Proceedings of the International MultiConference of Engineers and Computer Scientists 2012 Vol I,
IMECS 2012, March 14 - 16, 2012, Hong Kong

On Describing Terminating Algebraic Specifications Based on Their Models

Masaki Nakamura, Kazuhiro Ogata and Kokichi Futatsugi

***Abstract*—OBJ algebraic specification languages support automated equational reasoning based on term rewriting systems (TRSs) for specification verification. Termination is one of the most important properties of TRSs. Terminating TRSs guarantee that any equational reasoning terminates in finite times. Although termination is an undecidable property, several sufficient conditions have been proposed, and several termination provers have been developed. In this study, we focus on a way to describe terminating algebraic specifications, that is, the coresponding TRSs are terminating. Existing termination provers may inform us whether a given specification is terminating or not. However, they do not give a guideline to describe terminating specifications. We propose a model-based method for describing terminating specifications. In our method, a model of a given specification can be used for proving its termination. Since specifiers are describing a specification while thinking its model in their mind, our model-based termination methods are suitable for algebraic specifications.**

***Index Terms*—Algebraic specification, Term rewriting, OBJ languages, Termination.**

I. Introduction

OBJ languages [5], [6], [3], [10] are algebraic specification languages, based on order-sorted equational logic. OBJ languages support many advanced features, e.g. module system, typing system with ordered sorts, mix-fix syntax, for describing specifications, and a powerful interactive theorem proving system based on term rewriting systems (TRSs). Executability is an important feature of OBJ languages. Since most OBJ specifications are assumed to be executed for verifying desired properties, the knowledge about TRSs may help us to describe executable specifications. Termination is one of the most important properties of TRSs, which guarantees that execution of the specification must terminate in finite times. Properties of TRSs are important for not only verifying specification but also describing specifications correctly. For example, termination is strongly related to the existence of a solution. Confluence is another important property of TRSs, which is related to the uniqueness of a solution. Reducibility of TRSs helps us to check whether a function is well-defined for all elements of its domain [13], [12], [11]. Termination is considered most important among those fundamental properties. Although confluence and reducibility are also undecidable properties in general, they are decidable and automatically provable when a given TRS is terminating. In this study, we focus on describing algebraic specifications whose TRSs are terminating.

Manuscript received November 30, 2011.

This work was supported in part by Grant-in-Aid for Scientific Research (S) 23220002 from Japan Society for the Promotion of Science (JSPS) and Grant-in-Aid for Young Scientists (B) 22700027 from Ministry of Education, Culture, Sports, Science and Technology (MEXT) Japan.

Masaki Nakamura is with the Department of Information Systems Engineering, Toyama Prefectural University, Japan.

Kazuhiro Ogata and Kokichi Futatsugi are with the School of Information Science, Japan Advanced Institute of Science and Technology (JAIST).

In algebraic specifications, we define functions by a set of equations (axioms). For example, consider a function $even$ which take a natural number and returns $true$ if it is even, otherwise $false$. The function $even$ (and odd) can be defined by the following set of equations: $even(0) = true$, $odd(0) = false$, $even(n+1) = odd(n)$ and $odd(n+1) = even(n)$. We can compute the value of $even(n)$ by applying the equations in the left-to-right manner, like $even(3) = odd(2) = even(1) = odd(0) = false$. Computation of $even(n)$ for any n terminates since the argument of $even$ or odd is strictly decreasing by application of an equation, and will eventually reach zero. Consider another example: the reverse function on lists. The equations $rev(nil) = nil$ and $rev(e;l) = rev(l)@(e;nil)$ define the reverse function rev on lists, where nil is the empty list, $e;l$ is the list whose head element is e and tail list is l, and @ is the list concatenation. The reverse of $1;2;3;nil$ can be computed as follows:

$$\begin{aligned} rev(1;2;3;nil) &= rev(2;3;nil)@(1;nil) \\ &= rev(3;nil)@(2;nil)@(1;nil) \\ &= rev(nil)@(3;nil)@(2;nil)@(1;nil) \\ &= nil@(3;nil)@(2;nil)@(1;nil) \\ &= 3;2;1;nil \end{aligned}$$

The computation terminates since the length of the argument list of rev is strictly decreasing. Intuitively, the above examples are terminating since application of each equation may decrease something strictly. OBJ specifications provide a way to describe abstract data type. We can describe ordinary basic data types like numbers, strings, etc, and more complicated types like lists, sets, collection types and so on. When describing large and complex systems consisting of several data types and functions, it is not easy to describe equations where any combination of their application must terminate. In this study, we propose a model-based method for proving termination, which gives us a mathematical evidence that any computation must terminate.

II. Preliminaries

In this section, we introduce the notion of algebraic specifications [3] and term rewriting systems (TRSs) [13]. Though we give CafeOBJ specifications for explanation, our approach does not restrict the target to CafeOBJ, and can also be applied to other OBJ languages.

A. Algebraic specifications

An algebraic specification consists of signature and axioms. A signature $(S, \leq, \Sigma)$ (abbr. Σ) consists of a set S of sorts, a quasi-order $\leq$ on S, and a set Σ of operation symbols defined on the sorts[1]. An operation symbol $f \in \Sigma$

[1]A binary relation is a quasi-order if it is transitive and reflexive.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

has its rank. A rank consist of an arity and a sort. An arity is a sequence of sorts. An OBJ specification consists of modules. The following is a CafeOBJ module NAT+:

```
mod! NAT+{
  [Zero NzNat < Nat]
  op 0 : -> Zero
  op s_ : Nat -> NzNat
  op _+_ : Nat Nat -> Nat
  vars M N : Nat
  eq N + 0 = N .
  eq M + s N = s(M + N) .
}
```

NAT+ consists of a single module. Roughly speaking, NAT+ denotes natural numbers with the addition function. The symbol 0 stands for zero, and s_ stands for the Peano style successor function. The terms 0, s 0, s s 0, ... are regarded as $0, 1, 2, \ldots$ respectively. In NAT+, the sorts Zero, NzNat and Nat are declared with the subsort relation Zero < Nat and NzNat < Nat. In NAT+, the set $S_{\texttt{NAT+}}$ of sorts is $\{\texttt{Zero}, \texttt{NzNat}, \texttt{Nat}\}$ and the partial order $\leq_{\texttt{NAT+}}$ on $S_{\texttt{NAT+}}$ is the reflexive and transitive closure of $<$, that is, $\leq_{\texttt{NAT+}} = \{(\texttt{Zero}, \texttt{Zero}), (\texttt{NzNat}, \texttt{NzNat}), (\texttt{Nat}, \texttt{Nat}), (\texttt{Zero}, \texttt{Nat}), (\texttt{NzNat}, \texttt{Nat})\}$.

In CafeOBJ, an operation symbol f is declared with its rank as op f : *arity* -> *sort*. An operation symbol is called a constant when the arity is empty. Module NAT+ has the declarations of operation symbols 0, s_ and _+_, where 0 is a constant.

A (Σ, X)-term is a tree whose nodes are operation symbols in Σ and leaves are variables in X. We may omit the prefix (Σ, X)-. For a given signature $(S, \leq, \Sigma)$ and an S-sorted set X of variables[2], the S-sorted set $T_\Sigma(X)_s$ (abbr. T_s) of (Σ, X)-terms is defined as the smallest set satisfying that (1) $X_s \subseteq T_s$ for each $s \in S$, (2) $T_s \subseteq T_{s'}$ for each $s \leq s'$, and (3) $f(\vec{t}_n) \in T_s$ for each $f \in \Sigma$ with the arity $s_1 s_2 \cdots s_n$ and the sort s, and terms $t_i \in T_{s_i}$ $(i \in \{\vec{n}\})$[3]. A term t belonging to T_s is called a term of the sort s. In CafeOBJ, we can indicate the arguments positions of an operation symbol in term expression by underlines. For $t, t' \in T_{\texttt{Nat}}$, the expressions s t and t + t' are terms of Nat.

In the axiom part, we describe equations to be satisfied by the models of the specification. A (Σ, X)-equation is a pair of terms $t, t' \in T_s$, denoted by $t = t'$. In CafeOBJ, a variable x of a sort s is declared with the keyword var (or vars for plural) like var x : s (or vars x y z : s). An equation $t = t'$ is declared with the keyword eq and = like eq t = t'. In NAT+, the variables M and N of the sort Nat and two equations are declared in the axiom part. The first equation means that the terms (or patterns) N + 0 and N are equivalent. The second means that M + s N and s(M + N) are equivalent.

B. Models

A model of a specification is a Σ-algebra which satisfies all equations in the axiom part. For a signature $\Sigma = (S, \leq, \Sigma)$, Σ-algebra M consists of an S-sorted carrier set M where $M_s \subseteq M_{s'}$ for each $s < s'$, and functions $M_f : M_{s_1} \times \cdots \times M_{s_n} \to M_s$ for each op f : $s_1 \cdots s_n$ -> s in Σ. For a Σ-algebra M and an assignment $a : X \to M$, a map $\bar{a} : T \to M$ is defined as follows: $\bar{a}(x) = a(x)$ for each $x \in X$ and $\bar{a}(f(\vec{t}_n)) = M_f(\overrightarrow{\bar{a}(t_n)})$ for each $f(\vec{t}_n) \in T$. We may use a instead of $\bar{a}$ if no confusion arises. A Σ-algebra M satisfies an equation $l = r$ if $a(l) = a(r)$ for each assignment a, denoted by $M_l = M_r$. For a given set E of equations, a (Σ, E)-algebra is a Σ-algebra which satisfies all equations in E.

An initial model M of SP is a kind of the smallest (Σ, E)-algebras, which means that each element $e \in M$ corresponds to some ground (variable-free) term $t \in T_\Sigma(\emptyset)$ (no junk) and $M_t = M_{t'}$ implies $t =_E t'$ for all $t, t \in T_\Sigma(\emptyset)$ [4] (no confusion). The set of all models of SP is denoted by $M(SP)$ and the set of all initial models of SP is denoted by $IM(SP)$. There are two kinds of denotations of CafeOBJ modules: the tight denotation (mod!) and the loose denotation (mod*). The denotation of a basic CafeOBJ module SP, denoted by $[[SP]]$, is defined as follows: When mod! $SP\{\cdots\}$, $[[SP]] = IM(SP)$ and when mod* $SP\{\cdots\}$, $[[SP]] = M(SP)$, where a basic module is a module without any import. By using module imports, we can describe large and complex systems easily. For example, when a module SP' imports SP with the protect mode, each denotational model $M' \in [[SP']]$ includes a model $M \in [[SP]]$ of the imported module as it is, denoted by mod* $SP'\{$pr(SP) ...$\}$. For example, we can describe specifications of data types, and a system specification imports them with the protected mode. We omit details of denotation of modules including module imports (See the reference [3]).

Consider the following $\Sigma_{\texttt{NAT+}}$-algebras $B, N, Z, N*$ [5].

- B is a Boolean algebra: $B_{\texttt{Zero}} = \{false\}$, $B_{\texttt{NzNat}} = \{true\}$, $B_{\texttt{Nat}} = \{false, true\}$, $B_{\texttt{0}} = false$, $B_{\texttt{s}}(x) = true$, $B_{+}(x, y) = x \vee y$.
- N is an algebra of natural numbers: $N_{\texttt{Zero}} = \{0\}$, $N_{\texttt{NzNat}} = \mathcal{N}_+ = \{1, 2, \ldots\}$, $N_{\texttt{Nat}} = \mathcal{N} = \{0, 1, 2, \ldots\}$, $N_{\texttt{0}} = 0$, $N_{\texttt{s}}(x) = x + 1$, $N_{+}(x, y) = x + y$.
- Z is an algebra of integers: Z_x is same with N_x except $Z_{\texttt{Nat}} = \mathcal{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$.
- N^* is another algebra of natural numbers: N^*_x is same with N_x except $N^*_{+}(x, y) = x \times y$.

All B, N, Z, N^* are $\Sigma_{\texttt{NAT+}}$-algebras. Only N^* is not a $(\Sigma_{\texttt{NAT+}}, E_{\texttt{NAT+}})$-algebra since the equations are not satisfied, for example, $N^*_{\texttt{N+0}} = a(\texttt{N}) \times 0 = 0 \neq a(\texttt{N}) = \texttt{N}^*_{\texttt{N}}$ for some a which assigns N to a positive integer. A specification NAT+ denotes N, i.e. $N \in [[\texttt{NAT+}]]$. Z has negative integers (junk) and B interpret all positive integers to one element $true$ (confusion), that is, $B_{\texttt{ss0}} = B_{\texttt{s0}} = true$ though s s 0 $\neq_E$ s 0. Thus, $Z, B \notin [[\texttt{NAT+}]]$.

C. Term rewriting systems

An OBJ specification is executable, that is, some term may be reduced into its normal form automatically. The execution is based on a term rewriting system (TRS), which is a set of

[2] An S-sorted set A is a family of sets A_s $(s \in S)$. We may omit the subscript s if no confusion arises, for example, $a \in A$ instead of $a \in A_s$.

[3] We may write $\vec{a}_n$ instead of $a_1, \ldots, a_n$, and $\vec{n}$ instead of $1, \ldots, n$.

[4] $t =_E t'$ means that the equation can be deduced from E.

[5] In this paper, the contents of SP may be written like X_{SP}. For example, Σ_{SP} and E_{SP} are the signature and the set of all equations of a specification (or a module) SP respectively.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

rewrite rules. In a TRS, an equation is regarded as a left-to-right rewrite rule. A term is rewritten by replacing an instance of the left-hand side of an equation with the corresponding instance of the right-hand side. A term is reduced by applying rewrite rules to a given term until it cannot. The following is an example of reducing the term `s 0 + s s 0` in `NAT+`:

```
CafeOBJ> red in NAT+ : s 0 + s s 0 .
-- reduce in NAT+ : s 0 + s (s 0)
s (s (s 0)) : NzNat
```

The following is the trace of the above reduction showed by CafeOBJ system: `s 0 + s s 0` $\rightarrow$ `s(s 0 + s 0)` $\rightarrow$ `s s(s 0 + 0)` $\rightarrow$ `s s s 0`. The underlined subterms are matched with left-hand sides of equations, and replaced with the corresponding right-hand sides. The first and second rewrites are obtained by the second equation `eq M + s N = s(M + N)` and the last rewrite is obtained by the first equation `eq M + s N = s(M + N)`. The input and output terms of the CafeOBJ reduction command are equivalent in all models [3]. Thus, the above execution is a proof of $1 + 2 = 3$ in `NAT+`. The rewrite relation obtained by an equation e is denoted by $\rightarrow_e$, and the rewrite relation obtained by E is denoted by $\rightarrow_E$. The reflexive and transitive closure of $\rightarrow$ is denoted by $\rightarrow^*$. The above reduction can be written as `s 0 + s s 0` $\rightarrow^*_{\texttt{NAT+}}$ `s s s 0`.

D. Termination

A specification $SP = (\Sigma, E)$ (or a TRS E) is terminating if there is no infinite rewrite sequence [13]. Let E' be the set obtained by adding `eq s(0 + N) = 0 + s N` to `NAT+`. Then, E' is not terminating since there is an infinite rewrite sequence `s(0 + 0)` $\rightarrow_{E'}$ `0 + s 0` $\rightarrow_{E'}$ `s(0 + 0)` $\rightarrow_{E'} \cdots$. Several useful proof methods for termination have been proposed and several termination provers have been developed. We can use those tools to prove a given specification to be terminating. We introduce two classical approaches and one recent powerful notion for termination.

1) Recursive Path ordering: Recursive path ordering (RPO) is one of the most classical syntactic termination methods. Let Σ be a signature, X a set of variables, and $\trianglerighteq \subset \Sigma \times \Sigma$ a partial order on operation symbols. The RPO $>_{rpo} \subset T(\Sigma, X) \times T(\Sigma, X)$ is defined as follows:

$t >_{rpo} t' \overset{def}{\iff} t = f(\vec{t_m})$ and

1) $\exists\, i \in \{\vec{m}\}.t_i \geq_{rpo} t'$, or
2) $t' = g(\vec{t'_n})$, $f \rhd g$ and $\forall j \in \{\vec{n}\}.t >_{rpo} t'_j$, or
3) $t' = g(\vec{t'_n})$, $f \sim g$ and $\{\!|\vec{t_m}|\!\} >^{mul}_{rpo} \{\!|\vec{t'_n}|\!\}$,

where $a \rhd b \overset{def}{\iff} a \trianglerighteq b \wedge a \neq b$, $a \sim b \overset{def}{\iff} a \trianglerighteq b \wedge b \trianglerighteq a$. $\{\!| \ldots |\!\}$ stands for a multiset. The partial order $>^{mul}$ is a multiset order w.r.t. $>$[6]. The following property holds[7].

Proposition 2.1: [13] Let E be a set of equations. If there exists $\trianglerighteq$ such that $l >_{rpo} r$ for each $l = r \in E$, then E is terminating.

[6]A multiset is a collection where duplicated elements are allowed. For example, $\{\!|a, a, b|\!\}$ and $\{\!|a, b|\!\}$ are different. $FM(A)$ is the set of all finite multisets whose elements are of a given set A, e.g. $\{\!|0, 2, 2|\!\} \in FM(\mathcal{N})$. A multiset order $>^{mul} \subseteq FM(A) \times FM(A)$ w.r.t. a partial $> \subseteq A \times A$ is defined as follows: $M_1 >^{mul} M2 \Leftrightarrow \exists X, Y \in FM(A).[X \neq \emptyset \wedge X \subseteq M_1 \wedge M_2 = (M_1 \setminus X) + Y \wedge \forall y \in Y. \exists x \in X. x > y]$. For example, $\{\!|2, 2, 3|\!\} >^{mul} \{\!|1, 1, 2, 3|\!\}$ holds, where $X = \{\!|2|\!\}$ and $Y = \{\!|1, 1|\!\}$.

[7]We assume Σ is finite. Thus, there is no infinite sequence $f_1 \rhd f_2 \rhd \cdots$.

Let $+ \rhd s \rhd 0$. Then, `N + 0` $>_{rpo}$ `N` and `M + s(N)` $>_{rpo}$ `s(M + N)` hold. The former is trivial from `N` $\geq_{rpo}$ `N` (from 1). The latter is obtained as follows: `s(N)` $>_{rpo}$ `N` holds (from 1). Since $\{\!|\texttt{M}, \texttt{s(N)}|\!\} >^{mul}_{rpo} \{\!|\texttt{M}, \texttt{N}|\!\}$, we have `M + s(N)` $>_{rpo}$ `M + N` (from 3). Therefore, `M + s(N)` $>_{rpo}$ `s(M + N)` holds (from 2 with $+ \rhd s$).

2) Polynomial interpretations: In semantic methods, termination is proved based on a well-founded monotone compatible Σ-algebra, that is, a $(S, \leq, \Sigma)$-algebra together with a partial order on M_S which is well-founded, monotone and compatible with E. A partial order $>$ is well-founded if there is no infinite decreasing chain $a_1 > a_2 > \cdots$. It is monotone if $M_f(\vec{a}_n) > M_f(\vec{b}_n)$ whenever $f \in \Sigma$, $\exists i \in \{\vec{n}\}.a_i > b_i$ and $\forall j \neq i.a_j = b_j$. It is compatible with E if $M_l > M_r$ whenever $l = r \in E$.

Proposition 2.2: [13] A specification E is terminating if and only if there exists a well-founded monotone compatible Σ-algebra.

We show that termination of `NAT+` by giving a Σ-algebra M where operation symbols are interpreted into polynomials on natural numbers. Let $M_0 = 1$, $M_{\texttt{s}}(x) = x + 1$ and $M_+(x, y) = x + 2y + 1$. Let a be an assignment such that $a(\texttt{N}) = n$ and $a(\texttt{M}) = m$. Then the first equation in `NAT+` is ordered by $>$: $M_{\texttt{N+0}} = M_+(n, M_0) = n + 3 > n$. So is the second equation: $M_{\texttt{M+sN}} = M_+(m, M_{\texttt{s}}(n)) = m + 2n + 3 > m + 2n + 2 = M_{\texttt{s}}(M_+(m, n)) = M_{\texttt{s(M+N)}}$. Thus, M is a proof of termination of `NAT+`.

3) Dependency pairs: Recently a landmark in termination methods has been proposed, called the dependency pair (DP) method [1]. A dependency pair is a pair of terms obtained from an equation, which is essential for analyzing termination. Combinations of the DP method and classical methods like RPO, Polynomial ordering, etc, extend a class of TRSs which can automatically be proved terminating. Most of the active termination tools adopt the DP method.

A root symbol, i.e. the symbol at the root position, of a left-hand side of an equation is called a defined symbol. For example, `_+_` is a defined symbol and `0` and `s_` are not. A rename of an operation symbol f is denoted by $f^\#$. We assume that $f^\#$ is not included in the original signature Σ. The term whose root symbol is renamed is denoted by $t^\#$. For example, $t^\#$ = $+^\#(\texttt{0}, \texttt{N})$ when t = `0 + N`. For the readability, we write $f^\#(\cdots)$ even if f is declared with the underlines, like $\_f\_$.

Let $l = r \in E$ be an equation. A dependency pair is a pair $(l^\#, u^\#)$ of terms where u is a subterm of r whose root symbol is a defined symbol [1]. The all dependency pairs obtained from E is denoted by $DP(E)$. A chain of dependency pairs is a (possibly infinite) sequence $(l_i^\#, u_i^\#)$ $(i = 0, 1, 2, \ldots)$ of dependency pairs in $DP(E)$ whose variables are distinct such that dependency pairs do not share variables and there is a substitution θ such that $u_i^\#\theta \rightarrow^*_E l_{i+1}^\#\theta$ for each $i = 0, 1, 2, \ldots$ [8]. Termination can be characterized by a chain of dependency pairs.

Proposition 2.3: [1] A specification is terminating if and only if there is no infinite dependency chain.

$DP(\texttt{NAT+}) = \{(+^\#(M, sN), +^\#(M, N))\}$ whose dependency pair is made from the second equation `M + s N`

[8]A substitution θ is a map from X to T. For a term t, the term whose all variable x are replaced with $\theta(x)$ is denoted by $t\theta$. For example, let θ(`M`) = `0` and θ(`N`) = `s M`. Then, (`M + s N`)θ =`0 + s s M`.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

= s(M + N) of NAT+. The following sequence is a DP chain:

$$(+^{\#}(M_0, s\,N_0), +^{\#}(M_0, N_0))$$
$$(+^{\#}(M_1, s\,N_1), +^{\#}(M_1, N_1))$$
$$(+^{\#}(M_2, s\,N_2), +^{\#}(M_2, N_2))$$

where each M_i and N_i are variables, since there exists θ such that $+^{\#}(M_i, N_i)\theta = +^{\#}(M_{i+1}, sN_{i+1})\theta$ $(i = 0, 1)$, where $\theta(M_i) = 0$ for each $i = 0, 1, 2$, and $\theta(N_0) = s\,s\,0$, $\theta(N_1) = s\,0$ and $\theta(N_2) = 0$. Note that $\rightarrow^*$ includes the zero-step rewrite, i.e. $=\subseteq\rightarrow^*$. Intuitively there is no infinite DP-chain in $DP(\mathtt{NAT}+)$. Let us assume an infinite DP chain $(+^{\#}(M_i, sN_i), +^{\#}(M_i, N_i))$ $(i = 0, 1, 2, \ldots)$. Since $+^{\#}(M_i, N_i)\theta \rightarrow^*_{\mathtt{NAT}+} +^{\#}(M_{i+1}, sN_{i+1})\theta$ and the renamed root symbol $+^{\#}$ must not be rewritten, it holds that $N_i\theta \rightarrow^*_{\mathtt{NAT}+} s\,N_{i+1}\theta$. In the model, $N_i\theta$ and $s\,N_{i+1}\theta$ are interrupted into a same natural number since they are connected by $\rightarrow^*_{\mathtt{NAT}+}$, i.e. they are deducible from the equations in NAT+. Therefore, $a(N_i\theta) = a(s\,N_{i+1}\theta) > a(N_{i+1}\theta)$ and it contradicts to the well-foundedness of $>$ on natural numbers.

III. Proving termination based on models

Existing methods and tools for proving termination, including those we introduced above, are used after describing specifications. Those methods and tools tell us "Yes", "No" or "Unknown" for the query of whether a specification is terminating or not, and they do not tell how to describe terminating ones. One way to give a guideline to describe terminating specification is to give a syntactic condition under which any specification satisfying the condition is terminating. However, it may restrict the flexibility of OBJ languages. Another way is to describe a specification with a terminating proof. To do it, specifiers are expected to be familiar with a termination method. Our motivation is to propose a termination method which is easy to be learned and used by OBJ users.

Since algebraic specifications denote Σ-algebras, semantic methods based on well-founded Σ-algebras seemingly may fit for algebraic specifications. However, as we showed above, they are completely different models. Equations $l = r$ are interpreted into the equalities $M_l = M_r$ in their denotational models, while they are interpreted to the inequalities $M_l > M_r$ in well-founded Σ-algebras for termination, like the above example of polynomial interpretations for NAT+, where $M_+(x, y) = x + 2y + 1$.

The DP method gives us a solution for reducing the gap between denotational models of specifications and well-founded models for termination. In classical methods, a rewrite relation corresponds to a well-founded order directly. Thus, each rewrite relation should be strictly decreased. In the DP method, only dependency pairs are strictly decreased, and rewrite relations are not needed to be strictly decreased. The following proposition holds.

Proposition 3.1: [1] A specification is terminating if there exists a quasi-order $\gtrsim$ on terms which satisfies the following conditions:

1) $>$ is well-founded,
2) $\gtrsim$ is weakly monotonic,
3) $\gtrsim$ and $>$ are closed under substation,
4) $l \gtrsim r$ for each $l = r \in E$ and
5) $l^{\#} > u^{\#}$ for each $(l^{\#}, u^{\#}) \in DP(E)$

where $>$ is defined as $a > b \Leftrightarrow a \gtrsim b \wedge b \not\gtrsim a$. A quasi-order $\gtrsim$ is weakly monotonic if $\forall i \in \{\vec{n}\}.s_i \gtrsim t_i$ implies $\forall f.f(\vec{s}_n) \gtrsim f(\vec{t}_n)$.

Note that Σ and T do not include renamed operation symbols $g^{\#}$. The point of the above proposition is that equations are allowed to be interpreted into equations because of $=\subseteq\gtrsim$. Thus we can use a model of a specification for constructing a model for proving termination. For a given specification, we define a specification $SP^{\#}$ importing SP such that a model of $SP^{\#}$ can be used for proving termination of SP.

Definition 3.2: Let SP be a specification. The DP-specification of SP, denoted by $SP^{\#}$, is defined as the loose module including the followings:

1) a protected import of SP,
2) a sort DP, which is not included in SP,
3) an operation symbol `op _>_ : DP DP -> Bool` [9],
4) an operation symbol `op` $f^{\#}$ `:` $s_1 \cdots s_n$ `-> DP` for each `op` f `:` $s_1 \cdots s_n$ `->` s in SP,
5) an equation `eq` $l^{\#}$ `>` $u^{\#}$ `= true` for each dependency pair $(l^{\#}, u^{\#}) \in DP(SP)$.

Let $SP = ((S, \leq, \Sigma), E)$ be a specification. A DP-specification $SP^{\#}$ is formed as follows:

```
mod* SP# {
  pr(SP)
  [DP]
  op _>_ : DP DP -> Bool
  op f1# : ... -> DP
  op f2# : ... -> DP
  ...
  ep f1#(...) > f1#(...) = true .
  ep f2#(...) > f1#(...) = true .
  ep f2#(...) > f2#(...) = true .
  ...
}
```

where `pr(SP)` stands for the import of SP with the protect mode. The arguments of $f^{\#}$ (omitted as `...`) are same with the original f. For each `ep` $f_i^{\#}(\ldots)$ `>` $f_j^{\#}(\ldots)$ `= true`, there exists $(f_i^{\#}(\ldots), f_j^{\#}(\ldots))$ in $DP(SP)$, and for each $(f_i^{\#}(\ldots), f_j^{\#}(\ldots)) \in DP(SP)$, $SP^{\#}$ includes `ep` $f_i^{\#}(\ldots)$ `>` $f_j^{\#}(\ldots)$ `= true`. Let us consider a model of $SP^{\#}$. The sort DP can be interpreted into an arbitrary set, and renamed operation symbols $f^{\#}$ can also be arbitrary functions. Since BOOL denotes the Boolean algebra B and `true` stands for $true$ of B, the operation symbol _>_ is interpreted into a predicate which returns $true$ for each dependency pair. Since the import of SP is protected, a model M' of $SP^{\#}$ interprets all SP's contents as same as $M \in [[SP]]$, i.e., for each $M' \in [[SP^{\#}]]$ there exists $M \in [[SP]]$ such that $M_s = M'_s$ for each $s \in S$ and $M_f = M'_f$ for each $f \in \Sigma$.

A model $M \in [[SP^{\#}]]$ is well-founded if $M_>$ is well-founded. We have the following theorem.

Theorem 3.3: If there exists a well-founded model $M \in [[SP^{\#}]]$, then SP is terminating.

[9] Each CafeOBJ specification implicitly imports the built-in module BOOL, which includes the sort Bool, the constants true and false of Bool, and the operation symbols _and_, _or_, etc of Boolean operations. BOOL denotes the Boolean algebra B, i.e. $[[\mathtt{BOOL}]] = \{B\}$, where B interprets elements in BOOL naturally.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Proof of Theorem 3.3 Let $SP = ((S, \leq, \Sigma), E)$. We give a S-sorted quasi-order $\gtrsim$ satisfying Proposition 3.1. Define $\gtrsim$ as follows:

a) For each $s \in S$ and $t, t' \in T_s$, $t \gtrsim t' \Leftrightarrow M_t = M_{t'}$.
b) For each $t, t' \in T_{\mathtt{DP}}$, $t \gtrsim t' \Leftrightarrow \forall a. M_>(a(t), a(t'))$.

Note that t and t' in the case a) are terms of the original SP, i.e., no renamed symbols are included, and their sort are not DP. Consider the strict part $>$ of $\gtrsim$. For each $t, t' \in T_s$ ($s \in S$), Since $t \gtrsim t' \Rightarrow M_t = M_{t'} \Rightarrow M_{t'} = M_t \Rightarrow t' \gtrsim t$ from the definition of the case a), $t > t'$ does not hold. For each $t, t' \in T_{\mathtt{DP}}$, if $t \gtrsim t'$ then $t' \not\gtrsim t$ from the well-foundedness of $M_>$, therefore $t > t'$ holds. Thus, we have the following lemma: $t > t' \Leftrightarrow \forall a. M_>(a(t), a(t'))$ for each $t, t' \in T_{\mathtt{DP}}$.

We show $\gtrsim$ satisfies 1) - 5) of Proposition 3.1.

1) Since $M_>$ is well-founded, $>$ is well-founded from the lemma.
2) Assume $\forall i \in \{\vec{n}\}. s_i \gtrsim t_i$. We show $f(\vec{s}_n) \gtrsim f(\vec{t}_n)$ for each operation symbol f. The sorts of s_i, t_i are not DP since no operation symbol can take terms of DP in its argument. From a), $M_{s_i} = M_{t_i}$ and thus $M_{f(\vec{s}_n)} = M_{f(\vec{t}_n)}$. From a) again, we have $f(\vec{s}_n) \gtrsim f(\vec{t}_n)$.
3) Let θ be an arbitrary substitution and $t, t' \in T_s$ for some $s \in S$. Assume $t \gtrsim t'$. From a) $M_t = M_{t'}$, and it means that $\forall a. a(t) = a(t')$. Let a_0 be an arbitrary assignment. Define the assignment a'_0 as $a'_0(x) = a_0(\theta(x))$. Then, $a_0(t\theta) = a'_0(t)$. From the assumption, $a'_0(t) = a'_0(t')$. Thus, $\forall a. a(t\theta) = a'(t) = a'(t') = a(t'\theta)$holds, and $\gtrsim$ is closed under substitution θ. For $>$, it can be proved similarly.
4) Since M satisfies $l = r \in E$, i.e. $M_l = M_r$. Thus, $l \gtrsim r$.
5) Let $(l^\#, u^\#) \in DP(SP)$. From the definition of $SP^\#$, $M_>(a(l^\#), a(u^\#))$ holds for each assignment a. From the lemma, $l^\# > u^\#$. □

IV. Describing terminating specifications

In this section, we show examples of proving termination based on Theorem 3.3.

A. The specification NAT+

Since $DF(\mathtt{NAT+}) = \mathtt{\_+\_}$ and $DP(\mathtt{NAT+}) = (\ \mathtt{+\#(M,\ s\ N)},\ \mathtt{+\#(M,\ N)}\ )$, the DP-specification NAT+# is given as follows:

```
mod* NAT+#{
  pr(NAT+)
  [DP]
  op _>_ : DP DP -> Bool
  op +# : Nat Nat -> DP
  vars M N : Nat
  eq +#(M, s N) > +#(M, N) = true .
}
```

Note that the terms +#(M, s N) and +#(M, N) are of the sort DP. Let $N \in [[\mathtt{NAT+}]]$. We give a model M of NAT+# as follows: $M_x = N_x$ for all $x \in$ NAT+ where x is Nat, NzNat, Zero, 0, s_ or _+_. The sort DP is interpreted into the set of the natural numbers, i.e. $M_{\mathtt{DP}} = \mathcal{N}$. The operation symbol _>_ is interpreted into the ordinary order on $\mathcal{N}$. The renamed operation symbol +# is interpreted as $M_{+\#}(x, y) = y$. Then, M satisfies the equations in NAT+#. Since $M_x = N_x$ for all $x \in$ NAT+, M satisfies all equations in NAT+. We have $M_{+\#(\mathtt{M,sN})} = M_{\mathtt{N}} + 1 > M_{\mathtt{N}} = M_{+\#(\mathtt{M,N})}$. Thus, M satisfies the equation +#(M, s N) > +#(M, N) = true. Since the ordinary order on $\mathcal{N}$ is well-founded, $M \in [[\mathtt{NAT+\#}]]$ is a well-founded model.

The reason why $M_{+\#}(x, y)$ is defined as y is because the equations in NAT+ define the operation symbol _+_ recursively on the second argument. Such an interpretation is not allowed in the classical semantic methods, where the strict order $>$ should be monotonic, i.e. $f(\vec{a}_n) > f(\vec{b}_n)$ whenever $\exists i. a_i > b_i$ and $\forall j \neq i. a_j = b_j$. Consider $f(x, y) = y$. Then, $f(a, c) = f(b, c)$ even if $a > b$.

B. The specification LIST

The following specification LIST specifies lists on natural numbers and the reverse function on the lists:

```
mod! LIST{
 pr(NAT+) [List]
 op nil : -> List
 op _;_ : Nat List -> List
 op rev : List -> List
 op revApp : List List -> List
 vars L L' : List
 var N : Nat
 eq revApp(nil, L) = L .
 eq revApp(N ; L, L')
     = revApp(L, N ; L') .
 eq rev(L) = revApp(L, nil) .
}
```

The following is the CafeOBJ execution of applying the reverse function rev to the list $[2, 1, 0]$, which returns the list $[0, 1, 2]$:

```
LIST> red rev(s s 0 ; s 0 ; 0 ; nil) .
...
0 ; (s 0 ; (s (s 0) ; nil)) : List
```

LIST denotes the following model L. All elements in NAT+ are interpreted as same as $N \in [[\mathtt{NAT+}]]$. The sort List is interpreted into the set $List(\mathcal{N})$ of all lists on natural numbers. The operation symbols nil, _;_, rev and revApp are interpreted into the empty list $[\,]$, the list constructer $L_;(x, [\vec{x}_n]) = [x, \vec{x}_n]$, the reverse function $rev : List(\mathcal{N}) \to List(\mathcal{N})$, and the auxiliary function $L_{\mathtt{revApp}}(l_0, l_1) = rev(l_0)@l_1$ where @ is the concatenation of lists. For example, $L_{\mathtt{revApp}}([3, 4], [2, 1]) = [4, 3, 2, 1]$.

We show termination of LIST. The DP-specification LIST# is given as follows:

```
mod* LIST#{
 pr(LIST) [DP]
 op _>_ : DP DP -> Bool
 op +# : Nat Nat -> DP
 op rev# : List -> DP
 op revApp# : List List -> DP
 vars M N : Nat
 vars L L' : List
 eq +#(M, s N) > +#(M, N) = true .
 eq revApp#(N ; L, L')
     > revApp#(L, N ; L') = true .
```

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

```
  eq rev#(L) > revApp#(L, nil) = true .
}
```

The set $DF(\texttt{LIST})$ of all defined symbols in LIST is $\{\texttt{rev}, \texttt{revApp}, \_+\_\}$. Note that we should consider the defined symbols in the imported module NAT+. We give a model $M \in \texttt{LIST\#}$. $M_x = L_x$ for all elements x in the original LIST. The sort DP is given as $M_{\texttt{DP}} = \mathcal{N}$. The renamed functions are interpreted as follows: $M_{+\#}(x, y) = y$, $M_{\texttt{revApp\#}}(l, l') = |l|$ and $M_{\texttt{rev\#}}(l) = |l| + 1$, where $|l|$ is the length of the list l. Since the operational symbol revApp is inductively defined on the length of the list in the first argument, we give $M_{\texttt{revApp\#}}(l, l') = |l|$. Since rev is defined by revApp, $M_{\texttt{rev\#}}(l)$ is defined larger than $M_{\texttt{revApp\#}}(l, l')$. Then, $M \in [[\texttt{LIST\#}]]$ is well-founded, and LIST is terminating from Theorem 3.3.

The notion of dependency cycles has been proposed for analyzing which dependency pairs are essential to prove termination [1]. By using the dependency cycles, the dependency pair (rev#(L), revApp#(L, nil)) can be removed from elements of the dependency sequence. Thus, we do not need to give an interpretation for rev. See [1] for details.

V. Conclusion

We proposed a termination method for algebraic specifications based on their models. In our method, models of specifications are used for obtaining models for proving termination unlike classical semantic methods, e.g. polynomial interpretation. A combination of the dependency pair method and polynomial interpretation is known as a useful approach to give powerful termination prover [2], and is adopted by several termination provers like AProVE[10], CiME[11] and TTT[12]. Our method can be seen as a generalization of the combination of the dependency pair method and polynomial interpretation. In polynomial interpretation, all operation symbols are interpreted into a polynomials on natural numbers, where the order on natural numbers is well-founded. In our method, the sorts and the operation symbols in the original specification are interpreted by a denotational model of the specification, and the renamed operation symbols are interpreted into functions on a well-founded ordered set. Unfortunately, our method is not suitable for automated reasoning since it is impossible in practice to implement all models of algebraic specifications to be described in advance. Our method is useful for describing terminating specifications since a model of the specification exists on the mind of the specifiers. The purpose of a specifier is to describe specification of some model, for example, a system to be implemented.

In this article, we restrict target specifications to simple ones, that is, no operation attributes and no conditional equations, which are useful to describe practical specifications. For example, associate and commutative operation attributes are very useful to describe a collection type. The associativity $((a \circ b) \circ c = a \circ (b \circ c))$ and the commutativity $(a \circ b = b \circ a)$ of a binary operation symbol $(\_ \circ \_)$ are commonly used in algebraic specifications, however, they make a specification to be non-terminating and/or non-confluent. OBJ languages provide operation attributes to solve this problem. We describe associativity and commutativity as operation attributes instead of describing their equations directly. In reduction, those operation attributes are implemented in the matching phase, that is, the term $(a \circ b) \circ c$ is matched with the rewrite rule $(c \circ b) \circ a = d$ and is rewritten to d as a one-step rewriting. AC-TRS is a generalization of TRS, in which we declare associative and commutative operation symbols. Termination of AC-TRS has also been studied. Application of the dependency pair method to AC-TRS has been studied, for example, in [8], [7]. We may apply these results to obtain DP-specification with AC-symbols. A conditional equation is an equation equipped with a condition c, which is a term of Boolean term, formed of $l = r$ if c. A model has to satisfy the body part $l\theta = r\theta$ if the condition part $c\theta$ is interpreted into $true$. In reduction, before applying the body part to a given term, the condition part is checked whether to be reduced into true. CTRS is a generalization of TRS, in which we declare conditional rewrite rules. Besides infinite rewrite sequences we should care infinite calls of reductions of condition parts. Effective termination [12] and operational termination [9], [4] give the notion of termination for CTRS. Termination methods for effective (or operational) termination have been proposed. In the existing methods, effective (or operational) termination is proved by transforming a given CTRS into a TRS whose termination implies effective (or operational) termination of the original CTRS. Transformations may not preserve models of the original specification. To give a DP-specification for CTRS is one of our future work.

[10] http://aprove.informatik.rwth-aachen.de/

[11] http://cime.lri.fr/

[12] http://colo6-c703.uibk.ac.at/ttt/

References

[1] T.Arts and J.Giesl, Termination of term rewriting using dependency pairs, Theor. Comp. Sci. 236 (2000), 133-178.

[2] E.Contejean, C.Marche, A.P.Tomas, X.Urbain, Mechanically Proving Termination Using Polynomial Interpretations, Journal of Automated Reasoning, Vol.34, No.4, 325 -363, 2005.

[3] R.Diaconescu and K.Futatsugi, "CafeOBJ report," World Scientific, 1998.

[4] F.Duran, S.Lucas, C.Marche, J.Meseguer, and X.Urbain, Proving operational termination of membership equational programs, Higher-Order and Symbolic Computation, 21(1-2), pp. 59?88, 2008.

[5] K.Futatsugi, J.A.Goguen, J.-P.Jouannaud, and J.Meseguer, Principles of OBJ2., *Proceedings of the 12th ACM Symposium on Principles of Programming Languages, POPL*, pp. 52-66, 1985.

[6] J.A.Goguen, T.Winkler, J.Meseguer, K.Futatsugi and J.-P.Jouannaud, *Software Engineering with OBJ: Algebraic Specification in Action*, Kluwers Academic Publishers, 2000, chapter Introducing OBJ*.

[7] K.Kusakari, M.Nakamura and Y.Toyama, Elimination Transformations for Associative-Commutative Rewriting Systems, Journal of Automated Reasoning, Vol.37, No.3, pp.205-229, Oct 2006.

[8] K.Kusakari and Y.Toyama, On Proving AC-Termination by AC-Dependency Pairs, IEICE Transactions on Information and Systems, Vol.E84-D, No.5, pp.604-612, 2001.

[9] S.Lucas, C.Marche and J.Meseguer, Operational termination of conditional term rewriting systems, Information Processing Letter, 95(4), pp. 446-453, 2005.

[10] *The Maude System*, http://maude.cs.uiuc.edu/.

[11] M.Nakamura, K.Ogata, and K.Futatsugi, Reducibility of operation symbols in term rewriting systems and its application to behavioral specifications, Journal of Symbolic Computation, Volume 45, Issue 5, May, 2010.

[12] E.Ohlebusch, *Advanced topics in term rewriting*, Springer, 2002.

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

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)