A_s \mid s \in S\}$ of sets sorted by S. For S-sorted sets A and B, an S-sorted map $m : A \to B$ is a family $\{m_s : A_s \to B_s \mid s \in S\}$ of maps sorted by S. We may omit the subscript s of A_s if no confusion arises, e.g. $a \in A$ instead of $a \in A_s$. A triple $(S, \leq, \Sigma)$ of a set S, an order $\leq$ on S and $S^* \times S$-sorted set Σ is called a signature. An element of S is called a sort and an element of $\Sigma_{w,s}$ is called an operation symbol. For an operation symbol $f \in \Sigma_{w,s}$, the sequence $w \in S^*$ is called its arity and the sort $s \in S$ is called its co-arity. If $n = 0$, we write $f \in \Sigma_s$ and call f a constant. We may abbreviate $(S, \leq, \Sigma)$ to Σ.

Let V be an S-sorted set which is distinct from Σ, i.e., $\Sigma_s \cap V_{s'} = \emptyset$ for each $s, s' \in S$. An S-sorted set $T(\Sigma, V)$ (abbr. T) of terms is defined as the smallest set satisfying the following: $V_s \subseteq T_s$, $T_{s'} \subseteq T_s$ if $s' \leq s$, and $f(\vec{t}) \in T_s$ if $f \in \Sigma_{\vec{s},s}$ and $t_i \in T_{s_i}$ for each s_i. If t belongs to a sort s, i.e., $t \in T_s$, we call t an s-sorted term. For a subset $S' \subseteq S$ of sorts, if there exists $s \in S'$ such that t is a s-sorted term, we call t an S'-sorted term. The set of all S'-sorted terms is denoted by $T_{S'}$. Hereafter, we often use s as an arbitrary sort, w as a sequence of sorts, f, g, h as operation symbols, x, y, z as variables, and t, u, l, r as terms. It is similar for a variety of them, like s', s_i, $\vec{s}$. A position of a term is indicated by a sequence of positive integers. $\mathcal{N}_+$ denotes $\mathcal{N} \setminus \{0\}$. For a term $t \in T$, the set $Pos(t) \subseteq \mathcal{N}_+^*$ of positions of t is defined as the smallest set satisfying the following: $Pos(t) = \{\varepsilon\}$ if $t \in V$ and $Pos(t) = \{\varepsilon\} \cup \{i.p \in \mathcal{N}_+^* \mid 1 \leq i \leq ar(f), p \in Pos(t_i)\}$ if $t = f(\vec{t})$ and $f \in \Sigma_{\vec{s},s}$, where ε is the empty sequence, and $ar(f)$ is the number of the arguments of f, i.e., $ar(f) = n$ if the arity of f is $s_1 s_2 \cdots s_n$. When $p = q.q'$ for some q', we write $p \geq q$. When $p \geq q$ and $p \neq q$, we write $p > q$. For example, $1.2.3.4 > 1.2$. A

[3] We say that a sort is tight and protected if it is declared in a tight specification and the specification is imported with the protect mode. Roughly speaking, a tight specification denotes the initial model, and a protected import preserves the model of sorts declared in the imported specification.

symbol of a term t at position p is denoted by $t(p)$, defined as $x(\varepsilon) = x$, $(f(\vec{t}))(\varepsilon) = f$ and $(f(t_1, \ldots, t_i, \ldots, t_n))(i.p) = t_i(p)$. The position ε is called the root position. The set of all variables included in t is denoted by $V(t)$. We say that t contains an operation symbol f when $t(p) = f$ for some $p \in Pos(t)$. The set of all terms which contain $f \in \Sigma$ is denoted by $T_f = \{t \in T \mid \exists p \in Pos(t).(t(p) = f)\}$. A subterm $t|_p$ of t at position p is defined as $t|_\varepsilon = t$ and $f(\vec{t})|_{i.p} = t_i|_p$. The set of all subterms of t is denoted by $Sub(t) = \{u \in T \mid p \in Pos(t), u = t|_p\}$. The result of replacing subterm $t|_p$ of t with u, denoted by $t[u]_p$, is defined as $t[u]_\varepsilon = u$ and $(f(\vec{t}))[u]_{i.p} = f(\ldots, t_{i-1}, t_i[u]_p, t_{i+1}, \ldots)$. The replacement is naturally generalized as $t[\vec{s}]_{\vec{p}}$ when the positions $\vec{p}$ are disjoint, that is, $p_i \not\geq p_j$ if $i \neq j$. We may omit the subscript p or $\vec{p}$ if no confusion arises. A term $C[z]_p$ with a marked variable z occurring only once in $C[z]_p$ is called a context. We often write C or $C[z]$ instead of $C[z]_p$. Hereafter, we use the variable z only for the marked variable of a context. An S-sorted map $\theta : V \to T$ is called a substitution. The result of replacing all variables x in t with $\theta(x)$ is denoted by $t\theta$. When $t = u\theta$ for some θ, we call t an instance of u. A term constructed by a constant symbol only is called a constant term or just a constant and denoted by c instead of $c()$. A variable-free term is called a ground term. The set $T(\Sigma, \emptyset)$ of ground terms is abbreviated to T_Σ.

An equation $(\forall X)l = r$ consists of an S-sorted set X of variables and terms $l, r \in T(\Sigma, X)_s$ belonging to a same sort s. When we omit the variable part and write $l = r$, it is an abbreviation of $(\forall X)l = r$ where $X = V(l) \cup V(r)$. An algebraic specification consists of a signature $(S, \leq, \Sigma)$ and a set E of equations constructed from Σ. For a specification SP, the signature and the set of the equations are denoted by $(S_{SP}, \leq_{SP}, \Sigma_{SP})$ and E_{SP} respectively. The congruence relation $=_E$ derived from equations in E is defined as the smallest equivalence relation (reflexive, symmetric and transitive relation) on T satisfying the substitutive law: $l\theta = r\theta$ for each $l = r \in E$ and θ, and the congruence law: $t_0 = t'_0, \ldots, t_n = t'_n$ implies $f(\vec{t}) = f(\vec{t'})$ for each $\vec{t}, \vec{t'} \in T$ and $f \in \Sigma$. We may omit the subscript E of $=_E$ if no confusion arises.

2.2. CafeOBJ specification

CafeOBJ is an algebraic specification language (Diaconescu and Futatsugi, 1998). In this paper, we deal with a part of CafeOBJ order-sorted equational specifications (in this section and Section 3) and behavioral specifications (in Section 4), do not deal with conditional equations, rewriting logic specifications, and so on. CafeOBJ specifications are built from modules. CafeOBJ modules are classified into those with tight denotation (`mod!` MOD $\{\cdots\}$) and those with loose denotation (`mod*` MOD $\{\cdots\}$). Each module consists of an import part, a signature part and an equation part. We regard a module as a specification which consists of all sorts, operation symbols and equations declared in the module and all modules imported by the module.

A basic module is a module with no imports.

Example 1. The following is an example of CafeOBJ basic module:
```
mod! NAT+{
  [Zero NzNat < Nat]
  op 0 : -> Zero
  op s_ : Nat -> NzNat
  op _+_ : Nat Nat -> Nat
  vars M N : Nat
  eq      0 + N = N .
  eq (s M) + N = s(M + N) .
}
```
The name of the module is $\texttt{NAT+}$. Sorts are declared between the square brackets `[ ]`. $S_{\texttt{NAT+}} = \{\texttt{Zero}, \texttt{NzNat}, \texttt{Nat}\}$. The order $\leq_{\texttt{NAT+}}$ on $S_{\texttt{NAT+}}$ is defined as the reflexive and transitive closure of the declared relation. Since $\texttt{Zero} < \texttt{Nat}$ and $\texttt{NzNat} < \texttt{Nat}$ are declared, $\leq_{\texttt{NAT+}}$ is $\{(\texttt{Zero}, \texttt{Zero}), (\texttt{Zero}, \texttt{Nat}), (\texttt{NzNat}, \texttt{NzNat}), (\texttt{NzNat}, \texttt{Nat}), (\texttt{Nat}, \texttt{Nat})\}$. Operation symbols are declared with the keyword `op`. The signature $\Sigma_{\texttt{NAT+}}$ is defined as $(\Sigma_{\texttt{NAT+}})_{\texttt{Zero}} = \{\texttt{0}\}$, $(\Sigma_{\texttt{NAT+}})_{\texttt{Nat},\texttt{NzNat}} = \{\texttt{s\_}\}$ and $(\Sigma_{\texttt{NAT+}})_{\texttt{Nat},\texttt{Nat},\texttt{Nat}} = \{\texttt{\_+\_}\}$. Note that the underlines in operation symbols indicate the positions of the arguments in their term expressions. For example, we can use the expression $(t\ \texttt{+}\ t')\ \texttt{+}\ t''$ instead of the expression $\texttt{\_+\_}(\texttt{\_+\_}(t,\ t'),\ t'')$. Equations are declared with `eq` where the variables used in an equation are declared before the declaration of the equation with `var` or `vars`. $E_{\texttt{NAT+}} = \{(\forall\{\texttt{N}\})\texttt{0 + N} = \texttt{N},\ (\forall\{\texttt{M},\texttt{N}\})\texttt{(s M) + N} = \texttt{s(M + N)}\}$. The equation $\texttt{s s 0 + s 0} = \texttt{s 0 + s s 0}$ can be derived from $E_{\texttt{NAT+}}$ since $\texttt{s s 0 + s 0} = \texttt{s (s 0 + s 0)} = \texttt{s s (0 + s 0)} = \texttt{s s s 0} = \texttt{s (0 + s s 0)} = \texttt{s 0 + s s 0}$. Each step can be derived by the substitutive law, the congruence law and the symmetric law.

A module can import other modules. There are three import modes: protecting, extending and using imports, denoted by `pr`, `ex` and `us`. Roughly speaking, a protecting import preserves the model of the imported module, an extending import can add an element to the model, and a using import can compress the model. For example, when declared `ex(NAT+)`, the sort `Nat` can be interpreted into the set of integers, etc. When declared `us(NAT+)`, the sort `Nat` can be interpreted into the quotient set $\{[0], [1], [2]\}$ of natural numbers modulo 3, where $1, 4, 7\ldots$ are compressed into $[1]$.

There is a special built-in module `BOOL` in CafeOBJ. `BOOL` is a module with tight denotation whose elements are the sort `Bool`, the constants `true` and `false`, the operation symbols `not_`, `_and_`, `_or_`, ... , and the equations defining those operation symbols, for example, `eq not false = true`. The reason why `BOOL` is special is because all CafeOBJ modules implicitly import `BOOL` with the protect mode[4].

Example 2. The following is an example of CafeOBJ module with imports:
```
mod! ZERO{
  pr(NAT+)
  op zero : Nat  -> Bool
  vars M N : Nat
  eq zero(0) = true .
  eq zero(s N) = false .
  eq zero(M + N) = zero(M) and-also zero(N) .
}
```
`ZERO` imports `NAT+` with the protect mode. Each element in `NAT+` is also an element of the specification `ZERO`, for example, $(\Sigma_{\texttt{NAT+}})_{w,s} \subseteq (\Sigma_{\texttt{ZERO}})_{w,s}$. The operation symbol `_and-also_` denotes the logical conjunction[5].

[4] Thus, strictly speaking, there is no basic module. We regard a module as a basic module if any element of `BOOL` is not used in the module (and in its execution).

[5] Although the denotations of `_and_` and `_and-also_` are same, they behave differently when executing specifications. When evaluating t_0 `and-also` t_1, the second argument t_1 may not be evaluated (lazy evaluation) unlike the case of evaluating t_0 `and` t_1 where both arguments are evaluated eagerly.

A model of an algebraic specification is an algebra. For a signature $(S, \leq, \Sigma)$, a $(S, \leq, \Sigma)$-algebra M consists of (1) carrier sets M_s for all $s \in S$ which satisfy that $s \leq s'$ implies $M_s \subseteq M_{s'}$ and (2) functions $M_f : M_w \to M_s$ for all operation symbols $f \in \Sigma_{w,s}$, where $M_{s_1 s_2 \cdots s_n}$ stands for $M_{s_1} \times M_{s_2} \times \cdots \times M_{s_n}$. We use M as an arbitrary Σ-algebra. For a Σ-algebra M and an S-sorted set V of variables, an S-sorted map $a : V \to M$ is called an assignment, and is naturally extended to an S-sorted map $a : T(\Sigma, V) \to M$ with $a(f(t_1, \ldots, t_n)) = M_f(a(t_1), \ldots, a(t_n))$. A Σ-algebra M satisfies an equation $t = t'$ if $a(t) = a(t')$ for any assignment $a : V \to M$. A term $t \in T(\Sigma, V)_s$ is interpreted into a function $M_t : M_{\vec{s}} \to M_s$ where $V(t) = \{x_1, x_2, \ldots, x_n\}$ and $x_i \in V_{s_i}$ for each x_i. Notice that a ground term $t \in T_s$ is interpreted into an element $M_t \in M_s$. For a specification SP, an SP-algebra is defined as a Σ_{SP}-algebra M satisfying all equations in E_{SP}. For Σ-algebras M and M', a Σ-morphism $h : M \to M'$ is an S-sorted map from the carrier sets of M to those of M' which satisfies that $h_s(M_f(\vec{a})) = M'_f(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$ for each $f \in \Sigma_{\vec{s},s}$ and $a_i \in M_{s_i}$ $(i = 1, \ldots, n)$. An initial (SP-)algebra is an SP-algebra I satisfying that for any SP-algebra M there exists a unique Σ-morphism $h : I \to M$. All initial algebras I satisfy the following conditions: For any $e \in I_s$, there exists $t \in (T_\Sigma)_s$ such that $I_t = e$ (no junk). For any $t, t' \in T_\Sigma$, if $t \neq_E t'$ then $I_t \neq I_{t'}$ (no confusion). The term algebra $\mathcal{T}$ is an initial algebra, which defined as follows: Let (Σ, E) a specification. $\mathcal{T}_s = (T_\Sigma)_s/_{=_E}$ and $\mathcal{T}_f(\overrightarrow{[t]}) = [f(\vec{t})]$ [6] . The notation for the denotation of a CafeOBJ basic module MOD is $[MOD]$, defined as follows: $[MOD]$ is the set of all MOD-algebras if MOD is loose (`mod*`) and $[MOD]$ is the set of all initial MOD-algebras if MOD is tight (`mod!`).

Example 3. Let N, N', B, Z be $\Sigma_{\texttt{NAT+}}$-algebras defined as follows:

- $N_{\texttt{Zero}} = \{0\}$, $N_{\texttt{NzNat}} = \mathcal{N} \backslash \{0\}$, $N_{\texttt{Nat}} = \mathcal{N}$, $N_0 = 0$, $N_{\texttt{s}}(x) = x+1$, and $N_{\texttt{+}}(x, y) = x+y$.
- N' is same with N except $N'_{\texttt{+}}(x, y) = x \times y$.
- Z is same with N except $Z_{\texttt{Nat}}$ is the set of all integers.
- $B_{\texttt{Zero}} = \{false\}$, $B_{\texttt{NzNat}} = \{true\}$, $B_{\texttt{Nat}} = \{true, false\}$, $B_0 = false$, $B_{\texttt{s}}(x) = true$, and $B_{\texttt{+}}(x, y) = x \vee y$.

All algebras except N' are `NAT+`-algebras, i.e., they satisfy all equations in $E_{\texttt{NAT+}}$. The Σ-algebra N satisfies `(s M) + N = s(M + N)` as follows: $a(\texttt{(s M) + N}) = N_{\texttt{+}}(N_{\texttt{s}}(a(\texttt{M})), a(\texttt{N}))$ $= (a(\texttt{M}) + 1) + a(\texttt{N}) = (a(\texttt{M}) + a(\texttt{N})) + 1 = N_{\texttt{s}}(N_{\texttt{+}}(a(\texttt{M}), a(\texttt{N}))) = a(\texttt{s}(\texttt{M} + \texttt{N}))$ for any assignment $a : V \to N$. The Σ-algebra B satisfies `0 + N = N` as follows: $a'(\texttt{0 + N}) = B_{\texttt{+}}(B_0, a'(\texttt{N})) = false \vee a'(\texttt{N}) = a'(\texttt{N})$ for any $a' : V \to B$. The Σ-algebra N' does not satisfy `0 + N = N` for the assignment $a'' : V \to N'$ such that $a''(\texttt{N}) = 1$: $a''(\texttt{0 + N})$ $= N'_{\texttt{+}}(N'_0, a''(\texttt{N})) = 0 \times 1 = 0 \neq a''(\texttt{N})$. Only N is an initial algebra. For any $n \in \mathcal{N}$, there exists $\texttt{s}^n(0) \in T_\Sigma$ (no junk). If $t \neq t'$ for given $t, t' \in T_\Sigma$, they should satisfy that $t = \texttt{s}^n(0)$, $t' = \texttt{s}^m(0)$ and $n \neq m$ (no confusion). Z has a junk since there is no term t such that $Z_t = -1$. B confuses `s s 0` and `s 0` since $B_{\texttt{s s 0}} = true = B_{\texttt{s 0}}$. Thus, $N \in [\texttt{NAT+}]$ and $Z, B \notin [\texttt{NAT+}]$.

[6] $T_\Sigma/_=$ is the quotient set of T_Σ by $=$, $[t]$ denotes the equivalence class including t. Since $[t]$ is the equivalence class, the function $\mathcal{T}_f$ is well-defined.

A CafeOBJ module MOD with imports denotes MOD-algebras which satisfy the import modes. Denotation for CafeOBJ modules with imports can be found in (Diaconescu and Futatsugi, 1998). We only introduce the properties related to our study. SP' is a sub specification of SP if $S_{SP'} \subseteq S_{SP}$, $\leq_{SP'} \subseteq \leq_{SP}$, $\Sigma_{SP'} \subseteq \Sigma_{SP}$ and $E_{SP'} \subseteq E_{SP}$. For an SP-algebra M, the restricted algebra that ignores sorts and operation symbols not included in a sub specification SP' is denoted by $M \upharpoonright_{SP'}$. For a module MOD with imports, $[MOD]$ satisfies the following import conditions: if MOD imports MOD' with the protect mode, i.e., $\texttt{pr(}MOD'\texttt{)}$, then for any $M \in [MOD]$ and $s \in S_{MOD'}$, there exists $M' \in [MOD']$ such that $M' = M \upharpoonright_{MOD'}$. If $\texttt{ex(}MOD'\texttt{)}$ then for any $M \in [MOD]$ and $s \in S_{MOD'}$, there exists a Σ-morphism $h : M' \to M \upharpoonright_{MOD'}$ such that h is inclusive, i.e., M' is a sub algebra of $M \upharpoonright_{MOD'}$. If $\texttt{us(}MOD'\texttt{)}$ then for any $M \in [MOD]$ and $s \in S_{MOD'}$, there exists an arbitrary Σ-morphism $h : M' \to M \upharpoonright_{MOD'}$.

Example 4. Let N and B be $\Sigma_{\texttt{NAT+}}$-algebra and $\Sigma_{\texttt{BOOL}}$-algebra defined as follows:
- $N_{\texttt{Nat}} = \mathcal{N}$, $N_{\texttt{0}} = 0$, $N_{\texttt{s}}(x) = x + 1$, and $N_{\texttt{+}}(x, y) = x + y$.
- $B_{\texttt{Bool}} = \{true, false\}$, $B_{\texttt{false}} = false$, $B_{\texttt{true}} = true$, $B_{\texttt{and-also}}(x, y) = x \wedge y$, ….

Then, $N \in [\texttt{NAT+}]$ and $B \in [\texttt{BOOL}]$. The following $\Sigma_{\texttt{ZERO}}$-algebra M is included in $[\texttt{ZERO}]$:
- $M_s = N_s$ for all $s \in S_{\texttt{NAT+}}$,
- $M_s = B_s$ for all $s \in S_{\texttt{BOOL}}$,
- $M_f = N_f$ for all $f \in \Sigma_{\texttt{NAT+}}$,
- $M_f = B_f$ for all $f \in \Sigma_{\texttt{BOOL}}$ and
- if $x = 0$ then $M_{\texttt{zero}}(x) = true$, otherwise $M_{\texttt{zero}}(x) = false$.

2.4. Term rewriting systems

A term rewriting system (TRS) gives us a powerful method for equational reasoning. In TRS, bidirectional equations $l = r$ are regarded as directional rewrite rules $l \to r$. A term is reduced by applying the rewrite rules repeatedly. An equation $t =_E u$ is proved by checking whether the terms t' and u' reduced from both sides are identical or not. A TRS (Σ, R) consists of a signature and a set of rewrite rules. R corresponds to the set E of equations. A rewrite rule $l \to r$ is a pair of terms (l, r) which satisfy that $l \notin V$ and $V(r) \subseteq V(l)$. In this paper, we regard a specification SP as a TRS where all equations in SP are regarded as left-to-right rewrite rules. We assume that each specification satisfies the above TRS conditions. We may write R as a TRS instead of (Σ, R) if Σ is the set of all operation symbols in R. For a TRS R and a position p, the rewrite relations $\to_p$ and $\to_R$ are defined as follows:

$$t \to_p u \overset{\text{def}}{\Longleftrightarrow} \exists l \to r \in R, \theta : V \to T.\, (t|_p = l\theta, u = t[r\theta]_p)\,,$$
$$t \to_R u \overset{\text{def}}{\Longleftrightarrow} \exists p \in Pos(t).\, (t \to_p u)\,.$$

For a binary relation $\to$, the reflexive and transitive closure is denoted by $\to^*$. If there is no u such that $t \to u$, we call t a $\to$-normal form. We may omit "$\to$-" if no confusion arises. We say a term t is reduced to u by $\to$ if there exists $n \in \mathcal{N}$ such that $t = t_0$, $u = t_n$ and $t_i \to t_{i+1}$ for any $i \in \{0..n-1\}$ [7]. A $\to_R$-normal form is called an

[7] We denote the set $\{m, m+1, m+2, \ldots, n-2, n-1, n\}$ of sequential numbers from m to n by $\{m..n\}$. If $m \geq n$, $\{m..n\} = \emptyset$.

R-normal form. The set of all R-normal forms is denoted by NF_R. An instance $l\theta$ of the left-hand side of a rewrite rule $l \to r$ is called a redex. The followings are equivalent: (1) t has a redex, (2) t is reducible and (3) $t \notin NF_R$. The set of all redexes is denoted by $Red(R) = \{l\theta \in T \mid l \to r \in R, \theta \in V \to T\}$.

Example 5. Consider the TRS `NAT+`. By the first rewrite rule with the substitution $\theta(N) = \mathtt{s(0)}$, the term `0 + (s 0)` is a redex, and thus $t =$ `(s s 0) + ((s 0) + 0)` $\to_2$ `(s s 0) + s 0` holds. The term t can be reduced to the normal form `s s s 0` as follows: $t \to_2$ `(s s 0) + s (0 + 0)` $\to_{2.1}$ `(s s 0) + s 0` $\to_\varepsilon$ `s ((s 0) + s 0)` $\to_1$ `s s (0 + s 0)` $\to_{1.1}$ `s s s 0`. This reduction can be regarded as a proof of $2 + (1 + 0) = 3$.

The CafeOBJ system supports a rewriting engine based on TRSs. The following is the experimental result of applying the CafeOBJ reduction command to the term `(s s 0) + ((s 0) + 0)`:

```
-- reduce in NAT+ : ((s (s 0)) + ((s 0) + 0)):Nat
(s (s (s 0))):NzNat
```

As we expected in Example 5, the CafeOBJ system reduces it into `s s s 0`.

3. Reducible operation symbols

The notion of reducibility of operation symbols is defined as follows: an operation symbol f is reducible if any term t which contains f, i.e., $t \in T_f$, is reducible. As we discussed in Section 1, the following statements are equivalent: (1) f is reducible and (2) there exists $f(\vec{x}) \to r \in R$. The arguments of reducible operation symbols are not needed to be rewritten. Let $R = \{id(x) \to x\}$. Then, id is reducible. The term $id(id(0))$ can be reduced to 0 by the innermost strategy (Ohlebusch, 2002; Terese, 2003) like $id(\underline{id(0)}) \to_R \underline{id(0)} \to_R 0$, where the underlined subterms are rewritten redexes. Even if we restrict the rewriting of arguments of id, we can also obtain the normal form 0 as follows: $\underline{id(id(0))} \to_R \underline{id(0)} \to_R 0$. Such a restriction of arguments can be formalized by context-sensitive rewriting (Lucas, 1998).

3.1. Context-sensitive rewriting

Context-sensitive rewriting (CSR) is a restriction of rewriting formalized by a replacement map on operation symbols (Lucas, 1998). A replacement map is a map $\mu : \Sigma \to \mathcal{P}(\mathcal{N}_+)$ which satisfies that $\forall f \in \Sigma. (\mu(f) \subseteq \{1..ar(f)\})$. Intuitively, in CSR, an argument t_i of $f(\vec{t})$ can be rewritten only if $i \in \mu(f)$. We say $\mu(f)$ is trivial if $\mu(f) = \{1..ar(f)\}$. The set $Pos_\mu(t)$ of replacement positions of a term t is recursively defined as follows: $Pos_\mu(t) = \{\varepsilon\}$ if $t \in V$ and $Pos_\mu(t) = \{\varepsilon\} \cup \{i.p \in \mathcal{N}_+^* \mid i \in \mu(f), p \in Pos_\mu(t_i)\}$ if $t = f(\vec{t})$. The CSR relation (or μ-rewrite relation) $\to_\mu$ is defined as follows: $t \to_\mu u \overset{\text{def}}{\Longleftrightarrow} \exists p \in Pos_\mu(t), t \to_p u$. We call a $\to_\mu$-normal form a μ-normal form. The set of all μ-normal forms is denoted by NF_μ. Trivially, $Pos_\mu(t) \subseteq Pos(t)$ and any R-normal form is a μ-normal form, i.e., $NF_R \subseteq NF_\mu$. Hereafter, when we define a replacement map for a subset $\{f_1, f_2, \ldots, f_n\} \subseteq \Sigma$ of operation symbols like $\mu(f_1) = L_1, \mu(f_2) = L_2, \ldots, \mu(f_n) = L_n$, we implicitly assume that $\mu(f)$ is trivial for the other operation symbols, i.e., $\mu(f) = \{1..ar(f)\}$ for each $f \in \Sigma \setminus \{f_1, f_2, \ldots, f_n\}$.

Example 6. Consider `NAT+` again. Let μ be a replacement map such that $\mu(\texttt{+}) = \{1\}$. Then, $Pos_\mu(\texttt{(s 0) + (0 + 0)}) = \{\varepsilon, 1, 1.1\}$. Thus, $\underline{\texttt{(s 0) + (0 + 0)}} \to_\mu$ `s (0 + (0 + 0))` but `(s 0) +` $\underline{\texttt{(0 + 0)}} \not\to_\mu$ `(s 0) + 0`.

The CafeOBJ reduction command reduces terms according to the E-strategy (Futatsugi et al., 1985) which supports a restriction of rewriting. For each operation symbol f, we can give a local strategy like `strat: (3 1 0)`, which means that for a term $f(t_1, t_2, t_3)$, the third t_3 is reduced first, the first t_1 is reduced next, and then rewrite rules are tried to be applied to the root position. The second argument t_2 is not to be reduced while under f. Thus, if we give a local strategy of f as `strat: (`$i_1\ i_2\ \cdots\ i_n$ `0)`, then the reduction command reduces terms according to the μ-rewrite relation $\to_\mu$ defined as $\mu(f) = \{i_1, i_2, \ldots, i_n\}$. The following experimental result shows the trace of applying the CafeOBJ reduction command to the term `(s s 0) + ((s 0) + 0)` with the local strategy `(1 0)` for _+_:

```
-- reduce in NAT+ : ((s (s 0)) + ((s 0) + 0)):Nat
[1]: ((s (s 0)) + ((s 0) + 0)) ---> (s ((s 0) + ((s 0) + 0)))
[2]: (s ((s 0) + ((s 0) + 0))) ---> (s (s (0 + ((s 0) + 0))))
[3]: (s (s (0 + ((s 0) + 0)))) ---> (s (s ((s 0) + 0)))
[4]: (s (s ((s 0) + 0))) ---> (s (s (s (0 + 0))))
[5]: (s (s (s (0 + 0)))) ---> (s (s (s 0)))
(s (s (s 0))):NzNat
```

The subterm `(s 0) + 0` of the input term is not rewritten in `[1]`, `[2]` and `[3]` since it is under the second argument of `+`. In `[4]` and `[5]`, the subterm is reduced to `s 0` since it is under `s`. Thanks to the restriction, a μ-normal form may not be an R-normal form, i.e., $NF_R \neq NF_\mu$ in general. For example, the reducible term `(0 + 0) + 0` is a μ-normal form when $\mu(\texttt{+}) = \emptyset$. Besides the termination and confluence properties, the properties with respect to the μ-normal forms are important in CSR. In this paper, we call a replacement map correct if $NF_\mu = NF_R$, and give sufficient conditions for several kinds of correctness. The correctness is useful when the μ-rewrite relation is terminating since it guarantees the existence of a normal form for each term. See Section 5.2 for more discussion.

3.2. Reducible operation symbols

It is trivial that μ is correct when $\mu(f)$ is trivial for each $f \in \Sigma$. One of our purposes is to give sufficient conditions under which μ is correct even if $\mu(f)$ is not trivial for some $f \in \Sigma$. The following properties hold trivially.

Proposition 7. *Let μ be a replacement map. If for each operation symbol $f \in \Sigma$ we have that f is reducible or $\mu(f)$ is trivial, then no μ-normal form contains reducible operation symbols.*

Proof. Assume that $t \in NF_\mu$ contains a reducible operation symbol, and the position p is the smallest position such that the operation symbol $t(p)$ is reducible, i.e., $t(q)$ is not reducible for each $q < p$. Let $f = t(p)$. As we discussed in Section 1, there exists $f(\vec{x}) \to r \in R$, and $t|_p$ is a redex. Since $\mu(g)$ is trivial for non-reducible operation symbols g, we have that $p \in Pos_\mu(t)$. It contradicts $t \in NF_\mu$. $\square$

If $\mu(f)$ is trivial for all operation symbols in a μ-normal form t, the term t is also an R-normal form since all position $p \in Pos(t)$ can be replaced, i.e., $p \in Pos_\mu(t)$. Therefore, if f is reducible or $\mu(f)$ is trivial for each operation symbol $f \in \Sigma$, μ is correct.

Example 8. A TRS is called a recursive program scheme (RPS) if each left-hand sides forms $f(\vec{x})$. From the above discussion, every replacement map μ can be correct for a given RPS R.

The reducibility of operation symbols for general terms is not so useful since it is equivalent to the existence of $f(\vec{x}) \to r \in R$. In the following sections, by restricting the form of terms to be reduced, we give more useful definitions of reducible operation symbols.

3.3. Ground reducible operation symbols

A term t is ground reducible if any ground instance of t is reducible (Jouannaud and Kounalis, 1986). Reducible terms are trivially ground reducible. For `NAT+` in Example 1 and variables x, y, the term x `+` y is not reducible but is ground reducible. Let the notion of ground reducibility of operation symbols define as follows: f is ground reducible if any term $t \in T_f$ is ground reducible, that is, any ground term $t \in T_f \cap T_\Sigma$ is reducible. Then, `_+_` is ground reducible. We can relax the restriction of ground terms. Consider a substitution which makes x `+` y is reducible again. In order to obtain a reducible term, it is enough to instantiate a variable x by a ground term. Variable y does not need any instantiation. In the other word, the term t `+` t' is reducible if t is a ground term. Moreover, t is not needed to be ground. For example, t `+` t' is reducible even if $t =$ `s 0` `+` x' where $x' \in V$. We formalize a notion of μ-ground terms.

Definition 9. Let μ be a replacement map. The S-sorted set GT^μ of μ-ground terms is the smallest set satisfying the following conditions: a constant is μ-ground, i.e., $\Sigma_s \subseteq GT^\mu_s$ and a term $f(\vec{t})$ is μ-ground if t_i is μ-ground for each $i \in \mu(f)$.

The term (`0` `+` x) `+` y is not ground but is μ-ground for $\mu($`+`$) = \{1\}$. We define the notion of μ-ground reducibility which is a generalization of the above definition of ground reducibility.

Definition 10. Let μ be a replacement map and $f \in \Sigma$. We say that f is μ-ground reducible if every term $t \in T_f \cap GT^\mu$ is reducible.

The following theorem makes it easier to prove the μ-ground reducibility of an operation symbol.

Theorem 11. *Let $f \in \Sigma$. The following statements are equivalent:*
(1) f is μ-ground reducible.
(2) $f(\vec{t})$ is a redex if t_i is a μ-ground μ-normal form for each $i \in \mu(f)$.

Proof. ($1 \Rightarrow 2$) Assume $t_i \in GT^\mu \cap NF_\mu$ for each $i \in \mu(f)$. From (1), $f(\vec{t})$ is reducible. From the definition of the replacement map, if $t_i \in NF_\mu$ for each $i \in \mu(f)$, $f(\vec{t})$ is a redex.
($1 \Leftarrow 2$) We prove that $t \in T_f \cap GT^\mu$ is reducible. From the definition of T_f and GT^μ, there exists $p \in Pos(t)$ such that $t|_p = f(\vec{t})$ and t_i is μ-ground for each $i \in \mu(f)$. If there

exists $i \in \mu(f)$ such that t_i is not a μ-normal form, then t is reducible. Otherwise, $f(\vec{t})$ is a redex from the assumption (2), and t is reducible. $\square$

By the notion of μ-ground reducibility, we give a non-trivial sufficient condition under which any μ-ground μ-normal form is an R-normal form. The property is called μ-ground correctness.

Definition 12. A replacement map μ is μ-ground correct if $NF_\mu \cap GT^\mu \subseteq NF_R$.

Lemma 13. *Let μ be a replacement map. If for each operation symbol $f \in \Sigma$ we have that f is μ-ground reducible or $\mu(f)$ is trivial, then no μ-ground μ-normal form contains any μ-ground reducible operation symbol.*

Proof. We prove the claim with proof by contradiction. Assume that $t \in NF_\mu \cap GT^\mu$ contains a μ-ground reducible operation symbol f and let p be one of the smallest position such that the operation symbol $t(p)$ is μ-ground reducible, i.e., $t(q)$ is not μ-ground reducible for each $q < p$. Let $t|_p = f(\vec{t})$. $p \in Pos_\mu(t)$ holds since $\mu(t(q))$ is trivial for each $q < p$. Let $i \in \mu(f)$. From the definition of the replacement map and the μ-rewrite relation, $p.i \in Pos_\mu(t)$ and t_i is a μ-normal form since t is a μ-normal form. By Theorem 11, $f(\vec{t})$ is a redex. Since $p \in Pos_\mu(t)$, this contradicts that $t \in NF_\mu$. $\square$

Theorem 14. *Let μ be a replacement map. If for each operation symbol $f \in \Sigma$ we have that f is μ-ground reducible or $\mu(f)$ is trivial, then μ is μ-ground correct.*

Proof. From Lemma 13. $\square$

Example 15. Consider `NAT+` again. Let $\mu(\texttt{+}) = \{1\}$. Any μ-ground μ-normal form is in the form of $\texttt{s}^n(0)$. Any pattern $\texttt{s}^n(0)$ `+` t is a redex. From Theorem 11, the operation symbol `_+_` is a μ-ground reducible. From Theorem 14, μ is μ-ground correct.

3.4. Sort reducible operation symbols

In this section, we propose the notion of reducibility of operation symbols for terms of sort $s \in S'$ for a given set $S' \subseteq S$ of sorts. Consider `ZERO` in Example 2. The operation symbol `zero` is reducible for ground terms, i.e., ground reducible, since it is defined for all patterns constructed from `0`, `s_` and `_+_`. On the other hand, the operation symbols `0`, `s_` and `_+_` can be regarded as reducible when we consider terms of the sort `Bool`. Although `0`, `s(`x`)` and x `+` y are not redexes themselves, they should be a part of a redex in a `Bool`-sorted term like `zero(0)`, `zero(s(`t`))` and `zero(`t `+` t'`)` respectively. Thus, for a replacement map μ satisfying that $\mu(0) = \mu(s) = \mu(\texttt{+}) = \emptyset$, any `Bool`-sorted μ-normal form is an R-normal form.

Definition 16. Let $S' \subseteq S$ be a set of sorts. An operation symbol f is S'-sort reducible if $t \in T_{S'} \cap T_f$ is reducible.

We introduce the cut function cut_μ for a replacement map μ, which replaces all maximal non-μ-replacing subterms of a given term with distinct fresh variables. The function cut'_μ in the following definition is same with the maximal replacing context defined in the literature (Lucas, 2002).

Definition 17. (Lucas, 2002) The auxiliary function cut'_μ is defined as follows: $cut'_\mu(x) = x$ for each $x \in V$ and $cut'_\mu(f(\vec{t})) = f(\vec{t'})$ where $t'_i = cut'_\mu(t_i)$ for each $i \in \mu(f)$ and $t_i = \Box$ for each $i \notin \mu(f)$ where $\Box$ is a special constant. The function $cut_\mu(t)$ is defined as the result of replacing all occurrences of $\Box$ in $cut'_\mu(t)$ with distinct fresh variables.

Let $t =$ `(s 0 + 0) + (0 + s 0)` and $\mu(\texttt{+}) = \{1\}$, for example. Then, $cut'_\mu(t) =$ `(s 0 +` $\Box$`) +` $\Box$ and $cut_\mu(t) =$ `(s 0 +` x`) +` y.

Lemma 18. *If t is a μ-normal form, $cut_\mu(t)$ is an R-normal form.*

Proof. Assume $t' = cut_\mu(t)$ is not an R-normal form. Since the cut ends are distinct fresh variables, there exists a substitution θ such that $t'\theta = t$. Let $p \in Pos(t')$ be a redex position, i.e., $t'|_p = l\theta'$ for a substitution θ' and $l \to r \in R$. Since all non-replaceable positions are replaced with variables and each variable is not a redex, the position p is replaceable, i.e., $p \in Pos_\mu(t')$. Since $t(q) = t'(q)$ for any $q < p$, $p \in Pos_\mu(t')$ implies $p \in Pos_\mu(t)$. The following equation holds: $t|_p = t'\theta|_p = (t'|_p)\theta = (l\theta')\theta = l(\theta;\theta')$. Therefore, t is not a μ-normal form. $\Box$

By using the cut function, we show that a replacement map can be correct for S'-sorted terms even if the rewriting of arguments of S'-sort reducible operation symbols is restricted.

Lemma 19. *Let μ be a replacement map. If for each operation symbol $f \in \Sigma$ we have that f is S'-sort reducible or $\mu(f)$ is trivial then no S'-sorted μ-normal form contains any S'-sort reducible operation symbol.*

Proof. Assume that $t \in NF_\mu \cap T_{S'}$ contains a S'-sort reducible operation symbol, and a position p is one of the smallest position such that $t(p) = f$ is S'-sort reducible. $p \in Pos_\mu(t)$ holds since $\mu(t(q))$ is trivial for each $q < p$. From Definition 17, $cut_\mu(t)$ contains f and is in $T_{S'}$. From Definition 16, $cut_\mu(t)$ is reducible. It contradicts $t \in NF_\mu$ and Lemma 18. $\Box$

Definition 20. Let $S' \subseteq S$ be a set of sorts. A replacement map μ is correct on S' if $NF_\mu \cap T_{S'} \subseteq NF_R$.

Theorem 21. *Let μ be a replacement map. If for each operation symbol $f \in \Sigma$ we have that f is S'-sort reducible or $\mu(f)$ is trivial, then μ is correct on S'.*

Proof. From Lemma 19. $\Box$

Example 22. Consider `ZERO`. The operation symbols `0`, `s`, `+` are {`Bool`}-sort reducible. Even if $\mu(\texttt{0}) = \mu(\texttt{s}) = \mu(\texttt{+}) = \emptyset$, the replacement map μ is correct on {`Bool`} from Theorem 21. Moreover, `zero`, `and-also` are μ-ground reducible. Therefore, no `Bool`-sorted ground μ-normal form contains those operation symbols, that is, it should be `true` or `false`. The following is the experimental result of reducing the term $t =$ `zero(s s s s s 0 + (s s s s 0 + (s s s 0 + (s s 0 + (s 0)))))` in a normal strategy, that is, $\mu(f)$ is trivial for each $f \in \Sigma_{\texttt{ZERO}}$:

```
-- reduce in ZERO : (zero(((s (s (s (s (s 0))))) + ...))):Bool
(false):Bool
(0.000 sec for parse, 19 rewrites(0.000 sec), 34 matches)
```

CafeOBJ system reports, in the third line, that nineteen rewrite steps are needed to reduce t into the normal form `false`. When we give the local strategy {`strat: (0)`} to the operation symbols `0`, `s_` and `_+_` in `NAT+`, which corresponds to $\mu(\texttt{0}) = \mu(\texttt{s}) = \mu(\texttt{+}) = \emptyset$, t is reduced to `false` by only two rewrite steps.

```
-- reduce in ZERO : (zero(((s (s (s (s (s 0))))) + ...))):Bool
(false):Bool
(0.000 sec for parse, 2 rewrites(0.000 sec), 4 matches)
```

Example 23. Consider the following specification of lists:

```
mod* LIST{
  pr(NAT+)
  [List]
  op _;_ : Nat List -> List {strat: (0)}
  op hd_ : List -> Nat
  op tl_ : List -> List
  op from : Nat -> List
  var N : Nat
  var L : List
  eq hd(N ; L) = N .
  eq tl(N ; L) = L .
  eq from(N) = N ; from(s N) .
}
```

The operation symbol `_;_` denotes the list constructor, `hd` and `tl` take a list and return the head element and the remaining list respectively. `from` makes an infinite list. The term `from(0)` denotes the infinite list $0 ; 1 ; 2 ; 3 ; 4 ; \cdots$. The operation symbol `_;_` is {`Nat`}-sort reducible and μ is correct on {`Nat`} when $\mu(cons) = \emptyset$ from Theorem 21. `LIST` is one of the typical examples to show the usefulness of CSR since for this replacement map μ, it is known that the TRS `LIST` is μ-terminating, that is, there is no infinite μ-rewrite sequence $t_1 \rightarrow_\mu t_2 \rightarrow_\mu \cdots$ (Lucas, 1998). Thus, it guarantees that we can compute an R-normal form of any given `Nat`-term in finite time. If $2 \in \mu(\texttt{;})$, reduction may fall into an infinite loop.

3.5. Ground sort reducible operation symbols

By combining the μ-ground reducibility and the S'-sort reducibility, we define the notion of μ-ground S'-sort reducibility of operation symbols.

Definition 24. Let μ be a replacement map, $f \in \Sigma$ and $S' \subseteq S$. f is μ-ground S'-sort reducible if any $t \in T_f \cap GT^\mu \cap T_{S'}$ is reducible.

Definition 25. Let $S' \subseteq S$ be a set of sorts. A replacement map μ is μ-ground correct on S' if $NF_\mu \cap GT^\mu \cap T_{S'} \subseteq NF_R$.

Lemma 26. *Let μ be a replacement map. If for each operation symbol $f \in \Sigma$ we have that f is μ-ground S'-sort reducible or $\mu(f)$ is trivial, then no μ-ground S'-sorted μ-normal form contains any μ-ground S'-sort reducible operation symbol.*

Proof. Similar to the proofs of Lemma 13 and 19. □

Theorem 27. *Let μ be a replacement map. If for each operation symbol $f \in \Sigma$ we have that f is μ-ground S'-sort reducible or $\mu(f)$ is trivial, then μ is μ-ground correct on S'.*

Proof. From Lemma 26. □

Example 28. Consider the following specification of a cell:

```
mod* CELL{
  pr(NAT+)
  [Cell]
  op empty : -> Cell
  op put : Nat Cell -> Cell
  op zero : Cell -> Bool
  var N : Nat
  var C : Cell
  eq zero(empty) = false .
  eq zero(put(0, C)) = true .
  eq zero(put(s N, C)) = false .
}
```

The element of `Cell` denotes a cell which stores a natural number. The constant `empty` is the initial empty cell. The operation symbol `put` overwrites the cell, i.e., `put(`n`,`c`)` denotes the result of putting the natural number n on the cell c. The operation symbol `zero` checks whether the stored number is zero or not. Although the operation symbol `put` is neither μ-ground reducible nor {`Bool`}-sort reducible for any μ, it is μ-ground {`Bool`}-sort reducible if μ(`put`) $= \{1\}$. The replacement map μ is μ-ground correct on {`Bool`} from Theorem 27.

4. Reducible operation symbols for behavioral specifications

In this section, we show the usefulness of reducibility of operation symbols by giving a sufficient condition for behavioral coherence in behavioral specifications.

4.1. Behavioral specification

Behavioral specifications are CafeOBJ specifications (modules) which contain a special sort, called a hidden sort, and special operation symbols, called behavioral operation symbols (Diaconescu and Futatsugi, 1998, 2000). A behavioral specification describes a behavior of a system. A hidden sort denotes the state space of a system to be described, and the system can be observed and modified through only behavioral operation symbols. We call a term of a hidden sort a state. Non-hidden sorts are called visible. The set of all hidden sorts and visible sorts are denoted by $\mathcal{H}$ and $\mathcal{V}$ respectively. A term $t \in T_{\mathcal{H}}$ is called a hidden term. A term $t \in T_{\mathcal{V}}$ is called a visible term. Any behavioral operation symbol should have exactly one hidden sort in its arity, i.e., if $f \in \Sigma_{\vec{s},s}$ is behavioral, then $\exists! i \in \mathcal{N}. s_i \in \mathcal{H}$. The converse is not always true. The set of all behavioral operation symbols is denoted by Σ^b ($\subseteq \Sigma$). Behavioral operation symbols are separated into observations and actions. Let $f \in \Sigma^b_{w,s}$ be a behavioral operation symbol. If its co-arity s is visible, then

f is called an observation (or an attribute). If its co-arity s is hidden, then f is called an action (or a method). An operation symbol is called hidden if it has a hidden sort in its arity. A tuple $(\mathcal{H}, \mathcal{V}, \Sigma, \Sigma^b)$ is called a CHA (coherent hidden algebra) signature (Diaconescu and Futatsugi, 2000), where $\mathcal{H} \cap \mathcal{V} = \emptyset$ and $\Sigma^b \subseteq \Sigma$. We may write Σ instead of $(\mathcal{H}, \mathcal{V}, \Sigma, \Sigma^b)$ if there is no confusion. A hidden but non-behavioral operation symbol is called a hidden constructor, and the set of all hidden constructors is denoted by $\mathcal{HC}$.

4.1.1. Behavioral equivalence

A central concept of the behavioral specification is behavioral equivalence, which is a weaker relation than the ordinary equality. In a denotational model of a behavioral specification, elements are called behaviorally equivalent if they are not distinguished by any behavioral operation symbol. Behavioral equivalence is defined by the notion of a behavioral context. A behavioral context is a context $C[z]_p$ in which all operation symbols above z are behavioral, i.e., $C(q) \in \Sigma^b$ for any $q < p$. The set of all behavioral contexts is denoted by $\mathcal{BC}$, and the set of all visible behavioral contexts is denoted by $\mathcal{BC}_\mathcal{V}$.

Definition 29. (Diaconescu and Futatsugi, 2000) Let Σ be a CHA signature, and M be a Σ-algebra. Elements $a, a' \in M_s$ are behaviorally equivalent, denoted by $a \sim a'$, if $M_C(a) = M_C(a')$ for any visible behavioral context $C[z] \in \mathcal{BC}_\mathcal{V}$ [8].

Note that visible elements are behaviorally equivalent if and only if they are equivalent in the ordinary sense, since z itself is a behavioral context. In CafeOBJ, a hidden sort H, a behavioral operation symbol $f \in \Sigma^b_{w,s}$ and a behavioral equation $t \sim t'$ are written like `*[` H `]*`, `bop` f `:` w `->` s and `beq` t `=` t', respectively.

Example 30. We give the behavioral specification `BCELL` by modifying `CELL` in Example 28 as follows: `[Cell]`, `op put` and `op zero` are changed into `*[Cell]*`, `bop put` and `bop zero` respectively. Then, `put` is an action and `zero` is an observation.

4.1.2. Behavioral coherence

We introduce another important property called behavioral coherence.

Definition 31. (Diaconescu and Futatsugi, 2000) An operation symbol $f \in \Sigma_{w,s} - \Sigma^b$ is called behaviorally coherent for $M \in [MOD]$ if it preserves the behavioral equivalence, i.e., $\vec{a} \sim_w \vec{b}$ implies $M_f(\vec{a}) \sim_s M_f(\vec{b})$ [9]

When f is behaviorally coherent for any $M \in [MOD]$, we say f is behaviorally coherent. It is trivial that each behavioral operation symbol is behaviorally coherent, and each non-hidden operation symbol, i.e., it has only visible sorts in its arity, is behaviorally coherent. Only hidden constructors can be a target to be proved behaviorally coherent. For a signature $(S, \leq, \Sigma)$ and a Σ-algebra M, an S-sorted relation $\equiv$ on M, i.e., $\equiv_s \subseteq M_s \times M_s$, is called a Σ-congruence if the relation $\equiv$ is an equivalence relation and is preserved by applying operation symbols $f \in \Sigma$, i.e., $\vec{a} \equiv_{\vec{s}} \vec{b} \Rightarrow M_f(\vec{a}) \equiv_s M_f(\vec{b})$ for any $f \in \Sigma_{\vec{s},s}$. From Definition 29, the behaviorally equivalence relation $\sim$ is Σ^b-congruence, however, is not Σ-congruence. The rewrite relation $\to_R$ is not sound for $\sim$ in general, i.e.,

[8] This equality means an equality between functions $M_{w_1 w_2} \to M_{s'}$ where $M_C : M_{w_1 s w_2} \to M_{s'}$. $M_C(e) : M_{w_1 w_2} \to M_{s'}$ is obtained by applying M_C to e at the position of z of the context $C[z]$.

[9] For S-sorted relation $\equiv$, $\vec{a} \equiv_{\vec{s}} \vec{b}$ is an abbreviation of $(a_1 \equiv_{s_1} b_1) \wedge (a_2 \equiv_{s_2} b_2) \wedge \cdots \wedge (a_n \equiv_{s_n} b_n)$.

$t \to_R t'$ does not imply $M_t \sim M_{t'}$. If all hidden constructors $f \in \mathcal{HC}$ are behaviorally coherent, then $\sim$ is a Σ-congruence.

Example 32. Consider adding the operation symbol `op merge : Cell Cell -> Cell` on the behavioral specification `BCELL`, which takes two cells and returns the merged cell, where we do not give any equation which defines what "merge" means. Since `merge` has more than one hidden sort `Cell` in its arity, it cannot be a behavioral operation symbol. Assume the constants `a` and `b` are behaviorally equivalent, i.e., `beq a = b`. Although `merge(a,a)` $=_E$ `merge(b,b)`, it does not guarantee that $\forall M \in [\mathrm{BCELL}].(M_{\texttt{merge(a,a)}} \sim M_{\texttt{merge(b,b)}})$.

4.1.3. Behavioral rewriting

We introduce a rewrite relation for behavioral specifications, denoted by $\hookrightarrow_R$, which is sound for $\sim$. When $C[l\theta] \to C[r\theta]$, we call C the rewrite context. We give the definition of rewrite contexts for behavioral rewrite relation $\hookrightarrow_R$, called behaviorally coherent contexts.

Definition 33. (Diaconescu and Futatsugi, 1998, 2000) For a Σ-algebra M, a behaviorally coherent context for M is defined as a context $C[z]_p$ such that all operation symbols above z are behaviorally coherent for M.

To define behavioral rewrite relation $\hookrightarrow_R$, we prepare Σ^{BC} as a set of operation symbols which are assumed to be proved behaviorally coherent. Note that all operation symbols except hidden constructors are included in Σ^{BC} since they are behaviorally coherent for any Σ-algebra. The set $\mathcal{BCC}$ of behaviorally coherent contexts for Σ^{BC} is defined as follows: $BCC[z]_p \in \mathcal{BCC}$ if $BCC(q) \in \Sigma^{BC}$ for each $q < p$. The behavioral rewrite relation $\hookrightarrow_R$ is defined as follows: $t \hookrightarrow_R t'$ if $\exists p, q \in Pos(t).\exists BC \in \mathcal{BC}.(t \to_p t' \wedge t|_q = BC[t|_p])$ [10]. The set of all $\hookrightarrow_R$-normal forms is denoted by BNF_R. The equivalence relation $=^b_R$ is defined as the reflexive, symmetric and transitive closure of $\hookrightarrow_R$.

Proposition 34. *(Diaconescu and Futatsugi, 2000) Let MOD be a behavioral specification and $M \in [MOD]$. Assume M_f is behaviorally coherent for each $f \in \Sigma^{BC}$. If $t \hookrightarrow_R t'$ (or $t =^b_R t'$) then $M_t \sim M_{t'}$.*

4.2. Reducibility of behavioral operation symbols

In a behavioral specification, elements are compared through visible behavioral contexts. Thus, in reasoning by TRS, terms to be reduced can be assumed to be visible, and the $\mathcal{V}$-sort reducibility is suitable for behavioral specifications. A term containing a fresh hidden constant h is often used for verification, in which h is considered as an arbitrary element. Thus, the μ-ground reducibility is suitable for behavioral specifications where $i \in \mu(f)$ implies $s_i \notin \mathcal{H}$ for each $f \in \Sigma_{\vec{s},s}$. We consider the $\mu_{\mathcal{H}}$-ground $\mathcal{V}$-sort reducibility as a candidate of behavioral reducibility, where $\mu_{\mathcal{H}}$ is defined as $\mu_{\mathcal{H}}(g) = \{i \in \mathcal{N} \mid s_i \notin \mathcal{H}\}$ for each $g \in \Sigma_{\vec{s},s}$. Consider the following example.

[10] The behavioral rewrite relation is implemented in CafeOBJ. Σ^{BC} is defined as the set of all behavioral operation symbols, non-hidden operation symbols and hidden constructors which declared with the attribute {`cohere`}. In CafeOBJ, each hidden constructor is declared without {`cohere`} first, and after proving it to be behaviorally coherent, it is declared again with {`cohere`}.

Example 35. We give a behavioral specification of an array whose indexes and values are both natural numbers. First we give a specification `NAT=` of the equality predicate on natural numbers.

```
mod! NAT={
  pr(NAT+)
  op _=_ : Nat Nat -> Bool
  vars M N : Nat
  eq (  0 = 0)   = true .
  eq (s M = 0)   = false .
  eq (  0 = s N) = false .
  eq (s M = s N) = (M = N) .
}
```

Next, we give a behavioral specification `ARRAY` of an array by importing `NAT=`.

```
mod* ARRAY{
  pr(NAT=)
  *[Array]*
  bop val : Nat Array -> Nat
  bop put : Nat Nat Array -> Array
  vars M N X : Nat
  var A : Array
  eq val(N, put(M, X, A)) = if (N = M) then X else val(N, A) fi .
}
```

where the operation symbol `if_then_else_fi` is defined in `BOOL` as follows: for each sort $s \in S$, it belongs to $\Sigma_{\texttt{Bool}\,s\,s,s}$ and the equations `eq if true then X else Y fi = X` and `eq if false then X else Y fi = Y` are declared where `X` and `Y` are variables of s. `Array` is a hidden sort. `val` and `put` are an observation and an action respectively. For an array `A`, `val`$(n,$ `A`$)$ denotes the value assigned to n of `A`. `put`$(n, x,$ `A`$)$ denotes the result of updating `A` by assigning x to n. Note that states are not defined directly but are defined through the observation in the equation. The meaning of the action `put` is defined by the equation which describes the values of the post-state `put(M,X,A)` is defined by the values of the pre-state `A`.

In this example, if the arguments of the hidden sort of `put` is restricted, i.e., $\mu(\texttt{put}) = \{1, 2\}$, then `put` is $\mu_{\mathcal{H}}(f)$-ground $\mathcal{V}$-sort reducible. Thus, from Lemma 26, no $\mu_{\mathcal{H}}$-ground visible $\mu_{\mathcal{H}}$-normal form contains `put`. The $\mu_{\mathcal{H}}(f)$-ground $\mathcal{V}$-sort reducibility seems to work well as behavioral reducibility of the example of `ARRAY`, however, it does not work well for the following example.

Example 36. Consider the following behavioral specification of an array with the addition function:

```
mod* ARRAY-ADD{
  pr(ARRAY)
  op add : Nat Nat Array -> Array
  vars M N X : Nat
  var A : Array
  eq val(N, add(M,   0, A)) = val(N, A) .
  eq val(N, add(M, s X, A)) = if (N = M) then s val(N, add(M, X, A))
                                             else val(N, A) fi .
}
```

The term $\mathtt{add}(n, x, \mathtt{A})$ denotes updating `A` by adding x to the value assigned to n. Note that `add` is a hidden constructor since it is not declared with `bop`.

The hidden constructor `add` is not $\mu_{\mathcal{H}}$-ground $\mathcal{V}$-sort reducible since, for example, `val(0,add(0,val(0,`x`),`y`))` is visible and μ-ground, but is not reducible. The reason why `val(0,add(0,val(0,`x`),`y`))` is not reducible is because `add` has `val(0,`x`)` in its argument. The operation symbol `add` seems to be completely defined for all natural numbers at the second argument, since it defined for all pattern $\mathtt{s}^n\mathtt{(0)}$, and is expected to be removed in normal forms. Since `NAT+` is imported by `ARRAY-ADD` with the protect mode[11], for any element $e \in M_{\mathtt{Nat}}$, there exists $t = \mathtt{s}^n\mathtt{(0)}$ such that $M_t = e$ for any $M \in [\mathtt{ARRAY\text{-}ADD}]$. We call such a sort like `Nat` in `ARRAY-ADD` a tightly protected sort.

Definition 37. In a module MOD, we call a sort $s \in S_{MOD}$ tightly protected if it is declared in a tight and basic module MOD' imported by MOD with the protect mode. The set of all tightly protected sorts w.r.t. MOD is denoted by TP_{MOD} (or TP).

The sort `Nat` is tightly protected in `ARRAY-ADD` since it is declared in `NAT+` which is a tight and basic module and is imported with the protect mode. If $s \in TP_{MOD}$ and s is declared in MOD', the following property holds: for any $M \in [MOD]$ and any element $e \in M_s$, there exists a ground term $t \in (T_{\Sigma_{MOD'}})_s$ such that $M_t = e$. Since a behavioral specification describes the behavior of a system and denotes all models (implementations) satisfying the behavior, the hidden sort should not be tightly protected, i.e., $TP \subseteq \mathcal{V}$. We give a suitable restriction of terms for behavioral specifications.

Definition 38. Let MOD be a behavioral specification. A term $t \in T$ is a TP-ground term if for any subterm $f(\vec{t}) \in Sub(t)$ where $f \in \Sigma^b_{\vec{s},s} \cup \mathcal{HC}_{\vec{s},s}$ is a hidden operation symbol, for any $s_i \in TP$, the term t_i is constructed from the operation symbols declared in MOD' in which s_i is declared, i.e., $t_i \in (T_{\Sigma_{MOD'}})_{s_i}$. The set of all TP-ground terms is denoted by GT^{TP}.

Let $TP = TP_{\mathtt{ARRAY\text{-}ADD}}$. The term `val(0,add(0,val(0,`$x$`),`$y$`))` is not TP-ground since `val(0,`x`)` is not in $T_{\Sigma_{\mathtt{NAT+}}}$. The term `val(0,add(s 0, 0 + s 0,`$y$`))` is TP-ground since `0`, `s 0`, `0 + s 0` are in $T_{\Sigma_{\mathtt{NAT+}}}$. We define the notion of reducibility of behavioral operation symbols.

Definition 39. Let MOD be a behavioral specification. A behavioral operation symbol $f \in \Sigma_{\vec{s},s}$ is behaviorally reducible if any $t \in T_f \cap GT^{TP} \cap T_{\mathcal{V}}$ is reducible.

The behavioral CSR relation $\hookrightarrow_\mu$ is defined as follows: $t \hookrightarrow_\mu t' \overset{\text{def}}{\iff} t \hookrightarrow_R t' \wedge t \rightarrow_\mu t'$. The set of all $\hookrightarrow_\mu$-normal forms is denoted by BNF_μ. The correctness of replacement maps for behavioral specifications is defined as follows.

[11] Note that the protect import is transitive, i.e., since `NAT=` has the declaration of `pr(NAT+)`, `ARRAY` has `pr(NAT=)` and `ARRAY-ADD` has `pr(ARRAY)`, then `NAT+` is imported by `ARRAY-ADD` with the protect mode.

Definition 40. Let MOD be a behavioral specification. A replacement map μ is behaviorally correct if $BNF_\mu \cap GT^{TP} \cap T_\mathcal{V} \subseteq BNF_R$.

For a behavioral specification MOD and Σ^{BC}, we define the replacement map μ^{BC} as follows: $\mu^{BC}(g)$ is trivial for each $g \in \Sigma^{BC}$ and $\mu^{BC}(f) = \{i \in \mathcal{N} \mid s_i \in TP_{MOD}\}$ for each $f \in \Sigma_{\vec{s},s} - \Sigma^{BC}$. We give a sufficient condition for behavioral correctness.

Lemma 41. *Let MOD be a behavioral specification. Let $\mu = \mu^{BC}$. If each $f \in \Sigma - \Sigma^{BC}$ is behaviorally reducible, no TP-ground visible $\hookrightarrow_\mu$-normal form contains behaviorally reducible operation symbols.*

Proof. Similar to the proofs of Lemma 13 and 19. Notice that in the proofs of Lemma 13 and 19, we consider the occurrence at the smallest position of reducible operation symbols f. Since $\mu(f)$ is trivial for all operation symbols $f \in \Sigma^{BC}$, the considered operation symbol $f \in \Sigma - \Sigma^{BC}$ is under a behavioral coherent context. Thus, the proofs can be applied to the case of $\hookrightarrow_R$ and $\hookrightarrow_\mu$. □

Theorem 42. *Let MOD be a behavioral specification. If each $f \in \Sigma - \Sigma^{BC}$ is behaviorally reducible, then μ^{BC} is behaviorally correct.*

Proof. From Lemma 41. □

Example 43. In `ARRAY-ADD`, the hidden constructor `add` is behaviorally reducible. From Lemma 41 and Theorem 42, no TP-ground visible $\hookrightarrow_\mu$-normal form contains `add` and is a $\hookrightarrow_R$-normal form.

4.3. A sufficient condition for behavioral coherence

In order to give a sufficient condition for behavioral coherence by the behavioral reducibility, we introduce the notion of weakly normalizing and TP condition. The relation $\rightarrow \subseteq T \times T$ is weakly normalizing if for any $t \in T$, there exists a $\rightarrow$-normal form u such that $t \rightarrow^* u$ (Ohlebusch, 2002; Terese, 2003). In general, a mathematical function should associate an element in its range to each element in its domain. Roughly speaking, the reducibility guarantees the part of "each element in its domain" and the weak normalization guarantees the existence of "an element in its range". We give a restriction of equations in specifications for preserving TP-ground terms. The TP condition guarantees that any term reduced from a TP-ground term is also TP-ground.

Definition 44. A behavioral specification MOD satisfies the TP condition if for each equation $l = r$ in MOD and all imported modules, each occurrence $f(\vec{r}) \in Sub(r)$ of a hidden operation symbol $f \in \Sigma^b_{\vec{s},s} \cup \mathcal{HC}_{\vec{s},s}$ in r, and each tightly protected sort $s_i \in TP_{MOD}$, the term r_i is constructed from operation symbols in Σ_{MOD_i} and variables in $V(l_i)$ for some $f(\vec{l}) \in Sub(l)$, where MOD_i is the module in which s_i is declared.

Lemma 45. *Let MOD be a behavioral specification satisfying the TP condition. If $t \in GT^{TP}$ and $t \rightarrow^*_{MOD} t'$, then $t' \in GT^{TP}$.*

Proof. Assume that $t \in GT^{TP}$ and $t = C[l\theta] \rightarrow_{MOD} C[r\theta] = t'$. It suffices to show $r\theta$ is TP-ground since if u and u' are TP-ground then $u[u']_p$ is also TP-ground from Definition 38. Let $r\theta|_p = f(\vec{u}) \in Sub(r\theta)$ where $f \in \Sigma^b_{\vec{s},s} \cup \mathcal{HC}_{\vec{s},s}$. Let MOD_i be the module in which s_i is declared. Let $s_i \in TP$ be an arbitrary tightly protected sort in $\{\vec{s}\}$. Then, $r\theta$ is TP-ground if u_i is constructed from Σ_{MOD_i}. **(a)** Consider the case of $r(p) \notin \Sigma$. There exists $x \in V(r)$ such that $f(\vec{u}) \in Sub(\theta(x))$. Since each TRS satisfies $V(r) \subseteq V(l)$, $f(\vec{u})$ is a subterm of $l\theta$ and is TP-ground. From the definition of TP-ground terms, u_i is constructed from Σ_{MOD_i}. **(b)** Consider the case of $r(p) \in \Sigma$. There exists $f(\vec{r}) \in Sub(r)$ such that $f(\vec{u}) = f(\vec{r})\theta$ and $u_i = r_i\theta$ for each i. From Definition 44, r_i is constructed from Σ_{MOD_i} and $V(l_i)$ for some $f(\vec{l}) \in Sub(l)$. Since $t \in GT^{TP}$ and $l\theta \in GT^{TP}$, $l_i\theta$ is constructed from Σ_{MOD_i} and so is $\theta(x)$ for each $x \in V(l_i)$. Since $u_i = r_i\theta$ and $V(r_i) \subseteq V(l_i)$, u_i is constructed from Σ_{MOD_i}. □

To prove f to be behaviorally coherent, we assume $\vec{a} \sim \vec{b}$ and prove $M_f(\vec{a}) \sim M_f(\vec{b})$. To prove $M_f(\vec{a}) \sim M_f(\vec{b})$, we prove $M_C[M_f(\vec{a})] = M_C[M_f(\vec{b})]$ for all visible behavioral contexts $C \in \mathcal{BC}_\mathcal{V}$. We show that only TP-ground contexts are enough to show behavioral equivalence.

Lemma 46. *Let MOD be a behavioral specification, $M \in [MOD]$ and $a, b \in M_s$. If $M_C(a) = M_C(b)$ for each visible TP-ground behavioral context $C \in \mathcal{BC}_\mathcal{V} \cap GT^{TP}$, then a and b are behaviorally equivalent, i.e., $a \sim b$.*

Proof. It suffices to show that $M_{C'}(a) = M_{C'}(b)$ for each visible behavioral context $C'[z]_p \in \mathcal{BC}_\mathcal{V}$. Let $s' \in \mathcal{V}$ be the sort of context $C' \in T_{s'}$. Note that $M_{C'}$ is a function whose domain is the product of the corresponding carrier sets of variables $V(C')$. We define the set TPA of all tightly protected arguments in C' as $TPA = \{t_i \in T \mid f(\vec{t}) \in Sub(C'), t_i \in T_{s_i}, s_i \in TP\}$. The set of all variables in TPA is denoted by $V_{TPA} = \bigcup_{t \in TPA} V(t)$. Then, the set $V(C')$ of variables in C' can be written as $\{z, \vec{x}, \vec{y}\}$ where z is the marked variable for the context, $\{\vec{x}\} = V_{TPA}$, and $\{\vec{y}\}$ is the set of other variables. We assume $M_{C'}$ is a function of $M_s \times \Pi_{i=1}^m X \times \Pi_{i=1}^n Y \to M_{s'}$ where $z \in V_s$, $x_i \in V_{s_i}$, $y_i \in V_{s'_i}$, $X_i = M_{s_i}$ and $Y_i = M_{s'_i}$. Let $\vec{e} \in \vec{X}$ be arbitrary elements in $\vec{X}$ (fixed). Then, $M_{C'}(\vec{e})$ is a function of $M_s \times \Pi_{i=1}^n Y \to M_s$. We construct a visible TP-ground behavioral context $C \in \mathcal{BC}_\mathcal{V} \cap GT^{TP}$ such that $M_C = M_{C'}(\vec{e})$ as follows: For each $t \in TPA$, it holds that $V(t) \subseteq \{\vec{x}\}$. Thus, M_t can be regarded as the function of $\Pi_{i=1}^m X \to M_{s''}$, and $M_t(\vec{e})$ is an element of $M_{s''}$ where $t \in T_{s''}$ and $s'' \in TP$. Since $s'' \in TP$, there exists a ground term $t' \in T(\Sigma_{MOD'})$ such that $M_t(\vec{e}) = M_{t'}$ where MOD' is the basic tight module where s'' is declared. Let C be the context obtained by replacing all occurrences of $t \in TPA$ in C' with $t' \in T(\Sigma_{MOD'})$ which is obtained as above. Then C is a visible TP-ground behavioral context satisfying $M_C = M_{C'}(\vec{e})$. From the assumption, $M_C(a) = M_C(b)$ holds, i.e., $M_C(a)(\vec{e'}) = M_C(b)(\vec{e'})$ for all $\vec{e'} \in \vec{Y}$. Therefore, we conclude that $M_{C'}(a) = M_{C'}(b)$ since $M_{C'}(a)(\vec{e}, \vec{e'}) = M'_c(b)(\vec{e}, \vec{e'})$ for all $\vec{e} \in \vec{X}$ and $\vec{e'} \in \vec{Y}$. □

We give a sufficient condition for behavioral coherence. In a standard methodology for proving behavioral coherence in CafeOBJ, the proof is done as follows: we declare `beq a = b` for fresh hidden constants `a` and `b`, and prove `f(a,...)` $\sim$ `f(b,...)` by checking if both C[`f(a,...)`] and C[`f(b,...)`] are reduced to a same term or not for each $C \in \mathcal{BC}$ (Diaconescu and Futatsugi, 1998, 2000). The proof of the following theorem is according to the methodology.

Theorem 47. *Let MOD be a behavioral specification satisfying the TP condition. Let μ be the replacement map μ^{BC}. If $\hookrightarrow_\mu$ is weakly normalizing and all hidden constructors are behaviorally reducible, then all hidden constructors are behaviorally coherent.*

Proof. Let $\Sigma^{BC} = \Sigma - \mathcal{HC}$. We prove $f \in \mathcal{HC}$ behaviorally coherent. Without loss of generality, we assume the hidden constructor f is in $\Sigma_{w,s}$ such that $w = \vec{s}\vec{s'}\vec{s''}$, $\vec{s} \in \mathcal{H}$, $\vec{s'} \in TP$ and $\vec{s''}$ are neither hidden nor tightly protected. Our goal is to prove $M_f(\vec{a}, \vec{e}, \vec{e'}) \sim M_f(\vec{b}, \vec{e}, \vec{e'})$ for all elements $\vec{a}$, $\vec{b}$, $\vec{e}$ and $\vec{e'}$ satisfying $\vec{a} \sim_{\vec{s}} \vec{b}$ for all $M \in [MOD]$. From Theorem of Constants (Goguen and Malcolm, 1996), the fresh constants $\vec{\mathtt{e'}}$ can be used as arbitrary elements $\vec{e'}$, and the fresh hidden constants $\vec{\mathtt{a}}$ and $\vec{\mathtt{b}}$ with the behavioral equations `beq` $\mathtt{a}_i$ `=` $\mathtt{b}_i$ can be used as arbitrary elements $\vec{a}$ and $\vec{b}$ satisfying $\vec{a} \sim_{\vec{s}} \vec{b}$. An arbitrary element $e_i \in M_{s'_i}$ of the tightly protected sort s'_i, there exists $t_i \in T(\Sigma_{MOD_i})_{s'_i}$ such that $M_{t_i} = e_i$ where s'_i is declared in MOD_i. Let $t_i \in T(\Sigma_{MOD_i})_{s'_i}$ be arbitrary ground terms. From Proposition 34 and Lemma 46, it suffices to show that $C[t_1] =^b_{MOD} C[t_2]$ for all visible TP-ground behavioral context $C \in \mathcal{BC}_\mathcal{V} \cap GT^{TP}$ where $t_1 = \mathtt{f}(\vec{\mathtt{a}}, \vec{t}, \vec{\mathtt{e'}})$ and $t_2 = \mathtt{f}(\vec{\mathtt{b}}, \vec{t}, \vec{\mathtt{e'}})$. Let $t = \mathtt{f}(\vec{x}, \vec{t}, \vec{\mathtt{e'}})$ where $\vec{x}$ are fresh distinct variables. Note that t, t_1 and t_2 are TP-ground. Let θ_1 and θ_2 such that $\theta_1(x_i) = \mathtt{a}_i$ and $\theta_2(x_i) = \mathtt{b}_i$ for each i. Then, $t_1 = t\theta_1$ and $t_2 = t\theta_2$. Since C and t are TP-ground, $C[t]$ is also TP-ground. From the weak normalization of $\hookrightarrow_{MOD}$, there exists $u \in BNF_\mu$ such that $C[t] \hookrightarrow^*_{MOD} u$. From Lemma 45, u is also TP-ground. Note that $\hookrightarrow_R \subseteq \rightarrow_R$. u is visible since C is visible. Therefore, from Lemma 41, u contains no behavioral reducible operation symbol. From the assumption, u contains no hidden constructor. Consider the occurrences of $\vec{x}$ in u. Let $\vec{p} \in Pos(u)$ such that $u(p_i) = x_i$, that is, $u = u[\vec{x}]_{\vec{p}}$. Since u contains no hidden constructor, $\vec{x}$ are under behaviorally coherent contexts. Thus, $u\theta_1 = u[\vec{\mathtt{a}}]_{\vec{p}} \hookrightarrow^*_{MOD} u[\vec{\mathtt{b}}]_{\vec{p}} = u\theta_2$. Therefore, we conclude that $C[t_1] = C[t\theta_1] =^b_{MOD} u\theta_1 =^b_{MOD} u\theta_2 =^b_{MOD} C[t\theta_2] = C[t_2]$. □

Example 48. We give a module for the hidden constructor `merge` in Example 32.

```
mod* BCELL-M{
  pr(BCELL)
  *[Cell]*
  op merge : Cell Cell -> Cell
  vars C C' : Cell
  eq zero(merge(C, C')) = zero(C) and zero(C') .
}
```

Consider a term t which is visible TP-ground and contains `merge`. Take the smallest position p such that $t(p) = \mathtt{merge}$. Then, $t = BCC[\mathtt{merge(t,t')}]$ for some $BCC \in \mathcal{BCC}$. Behaviorally coherent contexts BCC for `BCELL-M` can be written as $C[\mathtt{zero}(\mathtt{put}(t_1, \mathtt{put}(t_2, \ldots \mathtt{put}(t_n, z)\cdots)))]$ for ground terms $t_i \in T(\Sigma_{\mathtt{NAT+}})$ and a visible context C. Consider the case of $k = 0$, that is, there is no `put` between `zero` and z. Since $\mathtt{zero(merge(t,t'))}$ is a redex, $BCC[\mathtt{merge(t,t')}] = C[\mathtt{zero(merge(t,t'))}]$ is reducible. Consider the case of $k > 0$, that is, there are at least one `put` between `zero` and z. Then, $\mathtt{zero}(\mathtt{put}(t_1, \cdots))$ is reducible since (1) if t_1 contains `_+_` then t_1 is reducible, otherwise $t_1 = \mathtt{s}^n(\mathtt{0})$ for some $n \in \mathcal{N}$ and $\mathtt{zero}(\mathtt{put}(\mathtt{s}^n(\mathtt{0}), \cdots))$ is a redex. Thus, `merge` is behaviorally reducible. For `BCELL-M`, we can prove $\rightarrow_R$ is terminating by the recursive path order (Terese, 2003) with the precedence `zero` $>$ `+` $>$ `s` $>$ `and` $>$ `true` $>$ `false`, and thus $\hookrightarrow_\mu$ is also terminating[12]. Termination implies weak normalization. Therefore, from Theorem 47, `merge` is behaviorally coherent.

[12] In the built-in module `BOOL`, the operation symbol `and` is declared with the following equations: `eq false and A = false`, `eq true and A = A` and `eq A and A = A`.

Example 49. The hidden constructor `add` in `ARRAY-ADD` in Example 36 can be proved behaviorally coherent. Consider the hidden constructor `op shift : Array -> Array` which shifts all elements to the next cell. One may describe a module which contains only the following equation for this purpose: `eq val(s N, shift(A)) = val(N, A)`, which denotes that the $n+1$-th element of the shifted array is the n-th element of the original array. Then, `shift` is not behaviorally coherent since `slide(0, A)` is indeterminate. If we add `eq shift(0, A) = 0`, then `shift` is behaviorally reducible. We can prove $\rightarrow_R$ is terminating by the recursive path order with the precedence `val` $>$ `=` $>$ `s` $>$ `if_then_else_fi` $>$ `true` $>$ `false`, and thus $\hookrightarrow_\mu$ is also terminating. Since $\hookrightarrow_\mu$ is terminating, `shift` is behaviorally coherent.

Like the above example of `shift`, a lack of equations defining a hidden constructor may cause a behavioral specification with non-behaviorally coherent hidden constructors. The behavioral reducibility may be useful for detecting such a mistake.

5. Related studies

5.1. Canonical replacement map in CSR

In (Lucas, 2002, 1998), the notion of a canonical replacement map has been proposed and some theorems on μ-normal forms have been proved. The canonical replacement map, denoted by μ_R^{can}, is defined as follows: $\forall f \in \Sigma, i \in \{1..ar(f)\}$,

$$i \in \mu_R^{can}(f) \Leftrightarrow \exists l \rightarrow r \in R, p \in Pos_\Sigma(l), (l(p) = f \wedge p.i \in Pos_\Sigma(l))$$

where $Pos_\Sigma(t) = \{p \in Pos(t) \mid t(p) \in \Sigma\}$. For an operation symbol f and an argument i, if the i-th argument of f is a variable for any occurrence of $f(\vec{l}) \in Sub(l)$ in the left-hand side of *any* rewrite rule $l \rightarrow r \in R$, the argument is restricted in the canonical replacement map. On the other hand, the reducibility of an operation symbol f is defined in terms of the reducibility of the terms containing the operation symbol f; that is, the reducibility of an operation symbol depends on whether *there exists* a rewrite rule that can be applied to the terms. We present an example that shows the difference between the two notions. Consider `NAT+`. The canonical replacement map $\mu_{\texttt{NAT+}}^{can}$ for `NAT+` is represented as $\mu_{\texttt{NAT+}}^{can}(\texttt{+}) = \{1\}$. It coincides with the replacement map μ where `_+_` is μ-ground reducible. Consider the following `NAT++`:

```
mod! NAT++{
  pr(NAT+)
  vars M N : Nat
  eq M + 0 = M .
  eq M + s N = s(M + N) .
}
```

It should be noted that in `NAT+`, `_+_` is defined inductively on the first argument, whereas in `NAT++`, it is defined inductively on the second argument. Since `NAT++` imports `NAT+`, the module `NAT++` includes four equations. Then, the canonical replacement map $\mu_{\texttt{NAT++}}^{can}$ for `NAT++` is represented as the trivial replacement map, i.e., $\mu_{\texttt{NAT++}}^{can}(\texttt{+}) = \{1, 2\}$, since there exist `M + 0 = M` and `0 + N = N` such that the first and second arguments of `_+_` are not variable, respectively. On the other hand, `_+_` is μ-ground reducible for both $\mu(\texttt{+}) = \{1\}$ and $\mu(\texttt{+}) = \{2\}$.

The following property with respect to the canonical replacement map and μ-normal forms has been proved.

Proposition 50. *(Lucas, 1998, Theorem 8) Let R be a left-linear TRS and μ a replacement map such that $\mu^{can}_R \subseteq \mu$. Every μ-normal form is a head-normal form.*

Here, $\mu \subseteq \mu'$ is defined as $\mu(f) \subseteq \mu'(f)$ for all $f \in \Sigma$. A term t is a head-normal form (or a root-stable term) if there exists no redex u such that $t \to^*_R u$. Every R-normal form is a head-normal form. As is the case with our results, Proposition 50 can be applied to all terms, though our theorems restrict the term to S'-sorted μ-ground terms (or visible TP-ground terms). On the other hand, Proposition 50 assumes the left linearity of TRSs, whereas our theorems hold true for all TRSs (or TRSs satisfying the TP condition in the case of behavioral specifications). Moreover, Proposition 50 guarantees that every μ-normal form is *a head-normal form*, whereas our theorems guarantee that every S'-sorted μ-ground μ-normal form (or visible TP-ground $\hookrightarrow_\mu$-normal form) is *an R-normal form.*

5.2. Ground reducibility and sufficient completeness

Our μ-ground reducibility can be regarded as a generalization of ground reducibility. If all non-constructor operation symbols $\Sigma - C$ are μ-ground reducible, $\mu(f)$ is trivial for each $f \in C$, and $\to_\mu$ is weakly normalizing, the TRS can be proved to be sufficiently complete by using Theorem 14. In behavioral specifications (or specifications with loose modules), the original definition of the sufficient completeness is too strong, since for a hidden sort s (or a sort declared in a loose module), there may not exist a corresponding ground term $t \in T_\Sigma$ for an element $e \in M_s$ such that $M_t = e$ for $M \in [MOD]$. TP-ground terms are considered suitable for behavioral specifications rather than ground terms.

In (Hendrix and Meseguer, 2007), the correctness property of CSR for ground terms, called μ-canonical completeness, has been proposed. For a replacement map and a left-linear TRS, an algorithm for constructing a tree automaton that can be used to check whether the replacement map is correct for ground terms has been proposed and implemented in the algebraic specification language Maude. Our correctness property for μ-ground and/or S'-sorted terms is a generalization of the correctness property for ground terms described above. As shown in Example 23, `LIST` (or a similar example `INF-LIST` mentioned in (Hendrix and Meseguer, 2007)) is a typical example that proves the usefulness of CSR. Although any replacement map μ is not correct for ground terms of `LIST`, μ is correct for `Nat`-sorted terms, as previously proved.

Let $\mathcal{D} = \{f \in \Sigma \mid f(\vec{l}) \to r \in R\}$ and $\mathcal{C} = \Sigma - \mathcal{D}$. For a left-linear TRS, we show that it is decidable whether all $f \in \mathcal{D}$ are μ-ground reducible.

Proposition 51. *Let (Σ, R) be a left-linear TRS. Then, it is decidable whether all $f \in \mathcal{D}$ are μ-ground reducible.*

Proof. We first define the following set FT of terms:

$$FT = \left\{f(\vec{t}) \in T(\Sigma, V) \mid f \in \mathcal{D},\ \forall i \in \mu(f).t_i \in GT^\mu \cap T(\mathcal{C}, V)\right\}$$

Let $|R|$ be the maximal height of the left-hand sides of all rewrite rules in a given TRS R, i.e., $|R| = max\{|l| \in \mathcal{N} \mid l \to r \in R\}$ and $|t| = max\{|p| \in \mathcal{N} \mid p \in Pos(t)\}$ where $|p|$ is the length of the position p. We define a subset FT' of FT where the height of each term is less than $|R|+2$ as follows: $FT' = cut^{|R|+1}(FT)$ where cut^i is defined as $cut^0(t) = \diamond$ and $cut^{i+1}(f(\vec{t})) = f(cut^i(t_1), \ldots cut^i(t_n))$, where $\diamond$ is a special fresh constant. Then, FT' is finite and computable. The following statements are equivalent:

(1) Each $f \in \mathcal{D}$ is μ-ground reducible.

(2) Each $f(\vec{t}) \in FT'$ is a redex.

($1 \Rightarrow 2$) Let $f(\vec{t}) \in FT'$. Since $\mathcal{C}$ does not contain any root symbol of left-hand sides of rewrite rules, for each $i \in \mu(f)$ the term $t_i \in T(\mathcal{C} \cup \{\diamond\}, V)$ is an R-normal form, and thus is a μ-normal form. From Theorem 11, $f(\vec{t})$ is a redex.

($2 \Rightarrow 1$) Let $t \in GT^{\mu}$. It suffices to show that t is reducible if t contains some $f \in \mathcal{D}$. We take $p \in Pos(t)$ such that $t(p) \in \mathcal{D}$ and $t(q) \in \mathcal{C}$ for each $q < p$. Let $f(\vec{t}) = t|_p$. Then, $t_i \in GT^{\mu} \cap T(\mathcal{C}, V)$ for each $i \in \{1..ar(f)\}$ and $f(\vec{t}) \in FT$ from the definition of FT. From the assumption (2), $cut^{|R|+1}(f(\vec{t}))$ is a redex. There exists $l \to r \in R$ such that $cut^{|R|+1}(f(\vec{t}))$ is an instance of l. Since the length of the position of each $\diamond$ is longer than $|R|$ and l is linear, $f(\vec{t})$ is also an instance of l. Thus, $f(\vec{t})$ is a redex, and t is reducible. $\square$

For a non-left-linear case, techniques described in the literature (Comon and Jacquemard, 2003) may be useful; here, some decidability results for ground reducibility have been proposed. In order to extend Proposition 51 for S'-sort reducibility, we may require another set of operation symbols $\mathcal{D}$. In the future, we intend to clarify the decidability of μ-ground S'-sort reducibility and develop a behavioral coherent checker based on reducibility.

5.3. Checking behavioral coherence

There are two approaches to prove behavioral coherence: an interactive proof with behavioral rewriting and an automatic proof with observer complete definitions (OCDs).

In an interactive proof, we show that a given operation symbol whose behavioral coherence is to be proved directly preserves the behavioral equivalence according to Definition 31 by using the CafeOBJ system in which behavioral rewriting is implemented. In order to prove $a \sim b \Rightarrow f(a) \sim f(b)$, we first declare fresh operation symbols `ops a b : -> Elt` as arbitrary elements and a behavioral equation `beq a = b`. Then, we check whether `f(a)` is equivalent to `f(b)` by reducing both the terms by using the CafeOBJ reduction command. If they are reduced to a same term, f is behaviorally coherent. If they are reduced to different terms, we try to use different techniques to prove their joinability, such as case splitting, induction, and finding a suitable lemma. More details and examples can be found in the literatures (Diaconescu and Futatsugi, 1998, 2000).

As an automatic proof method for behavioral coherence, the notion of OCDs has been proposed (Bidoit and Hennicker, 1999). An OCD for an operation symbol f is a set $\{C_1[f(\vec{x})] = r_1, C_2[f(\vec{x})] = r_2, \ldots, C_k[f(\vec{x})] = r_k\}$ of equations satisfying the conditions that **(a)** $\{C_1, \ldots, C_k\}$ are behaviorally complete contexts[13], **(b)** all $\vec{x}$ are distinct variables that do not occur in each C_i, **(c)** all operation symbols containing hidden sorts in r_j are either f or observations, and **(d)** there exists a monotonic well-founded order

[13] CC are behaviorally complete contexts if $\forall BC \in \mathcal{BC}.\ \exists C_i \in CC.\ (BC[z] = C[C_i[z]])$.

$>$ on contexts such that $C_i > C$ for any $C[f(\overrightarrow{t})] \in Sub(r_i)$[14]. It has been shown that if a behavioral specification has an OCD for f, f is behaviorally coherent (Bidoit and Hennicker, 1999).

CafeOBJ implements a behavioral coherence checker based on the OCDs. If there exists an OCD for f in the input behavioral specification, f can be declared as behaviorally coherent by using the CafeOBJ system as follows:

```
CafeOBJ> mod* BCELL-M{
  pr(BCELL)
  *[Cell]*
  op merge : Cell Cell -> Cell
  vars C C' : Cell
  eq zero(merge(C, C')) = zero(C) and-also zero(C') .
}
-- defining module* BCELL-M...._.*
** system found the operator
  merge : Cell Cell -> Cell
  can be declared as coherent. done.
```

The (singleton set of) equation `zero(merge(C,C')) = zero(C) and-also zero(C')` is an OCD for `merge`. However, the CafeOBJ system cannot prove that `add` and `shift` are behaviorally coherent in `ARRAY-ADD`. Compared to our results, the conditions (a) and (b) correspond to behavioral reducibility and the conditions (c) and (d) correspond to the weak normalization of $\hookrightarrow_\mu$. A remarkable difference from our results is that in OCDs, all arguments should be variables and the form of right-hand sides are syntactically restricted. The equations `eq val(N, add(M, 0, A)) = val(N, A)` and `eq val(s N, shift(A)) = val(N, A)` cannot be a part of an OCD since the arguments of `add` and `shift` include non variable terms. Next, consider the hidden constructor `op addOneAtZero : Array -> Array` and the equation `eq addOneAtZero(A) = add(0, s 0, A)`. The hidden constructor `addOneAtZero` is regarded as being defined by using other hidden constructor `add`. The condition (c) does not allow such equations. The weak normalization of $\hookrightarrow_\mu$ in Theorem 47 essentially includes the conditions (c) and (d)[15]. Thus, the OCD can be considered as a special case of our sufficient condition for behavioral coherence. By checking the behavioral reducibility on the basis of our theorem with μ-termination checkers, for example, by using AProVE (Alarcón et al., 2008) and MU-TERM (Lucas, 2004)[16], we can obtain an automatic behavioral coherence checker that can be applied to `add` and `shift` in `ARRAY-ADD`.

[14] $>$ is monotonic if $C_1 > C_2$ implies $C[C_1] > C[C_2]$. $>$ is well-founded if there exists no infinite decreasing sequence $C_0 > C_1 > C_2 > \cdots$.

[15] Strictly speaking, an OCD may accept non weakly normalizing TRSs since the condition (d) of the OCDs depends on only the behavioral contexts. Our theorems can easily be improved for including the condition (d).

[16] AProVE: `http://aprove.informatik.rwth-aachen.de/`
MU-TERM: `http://www.dsic.upv.es/~slucas/csr/termination/muterm/`

5.4. *Observational transition system*

Let us assume there exists a set Υ for a universal state space and sets $\vec{D}$ of data. An observational transition system (OTS) consists of the set $\mathcal{O} = \{o_i : \Upsilon \to D_i\}$ of observations, set $\mathcal{I} \subseteq \Upsilon$ of initial states, and set $\mathcal{T} = \{\tau_i : \Upsilon \to \Upsilon\}$ of transitions, where each $\tau \in \mathcal{T}$ preserves the observational equivalence; that is, for all $u, u' \in \Upsilon$, $u =_{\mathcal{O}} u'$ implies that $\tau(u) =_{\mathcal{O}} \tau(u')$. The observational equivalence with respect to $\mathcal{O}$, denoted by $=_{\mathcal{O}}$, is defined as follows: $u =_{\mathcal{O}} u'$ if $\forall o \in \mathcal{O}.(o(u) = o(u'))$. The OTS is known to be useful for describing transition systems with infinitely many states. There are several case studies, for example, mutual exclusion algorithms (Ogata and Futatsugi, 2001, 2002), security protocols (Ogata and Futatsugi, 2003), etc, in which important properties for those systems are verified formally. The OTS specifications can be described as behavioral specifications. An OTS/CafeOBJ specification is a behavioral specification that has only observations as behavioral operation symbols and all other operation symbols are behaviorally coherent. For example, the behavioral specification `ARRAY` becomes an OTS/CafeOBJ specification if the action `bop put` is declared as a hidden constructor `op put`. Since `put` is behaviorally reducible and `ARRAY` is terminating, `put` is behaviorally coherent from Theorem 47. In addition, `ARRAY-ADD` with `shift` in Example 49 is an OTS/CafeOBJ specification. By combining our theorem with μ-termination checkers, we can obtain an OTS/CafeOBJ checker that checks whether a CafeOBJ behavioral specification is an OTS/CafeOBJ specification.

6. Conclusion

We proposed the notion of reducibility of operation symbols. In order to restrict the terms to be reduced, we obtained useful properties (Theorem 27 and 42) with respect to the normal forms between the ordinary rewrite relation and CSR relation; these properties allow us to reduce terms by the CSR relation instead of the ordinary rewrite relation. In general, the advantage of CSR is its efficiency of reduction (See Example 22) and termination (See Example 23). As an application of reducibility of operation symbols, we obtained a sufficient condition for behavioral coherence in behavioral specifications (Theorem 47); we also obtained a sufficient condition under which a behavioral specification is an OTS/CafeOBJ specification.

We identified several useful properties by restricting the input terms to μ-ground, S'-sorted, and/or TP-ground terms. We intend to find other interesting restrictions in a future study. Algebraic specifications deal with conditional equations, which are used in many practical case studies. We also intend to extend the reducibility of conditional equations or conditional TRSs in the future.

Acknowledgements

This research was partially supported by Grant-in-Aid for Young Scientists (B) No. 18700024 from the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan, and Kayamori Foundation of Informational Science Advancement.

References

Alarcón, B., Emmes, F., Fuhs, C., J., G., Gutiérrez, R., Lucas, S., Schneider-Kamp, P., Thiemann, R., 2008. Improving Context-Sensitive Dependency Pairs. In: Cervesato, I., Veith, H., Voronkov, A. (Eds.), Proceedings of the XV International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR'08. Vol. to appear of Lecture Notes in Computer Science. Springer-Verlag, Doha, Qatar, p. to appear.

Bidoit, M., Hennicker, R., 1999. Observer complete definitions are behaviorally coherent. In: Proceedings of OBJ/CafeOBJ/Maude Workshop at Formal Methods'99. pp. 83–94.

Comon, H., Jacquemard, F., 2003. Ground reducibility is EXPTIME-complete. Inf. Comput. 187 (1), 123–153.

Diaconescu, R., Futatsugi, K., 1998. CafeOBJ Report. World Scientific.

Diaconescu, R., Futatsugi, K., 2000. Behavioural coherence in object-oriented algebraic specification. Journal of Universal Computer Science 6 (1), 74–96.

Futatsugi, K., Goguen, J. A., Jouannaud, J.-P., Meseguer, J., 1985. Principles of obj2. In: Proceedings of the 12th ACM Symposium on Principles of Programming Languages, POPL. pp. 52–66.

Goguen, J. A., Malcolm, G., 1996. Algebraic Semantics of Imperative Programs. MIT Press, Cambridge, MA, USA.

Guttag, J. V., 1975. The specification and application to programming of abstract data types. Ph.D. thesis, University of Toronto, Toronto, Ont., Canada, Canada.

Guttag, J. V., Horning, J. J., 1978. The algebraic specification of abstract data types. Acta Inf. 10, 27–52.

Hendrix, J., Meseguer, J., 2007. On the completeness of context-sensitive order-sorted specifications. In: Baader, F. (Ed.), RTA. Vol. 4533 of Lecture Notes in Computer Science. Springer, pp. 229–245.

Jouannaud, J.-P., Kounalis, E., June 1986. Automatic proofs by induction in equational theories without constructors. In: Meyer, A. (Ed.), Proceedings of the First Annual IEEE Symp. on Logic in Computer Science, LICS 1986. IEEE Computer Society Press, pp. 358–366.

Kapur, D., Narendran, P., Zhang, H., 1987. On sufficient-completeness and related properties of term rewriting systems. Acta Inf. 24 (4), 395–415.

Lucas, S., January 1998. Context-sensitive computations in functional and functional logic programs. Journal of Functional and Logic Programming 1998 (1).

Lucas, S., 2002. Context-sensitive rewriting strategies. Information and Computation 178 (1), 294–343.

Lucas, S., 2004. Mu-term: A tool for proving termination of context-sensitive rewriting. In: van Oostrom, V. (Ed.), RTA. Vol. 3091 of Lecture Notes in Computer Science. Springer, pp. 200–209.

Nakamura, M., Ogata, K., Futatsugi, K., 2005. Reducible operation symbols for the term rewriting system and their applications. IPSJ Transactions on Programming 46 (SIG 6 (PRO25)), 47–59.

Ogata, K., Futatsugi, K., 2001. Formally modeling and verifying ricart&agrawala distributed mutual exclusion algorithm. In: APAQS. IEEE Computer Society, pp. 357–366.

Ogata, K., Futatsugi, K., 2002. Formal analysis of suzuki & kasami distributed mutual exclusion algorithm. In: Jacobs, B., Rensink, A. (Eds.), FMOODS. Vol. 209 of IFIP Conference Proceedings. Kluwer, pp. 181–195.

Ogata, K., Futatsugi, K., 2003. Flaw and modification of the ikp electronic payment protocols. Inf. Process. Lett. 86 (2), 57–62.

Ohlebusch, E., 2002. Advanced topics in term rewriting. Springer.

Terese, 2003. Term Rewriting Systems. Vol. 55 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.