Incremental Checking of Well-Founded Recursive Specifications Modulo Axioms *

Felix Schernhammer
Vienna University of Technology, Austria
felixs@logic.at

José Meseguer
University of Illinois at Urbana-Champaign, USA
meseguer@uiuc.com

Abstract

We introduce the notion of well-founded recursive order-sorted equational logic (OS) theories modulo axioms. Such theories define functions by well-founded recursion and are inherently terminating. Moreover, for well-founded recursive theories important properties such as confluence and sufficient completeness are modular for so-called fair extensions. This enables us to incrementally check these properties for hierarchies of such theories that occur naturally in modular rule-based functional programs. Well-founded recursive OS theories modulo axioms contain only commutativity and associativity-commutativity axioms. In order to support arbitrary combinations of associativity, commutativity and identity axioms, we show how to eliminate identity and (under certain conditions) associativity (without commutativity) axioms by theory transformations in the last part of the paper.

Categories and Subject Descriptors F.3.1 [*LOGICS AND MEANINGS OF PROGRAMS*]: Specifying and Verifying and Reasoning about Programs; D.2.4 [*SOFTWARE ENGINEERING*]: Software/Program Verification; D.3.3 [*PROGRAMMING LANGUAGES*]: Language Constructs and Features; D.1.1 [*PROGRAMMING TECHNIQUES*]: Applicative (Functional) Programming

General Terms Theory, Verification

Keywords Well-founded recursive theory, order-sorted rewriting modulo axioms, termination, confluence, sufficient completeness

1. Introduction

Scalability is a big, unsolved challenge in formal reasoning about executable algebraic specifications. When using such specifications as programs and reasoning about their correctness, we often need to check basic properties such as confluence, termination, and sufficient completeness. This is quite manageable for small specifications, but when dealing with larger specifications corresponding to realistic programs, we can encounter severe tool performance barriers. For example, a non-built-in specification in Maude of the natural numbers, which is the exact counterpart of Maude's built-in `NAT` module, cannot be proved terminating by the MTT tool, which performs a relatively simple transformation to make the order-sorted specification unsorted and then invokes the AProVE tool with a 900 second timeout. Likewise, Mu-term cannot prove the same specification terminating with the same timeout, even though both AProVE and Mu-Term are state-of-the-art tools. In a similar way, particularly in the presence of AC axioms, a large number of critical pairs is often generated when checking the local confluence of specifications. For example, a small AC specification of hereditarily finite sets with only 26 equations already generates 1027 critical pairs when using the Maude Church-Rosser Checker [10]. Modularity is crucial.

Modular methods for termination and confluence (for a good survey up to 2002 see [24]) are certainly helpful. However: (i) some of these methods make quite strong requirements (e.g., disjointness) on the kind of modularity they allow; (ii) little seems to be known about the modularity of sufficient completeness; and (iii) the modularity results we are aware of do not deal with sorts and subsorts, nor (except for, e.g., [19, 23]) with rewriting modulo axioms, which are key features of state-of-the-art rule-based languages such as ASF+SDF [30], ELAN [5], CafeOBJ [11], and Maude [6].

Our Approach is based on the observation that in practice algebraic specifications are often *recursive function definitions* based on *constructor patterns*, and whose right-hand sides involve *recursive calls* to the same and/or previously defined functions on *smaller arguments* in the *well-founded* subterm ordering. This includes, but goes beyond, the very common case of primitive-recursive definitions. For example, the equations defining Ackerman's function,

$$A(0,n) = s(n) \qquad A(s(m),0) = A(m,1)$$
$$A(s(m),s(n)) = A(m,A(s(m),n))$$

exemplify a well-founded recursive function definition based on natural number constructor patterns which is not primitive-recursive. Such specifications define *total* (i.e., terminating) functions on the set of constructor terms. Furthermore, they naturally form *hierarchies*, so that previously-defined functions can be used to define more complex ones. For example, natural number exponentiation can be recursively defined in terms of multiplication, which can in turn be recursively defined in terms of addition.

The main goal of this work is to reduce the checking of *confluence, termination, and sufficient completeness* for algebraic specifications based on well-founded recursive function definitions to relatively simple *incremental checks* on the module hierarchies containing such definitions. However, in order to be practically useful for rule-based languages, the notion of well-founded recursive function definition *needs to be generalized* to support: (a) mutually recursive definitions; (b) sorts, subsorts, and subsort overload-

* The first author has been supported by the Austrian Academy of Sciences under grant number 22.361 and by the Austrian Marshall Plan Foundation in the Marshall Plan Scholarship Program under grant number 154.265.20.3.2010. The second author has been supported by NSF Grants CNS 07-16638 and 09-04749, and CCF 09-05584.

PDPP'11, July 20–22, 2011, Odense, Denmark.

ing of function symbols; and (c) rewriting modulo axioms such as associativity and/or commutativity and/or identity. Such a generalization is non-trivial. Support for (a) is the least problematic, but support for (b) means that, because of subsort overloading, a function f can never be considered to be defined *once and for all:* it can always be *extended* to bigger sorts. For example, we can first define a $+$ function in a `NAT` module, and then extend its domain of definition in `INT`, `RAT`, and `COMPLEX` modules. Support for (c) is the least obvious, because the notion of "well-founded recursive function definition" does not have a straightforward extension to the modulo case. For example, if f is an associative-commutative (AC) function symbol, a definition of f based on a binary constructor g and a constructor constant a might include an equation $f(g(x,y),a) = g(f(x,y),g(a,a))$, which syntactically satisfies all the expected requirements of well-founded recursive function definitions, yet is non-AC-terminating (cf. Example 5 in Section 3). A related difficulty for axioms like AC is that the usual *syntactic* characterizations of classes of recursive functions (e.g., primitive recursive) are no longer adequate, because of the much greater flexibility in the constructor patterns that can be used. For example, the definition of the cardinality function `card` in the `MSET-NAT` module below could just as well have used an equation `card(MS,MS')= card(MS) + card(MS')` with `MS, MS'` of sort `MSet`, instead of the equation `card(N,MS)= s(0) + card(MS)` with `N` of sort `Nat` below. This work provides a notion of well-founded recursive function definition supporting features (a)–(b)–(c). We show in Section 3.1 that our approach generalizes an already very general notion of many-sorted well-founded recursive function.

To make the approach scalable, the cost of each incremental check should be small. This can be achieved by taking advantage of modular methodologies which ensure that in an immediate submodule inclusion $(\Sigma, E \cup Ax) \subset (\Sigma \cup \Sigma_\Delta, (E \cup E_\Delta) \cup (Ax \cup Ax_\Delta))$, while both modules *can be arbitrarily large*, the *incremental additions* Σ_Δ to the signature, E_Δ to the defining equations, and Ax_Δ to the axioms, are *small*. Such increments being big is a clear sign of bad software engineering practice, since usually a more modular design can be achieved by module refactoring. The incremental proof methods we propose are scalable precisely because they are based on checking the typically small increments $(\Sigma_\Delta, E_\Delta \cup Ax_\Delta)$ and not the, potentially very large, theory $(\Sigma \cup \Sigma_\Delta, (E \cup E_\Delta) \cup (Ax \cup Ax_\Delta))$.

A Running Example. Throughout the paper we use the following running example in Maude. Although small, it illustrates all the key features supported: mutual recursion, order-sortedness, and rewriting modulo axioms.

```
fmod NATURAL is pr TRUTH-VALUE .
 sort Nat .
 op 0 : -> Nat [ctor] .
 op s : Nat -> Nat [ctor] .
 op _+_ : Nat Nat -> Nat [comm id: 0] .
 ops even odd : Nat -> Bool .
 vars N M : Nat .
 eq s(N) + s(M) = s(s(N + M)) .
 eq even(0) = true .
 eq even(s(N)) = odd(N) .
 eq odd(0) = false .
 eq odd(s(N)) = even(N) .
endfm

fmod MSET-NAT is pr NATURAL .
 sort MSet .
 subsort Nat < MSet .
 op _,_ : MSet MSet -> MSet [ctor assoc comm id: null] .
 op null : -> MSet [ctor] .
 op card : MSet -> Nat .
 var MS : MSet .
 var N : Nat .
 eq card(null) = 0 .
 eq card(N,MS)= s(0) + card(MS) .
endfm

fmod LIST-MSET-NAT is pr MSET-NAT .
 sorts List NeList .
 subsorts MSet < NeList < List .
 op nil : -> List [ctor] .
 op U : List -> MSet .
 op _;_ : List List -> List [assoc id: nil] .
 op _;_ : MSet NeList -> NeList [ctor assoc id: nil] .
 var MS : MSet .
 var NL : NeList .
 eq U(nil) = null .
 eq U(MS) = MS .
 eq U(MS ; NL) = MS, U(NL) .
endfm
```

The `NATURAL` module defines the natural numbers with addition and with `even` and `odd` predicates. The `MSET-NAT` module defines multisets of naturals and cardinality of multisets. Finally, the `LIST-MSET-NAT` module forms lists of multisets of numbers and defines a multiset union operator on such lists. Associativity, commutativity and identity axioms are specified with the `assoc`, `comm`, and `id:` attributes. All constructor operators are declared with the `ctor` keyword. As illustrated for `_;_`, an operator can be a constructor for some typing (`NeList`) and a defined symbol for a looser typing: `(0,0) ; nil` and `nil ; nil` are *not* constructor terms.

The paper is organized as follows. Section 2 gives background on order-sorted rewriting. Section 3 introduces well-founded recursive theories. Section 4 describes and justifies the incremental checking methods modulo C and AC axioms. Section 5 extends the approach to other combinations of A, C and I (identity) axioms. Finally, Section 6 discusses related work and presents some conclusions. The proofs of all theorems can be found in the long version of the paper[1].

2. Background on Order-sorted Term Rewriting

We summarize here material from [14, 21] on order-sorted algebra and order-sorted rewriting. For standard notions and notations of ordinary term rewriting we refer to [3, 4]. We start with a partially ordered set $(S, \leq)$ of *sorts*, where $s \leq s'$ is interpreted as *subsort inclusion*. The *connected components* of $(S, \leq)$ are the equivalence classes $[s]$ corresponding to the least equivalence relation $\equiv_\leq$ containing $\leq$. When a connected component $[s]$ has a top element, we will also denote by $[s]$ such a top element. An order-sorted signature $\Sigma = (S, \leq, F)$ consists of a poset of sorts $(S, \leq)$ and a $S^* \times S$-indexed family of sets $F = \{F_{w,s}\}_{(w,s) \in S^* \times S}$, which are *function symbols* with given string of argument sorts and result sort. If $f \in F_{s_1 \ldots s_n, s}$, we declare the function symbol f as $f : s_1 \ldots s_n \longrightarrow s$. Some of these symbols f can be *subsort-overloaded*, i.e., they can have several declarations related in the $\leq$ ordering [14].

Given an S-sorted set $\mathcal{X} = \{\mathcal{X}_s \mid s \in S\}$ of *disjoint* sets of variables and an order-sorted (OS) signature $\Sigma = (S, \leq, F)$, the set $\mathcal{T}(\Sigma, \mathcal{X})_s$ of terms of sort s is the least set such that $\mathcal{X}_s \subseteq \mathcal{T}(\Sigma, \mathcal{X})_s$; if $s' \leq s$, then $\mathcal{T}(\Sigma, \mathcal{X})_{s'} \subseteq \mathcal{T}(\Sigma, \mathcal{X})_s$; and if $f : s_1 \ldots s_n \longrightarrow s$ is a declaration for symbol f and $t_i \in \mathcal{T}(\Sigma, \mathcal{X})_{s_i}$ for $1 \leq i \leq n$, then $f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{X})_s$. The set $\mathcal{T}(\Sigma, \mathcal{X})$ of order-sorted terms is $\mathcal{T}(\Sigma, \mathcal{X}) = \cup_{s \in S} \mathcal{T}(\Sigma, \mathcal{X})_s$. An element of any set $\mathcal{T}(\Sigma, \mathcal{X})_s$ is called a *well-formed* term. A simple syntactic condition on Σ called *preregularity* [14] ensures that each well-formed term t has always a *least-sort* possi-

[1] Available as technical report from `http://www.logic.at/people/schernhammer/` and `TODO`

ble among all sorts in S, which is denoted $ls(t)$. Furthermore, Σ is *monotonic* if for every two declarations $f : s_1 \dots s_n \longrightarrow s$ and $f : s'_1 \dots s'_n \longrightarrow s'$, $s_1 \dots s_n \geqslant s'_1 \dots s'_n$ implies $s > s'$, where $s_1, \dots, s_n \geqslant s'_1, \dots, s'_n$ means $s_i \geq s'_i$ for all $1 \leq i \leq n$ and $s_i > s'_i$ for some $1 \leq i \leq n$. Throughout this paper we assume that all order-sorted signatures are preregular and monotonic. Terms are viewed as labeled trees in the usual way. Positions $p, q, \dots$ are represented by chains of positive natural numbers used to address subterm positions of t. The set of positions of a term t is denoted $\mathcal{P}os(t)$. Positions of non-variable symbols in t are denoted as $\mathcal{P}os_\Sigma(t)$, and $\mathcal{P}os_\mathcal{X}(t)$ are the positions of variables. The subterm at position p of t is denoted as $t|_p$ and $t[u]_p$ is the term t with the subterm at position p replaced by u. We write $t \trianglerighteq u$, read *u is a subterm of t*, if $u = t|_p$ for some $p \in \mathcal{P}os(t)$ and $t \triangleright u$ if $t \trianglerighteq u$ and $t \neq u$.

An order-sorted substitution σ is an S-sorted mapping $\sigma = \{\sigma : \mathcal{X}_s \to \mathcal{T}(\Sigma, \mathcal{X})_s\}_{s \in S}$ from variables to terms. A *specialization* ν is an OS-substitution that maps a variable x of sort s to a variable x' of sort $s' \leq s$. We denote $\mathcal{D}om(\sigma)$ and $\mathcal{R}ng(\sigma)$ the domain and range of a substitution σ. An (order-sorted) rewrite rule is an ordered pair (l, r), written $l \to r$, with $l, r \in \mathcal{T}(\Sigma, \mathcal{X})$, $l \notin \mathcal{X}$, $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$ (and $ls(l) \equiv_\leq ls(r)$ for order-sorted rules). If for all specializations ν, $ls(\nu(l)) \geq ls(\nu(r))$, then we say that the OS-rule $l \to r$ is *sort-decreasing*. A term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ rewrites to u (at position $p \in \mathcal{P}os(t)$ and using the rule $l \to r$), written $t \xrightarrow{p}_{l \to r} s$ (or just $t \to_\mathcal{R} s$ or even $t \to s$ if no confusion arises), if $t|_p = \sigma(l)$ and $s = t[\sigma(r)]_p$, for some OS-substitution σ; if $l \to r$ is *not* sort-decreasing, we also require that $t[\sigma(r)]_p$ is a well-formed term.

An *order-sorted theory* (OS theory) is a triple $\mathcal{E} = (\Sigma, B, R)$ with Σ a preregular order-sorted signature such that each connected component has a top sort, B a set of unconditional Σ-equations, and R a set of unconditional Σ-rules. In this paper B will always be a combination of associativity and/or commutativity and/or identity axioms for some of the operators in Σ. Moreover, associative and commutative operators f are always typed $f\colon s\, s' \longrightarrow s$ for some sorts s, s' where $s' \leq s$. By Σ_{AC} (resp. Σ_C) we denote the subsignature of Σ where all function symbols are associative and commutative but do not have an identity (resp. where all function symbols are commutative but not associative and do not have an identity). Furthermore, we assume[2] that in each equation $u = v$ in B the variables $\{x_1, \dots, x_n\} = \mathcal{V}ar(u) = \mathcal{V}ar(v)$ have *top sorts* $[s_1], \dots, [s_n]$.

Given an OS theory $\mathcal{E}$ as above, $t \to_{R/B} t'$ iff there exist u, v such that $t =_B u$ and $u \to_R v$ and $v =_B t'$. We say that (Σ, B, R) is *B-confluent*, resp. *B-terminating*, if the relation $\to_{R/B}$ is confluent, resp. terminating. By $[w]_B$ we denote that equivalence class of terms that are B-equal to w. We call an order-sorted signature *B-preregular* if the set of sorts $\{s \in S \mid \exists w' \in [w]_B \text{ s.t. } w' \in \mathcal{T}(\Sigma, \mathcal{X})_s\}$ has a least upper bound, denoted $ls[w]_B$ which can be effectively computed.[3] If (Σ, B, R) is B-confluent, B-terminating, B-preregular, and sort-decreasing, then the initial algebra $\mathcal{T}_{\Sigma/R \cup B}$, where the rules R are interpreted as equations, is isomorphic to the canonical term algebra $\mathcal{C}_{\Sigma/R,B}$, whose elements are B-equivalence classes in $\to_{R/B}$-canonical form. An order-sorted subsignature $\Omega \subseteq \Sigma$ with the same poset of sorts as Σ is called a *constructor subsignature* iff for each ground Σ-term t there is a ground Ω-term u such that $t \to^*_{R/B} u$. Terms from $\mathcal{T}(\Omega, \mathcal{X})$ are called Ω-*constructor terms*, or just constructor terms if Ω is clear from the context. We then say that (Σ, B, R) is *sufficiently complete* with respect to Ω. Intuitively this means that the functions defined by the rules R have been fully defined. For instance, the operators declared with the `ctor` attribute in our running example define a constructor subsignature, so that the specification is sufficiently complete. Assuming that (Σ, B, R) is B-confluent, B-terminating, B-preregular, and sort-decreasing, and that Ω is a constructor subsignature, if for any $t \to t'$ in R, whenever t is an Ω-term then t' is also an Ω-term, we are then guaranteed that all the elements in the canonical term algebra $\mathcal{C}_{\Sigma/R,B}$ are B-equivalence classes of ground Ω-terms. If, in addition, any ground Ω-term t is in $\to_{R/B}$-canonical form, then we call Ω a signature of *free constructors modulo B*.

Given an OS theory $\mathcal{E} = (\Sigma, B, R)$, we call an unsorted theory $\mathcal{E}' = (\Sigma', B', R')$ a *sound reflection* of $\mathcal{E}$ if there exists a mapping $\mathcal{M}$ from $\mathcal{T}(\Sigma, \mathcal{X})$ to a set of unsorted terms such that $t \to_{R/B} t' \Rightarrow \mathcal{M}(t) \to^+_{R'/B'} \mathcal{M}(t')$ for all terms $t, t' \in \mathcal{T}(\Sigma, \mathcal{X})$ (in that case we say that $\mathcal{E}'$ is a sound reflection of $\mathcal{E}$ w.r.t. $\mathcal{M}$; cf. [25]). Given a strict order $\succ$ on some domain D, the lexicographic extension of $\succ$ to n-tuples over D is defined as $\langle d_1, \dots, d_n\rangle \succ^{lex} \langle d'_1, \dots, d'_n\rangle$ if there exists an $i \leq n$ such that $d_j = d'_j$ for all $j < i$ and $d_i \succ d'_i$. Moreover, the multiset extension of $\succ$ to multisets over D is defined by $D_1 \succ^{mul} D_2$ if $D_1 \neq D_2$ and for each $d \in D_2 \setminus D_1$ there exists a $d' \in D_1 \setminus D_2$ such that $d' \succ d$.

3. Well-founded Recursive Theories

In this section we introduce the notion of well-founded recursive OS theories modulo axioms. The basic idea is to impose conditions on the equations of such theories, that guarantee finiteness of rewrite derivations. These conditions are based on the notion of *recursive dependency* of function symbols. Intuitively, a function symbol f recursively depends on g if there is a rule $f(t_1, \dots, t_n) \to r$ in the OS theory with $root(r|_p) = g$ for some position p of r.

Definition 1 (recursive dependency)**.** *Assume the axioms B_0 of the theory $\mathcal{E} = (\Sigma, B_0, R)$ are only commutativity and associativity-commutativity axioms.*[4]

Let G be the names of function symbols in Σ. The relation $\blacktriangleright^1_\mathcal{E} \subseteq G \times G$ is defined as $f \blacktriangleright^1_\mathcal{E} g$ whenever, there is a rule $l \to r \in R$ and a position $p \in Pos(r)$ such that $root(l) = f$ and $root(r|_p) = g$. The preorder $\blacktriangleright_\mathcal{E} \subseteq G \times G$ is obtained as the reflexive and transitive closure of $\blacktriangleright^1_\mathcal{E}$.

For order-sorted theories, and in particular in the presence of subsort overloaded function symbols, it is advantageous to distinguish between subsort overloaded variants of function symbols, because by doing so one obtains a more fine-grained notion of recursive dependency. This more fine-grained notion is needed because recursive dependencies exclusively based on the *names* of overloaded function symbols are too coarse to faithfully capture the actual dependencies involved in order-sorted rewriting.

A straightforward approach to achieve this disambiguation of subsort overloaded function symbols is to label them with the sorts of their arguments. This approach was used for instance by Ölveczky et al. ([25][Definitions 2 and 3]) to obtain an unsorted *reflection* of order-sorted rewrite systems. Unfortunately, in the pres-

[2] This assumption makes the technical treatment simpler, but it does not involve loss of generality and does not have to be specified explicitly: we can always assume that a new top sort $[s]$ is added to each connected component so that a binary operator f to which some axiom in B apply is overloaded for the top sort as $f : [s]\ [s] \longrightarrow [s]$. Maude adds these "kind" sorts $[s]$ automatically to any specification.

[3] Maude automatically checks the B-preregularity of a signature Σ for any combination of associativity, commutativity and identity axioms (see [6, Chapter 22.2.5]).

[4] The subscript 0 in B_0 indicates that only C and AC axioms are present. In Section 5 below also other combinations of A, C and I axioms will be considered.

ence of associativity axioms the unsorted rewrite system obtained by this labeling may not reflect the original order-sorted one.

Example 1. *Consider an OS theory $\mathcal{E}$ with sorts $A < B$, a function symbol f which is subsort overloaded with typings $f\colon A\ A \to A$, and $f\colon B\ B \to B$, a unary function symbol s with typing $s\colon B \to A$, and a constant b of sort B. The symbol f is associative and commutative and the rules are*

$$\begin{aligned} f(b,x) &\to f(s(b),s(b)) \\ f(b,s(x)) &\to f(b,x) \end{aligned}$$

where the sort of the variable x is B. By labeling the function symbols according to the sorts of their arguments in the corresponding order-sorted rewrite system, one does not obtain a sound reflection. The labeled versions of the above rules (including specializations) are

$$\begin{aligned} f_{B,B}(b,x) &\to f_{A,A}(s_B(b),s_B(b)) \\ f_{B,A}(b,x) &\to f_{A,A}(s_B(b),s_B(b)) \\ f_{B,A}(b,s_B(x)) &\to f_{B,B}(b,x) \\ f_{B,A}(b,s_A(x)) &\to f_{B,A}(b,x) \end{aligned}$$

In [25], additionally rules that decrease the sort labelings of function symbols are needed. Here, these rules are

$$\begin{aligned} f_{B,B}(x,y) &\to f_{A,B}(x,y) \\ f_{B,B}(x,y) &\to f_{B,A}(x,y) \\ f_{A,B}(x,y) &\to f_{A,A}(x,y) \\ f_{B,A}(x,y) &\to f_{A,A}(x,y) \\ s_B(x) &\to s_A(x) \end{aligned}$$

The symbols $f_{B,B}$, $f_{B,A}$ and $f_{A,A}$ are considered to be associative and commutative. The following cyclic reduction sequence cannot be reflected.

$$\begin{aligned} f(f(b,s(b)),b) &\to f(f(s(b),s(b)),b) =_{AC} \\ &=_{AC} f(f(b,s(b)),s(b)) \to \\ &\to f(f(b,b),s(b)) =_{AC} f(f(b,s(b)),b) \end{aligned}$$

In fact the labeled rewrite system is terminating (which can automatically be proved by AProVE [13]).

The reason for the inability of the labeled rewrite system to simulate correctly the order-sorted rewriting of Example 1 is the complex interaction between sorts and structural axioms. More precisely, in the term $f(f(s(b),s(b)),b)$ the sort of the arguments of the inner f symbol is A. However, in the AC-equal term $f(f(b,s(b)),s(b))$ the two arguments of the inner f symbol have sorts B and A. Hence, there is an increase in the sorts of the arguments caused by the associativity axiom. Note that in the labeled version of the term $f(f(s(b),s(b)),b)$ which is $f_{A,B}(f_{A,A}(s_B(b_B),s_B(b_B)),b_B)$ no associativity equation is applicable since $f_{A,B} \neq f_{A,A}$.

The problem, therefore, is to find a C- and AC-compatible disambiguation scheme on which we can express order-sorted recursive dependencies. The solution to this problem is to label AC function symbols not by pairs of sorts, but by the multisets of sorts of arguments of the flattened versions of the terms in question. Commutative (but not associative) function symbols are labeled by unordered pairs of sorts of their arguments.

Definition 2 ((top) flattening)**.** *Let Σ be an unsorted signature containing free and AC-function symbols and let f be an AC symbol. Then,*

$$flat(t,f) = \begin{cases} x & \text{if } t = x, \text{ a variable} \\ g(t_1,\ldots,t_n) & \text{if } t = g(t_1,\ldots,t_n),\ g \neq f \\ f(T_1 \cup T_2) & \text{if } t = f(t_1,t_2) \end{cases}$$

where

$$T_i = \begin{cases} \{u_1,\ldots,u_m\} & \text{if } flat(t_i,f) = f(u_1,\ldots,u_m) \\ \{flat(t_i,f)\} & \text{otherwise} \end{cases}$$

Definition 3 (labeled signature, lab)**.** *Let $\Sigma = (S,<,F)$ be an order-sorted signature containing AC, C and free function symbols. Its associated unsorted labeled signature Σ^{os} is given by*

$$\begin{aligned} \{f_\Psi \mid\ & f\colon A\,B \to A \in \Sigma_{AC}, \\ & \Psi \text{ a finite multiset of sorts} \leq A\} \cup \\ \{f_{[A',B']} \mid\ & f\colon A\,B \to C \in \Sigma_C, [A',B'] \text{ an unordered} \\ & \text{pair of sorts } A' \leq A, B' \leq B\} \cup \\ \{f_{A'_1,\ldots,A'_n} \mid\ & f\colon A_1 \ldots A_n \to C \in \Sigma \setminus (\Sigma_C \cup \Sigma_{AC}), \\ & A'_i \leq A_i \text{ for all } 1 \leq i \leq n\}. \end{aligned}$$

For a function symbol $f_\mathcal{A}$ of Σ^{os} where $\mathcal{A}$ is a multiset, an unordered pair or a sequence of sorts, we denote by $erase(f_\mathcal{A})$ the unlabeled function symbol f and by $lab(f_\mathcal{A})$ the label $\mathcal{A}$.

Note that Σ^{os} is countably infinite in general if Σ is countable. Next we define a mapping from terms over Σ to terms over the labeled signature Σ^{os}.

Definition 4 (labeling terms)**.** *Let $\Sigma = (S,<,F)$ be an order-sorted signature containing AC, C and free function symbols which is preregular modulo the AC and C axioms. The mapping $\overline{\cdot}\colon \mathcal{T}(\Sigma,V) \to \mathcal{T}(\Sigma^{os},V)$ is defined by*

$$\overline{t} = \begin{cases} x & \text{if } t = x \in V \\ f_{ls(t_1),\ldots,ls(t_n)}(\overline{t_1},\ldots,\overline{t_n}) & \text{if } t = f(t_1,\ldots,t_n), \\ & f \in \Sigma \setminus (\Sigma_C \cup \Sigma_{AC}) \\ f_{[ls(t_1),ls(t_2)]}(\overline{t_1},\overline{t_2}) & \text{if } t = f(t_1,t_2),\ f \in \Sigma_C \\ f_\Psi(\lambda(t_1,f,\Psi),\lambda(t_2,f,\Psi)) & \text{if } t = f(v_1,v_2),\ f \in \Sigma_{AC}, \\ & flat(t,f) = f(u_1,\ldots,u_m), \\ & \Psi = \{ls(u_1),\ldots,ls(u_m)\} \end{cases}$$

where

$$\lambda(u,f,\Psi) = f_\Psi(\lambda(u_1,f,\Psi),\lambda(u_2,f,\Psi))$$

if $u = f(u_1,u_2)$ and $\overline{u}$ otherwise.

Note that, by definition, constants are always *labeled by the empty sequence ϵ. Thus, for notational simplicity we omit the label of constants if no confusion arises. Slightly abusing notation, we denote by $erase$ the inverse mapping of $\overline{\cdot}$, which erases the labels of function symbols and is defined for terms $f(t_1,\ldots,t_n) \in \mathcal{T}(\Sigma^{os},V)$ as $erase(f)(erase(t_1),\ldots,erase(t_n))$.*

Example 2. *Consider the signature of $\mathcal{E}$ in Example 1. The labeled term $\overline{f(f(s(b),s(b)),b)}$ is*

$$f_{\{A,A,B\}}(f_{\{A,A,B\}}(s_B(b),s_B(b)),b).$$

By labeling terms in equations of an OS theory, we obtain a theory transformation that maps OS theories modulo axioms to unsorted theories modulo axioms.

Definition 5 (labeled theory)**.** *Let $\mathcal{E} = (\Sigma,B_0,R)$ be an OS theory with axioms B_0 including only C and AC axioms. By $\overline{\mathcal{E}}$ we denote the unsorted theory $(\Sigma^{os},\overline{B_0},\overline{R})$ where*

$$\begin{aligned} \overline{B_0} &= \{\overline{l\theta} = \overline{r\theta} \mid l = r \in B_0,\ \theta \text{ a sort specialization}\} \\ \overline{R} &= \{\overline{l\theta} \to \overline{r\theta} \mid l \to r \in R,\ \theta \text{ a sort specialization}\} \end{aligned}$$

Example 3. *Consider the module* `LIST-MSET-NAT` *of our running example of Section 1 and the equation*

```
(MS ; NL) ; L = MS ; (NL ; L)
```

that is added by the theory transformation of Section 5 below, thus eliminating the associativity axiom for ";". The sorts of the variables are $MS\colon MSet, NL\colon NeList$ and $L\colon List$. Hence, the labeled version of this equation (considering the identity specialization) is

$$(MS\ ;_{MSet,NeList}\ NL)\ ;_{NeList,List}\ L = MS\ ;_{MSet,List}\ (NL\ ;_{NeList,List}\ L).$$

Thus, the various occurrences of the symbol ";" in the equation are explicitly disambiguated. Moreover, there are 24 specializations of this equation (including the identity), because the variables can be specialized in the following way:

$$\begin{array}{rcl} MS & to & MSet, Nat \\ NL & to & NeList, MSet, Nat \\ L & to & List, NeList, MSet, Nat \end{array}$$

Hence, according to Definition 5, our equation is transformed into 24 labeled equations given by

$$(MS\ ;_{X,Y}\ NL)\ ;_{Y,Z}\ L = MS\ ;_{X,Z}\ (NL\ ;_{Y,Z}\ L)$$
where $X \in \{MSet, Nat\}, Y \in \{NeList, MSet, Nat\}$, $Z \in \{List, NeList, MSet, Nat\}$.

As shown by Example 4 below, the theory transformation of Definition 5 is *not* a sound reflection (w.r.t. $\bar{\cdot}$). However, it can be extended to a theory transformation that is a sound reflection (w.r.t. $\bar{\cdot}$). This is detailed in the long version of this paper[5]. Nevertheless, in this short version of the paper we use labeled versions of rewrite rules exclusively to derive a "sort-aware" recursive dependency relation (cf. Definition 8 below). For this purpose, it suffices to use the simpler theory transformation of Definition 5. Hence, for the sake of simplicity, we use only this transformation in the rest of this section.

Example 4. *Consider the theory $\mathcal{E} = (\Sigma, B_0, R)$ of Example 1. The rules of $\overline{\mathcal{E}} = (\Sigma^{os}, \overline{B_0}, \overline{R})$ are*

$$\begin{array}{rcl} f_{\{B,B\}}(b,x) & \to & f_{\{A,A\}}(s_B(b), s_B(b)) \\ f_{\{A,B\}}(b, s_B(x)) & \to & f_{\{B,B\}}(b,x) \\ f_{\{A,B\}}(b,x) & \to & f_{\{A,A\}}(s_B(b), s_B(b)) \\ f_{\{A,B\}}(b, s_A(x)) & \to & f_{\{A,B\}}(b,x) \end{array}$$

In the unlabeled system, we have $s = f(b, f(b,b)) \to_R f(s(b), s(b)) = t$, but after labeling the corresponding reduction is impossible: $\overline{s} = f_{\{B,B,B\}}(b_\epsilon, f_{\{B,B,B\}}(b_\epsilon, b_\epsilon)) \not\to_{\overline{R}} f_{\{A,A\}}(s_B(b), s_B(b)) = \overline{t}$.

Based on the notion of recursive dependency and the labeling of function symbols, we now define well-founded recursive OS theories modulo axioms. These well-founded recursive theories are guaranteed to be terminating and properties like confluence and sufficient completeness can be verified incrementally. Left-hand sides of equations in well-founded OS theories are linear *patterns*, or linear constructor terms with constructor right-hand sides in case constructors are not B_0-free ($\Omega \subseteq \Sigma$ is B_0-free iff for each specialization $l\nu \to r\nu$ with $l \to r \in R$, $l\nu$ is not an Ω-term).

Definition 6 (pattern)**.** *Let $\mathcal{E} = (\Sigma, B_0, E)$ be an OS theory where $\Sigma = \mathcal{D} \uplus \Omega$ is partitioned into defined function symbols and constructors.*[6] *A term t is a* pattern *if it is linear and every proper subterm is from $\mathcal{T}(\Omega, V)$.*

In order to obtain termination of well-founded recursive theories, arguments of functions called recursively by other mutually recursively dependent functions have to decrease. For example, when considering a rewrite rule $f(s_1, s_2, s_3) \to C[f(t_1, t_2, t_3)]$ for some context C, we demand that the tuple $\langle s_1, s_2, s_3\rangle$ resp. the multiset $\{s_1, s_2, s_3\}$ is greater than $\langle t_1, t_2, t_3\rangle$ resp. $\{t_1, t_2, t_3\}$ w.r.t. some extension of the subterm ordering to tuples or multisets that preserves well-foundedness. Whether arguments of non-commutative functions are compared as tuples (thus using a lexicographic ordering lex) or multisets (thus using a multiset ordering mul or a more specialized version tup for AC functions) is determined by a status function. This provides a maximum of flexibility, since arguments of different functions can be compared in different ways (unless the functions are mutually recursive). The idea of recursive calls to functions with smaller collections of arguments is formalized by the notion of *argument decreasing rules*, where all recursive function calls are to functions with smaller argument collections.

Definition 7 (decreasing rule)**.** *Let $\mathcal{E} = (\Sigma, B_0, R)$ be an OS theory where B_0 consists exclusively of AC and C axioms and let $stat\colon \Sigma \to \{lex, mul\}$ be a status function on Σ. Moreover, let $l \to r$ be a rule of R and g a function symbol such that $root(l)$ and g are either both AC or none of them is AC. We say that $l \to r$ is* g-argument decreasing *if $stat(root(l)) = stat(g)$ and for each subterm $r|_p$ of r with $root(r|_p) = g$ there are terms $l' =_{B_0} l$ and $w' =_{B_0} r|_p$ such that*

$$\{l''|_1, \ldots, l''|_{ar(root(l''))}\} \rhd^{tup} \{w''|_1, \ldots, w''|_{ar(root(w''))}\}$$

if $root(l'), root(w') \in \Sigma_{AC}$ where $l'' = flat(l', root(l'))$ and $w'' = flat(w', root(w'))$[7]*; and*

$$\{l'|_1, \ldots, l'|_{ar(root(l'))}\} \rhd^{mul} \{w'|_1, \ldots, w'|_{ar(root(w'))}\}$$

if $root(l'), root(w') \notin \Sigma_{AC}$ and $stat(root(l)) = mul$; and

$$\langle l'|_1, \ldots, l'|_{ar(root(l'))}\rangle \rhd^{lex} \langle w'|_1, \ldots, w'|_{ar(root(w'))}\rangle$$

if $root(l'), root(w') \notin \Sigma_{AC}$ and $stat(root(l)) = lex$.

Note that the status function does not assign the tup extension of the subterm ordering to non-AC function symbols, as the mul extension is more general and thus subsumes the use of the tup extension. For AC function symbols it is crucial to compare multisets of arguments by $\rhd^{tup}$ instead of $\rhd^{mul}$ in order to obtain a well-founded decrease of argument multisets. Example 5 shows that theories may be non-terminating even if multisets of mutually recursive functions decrease w.r.t $\rhd^{mul}$.

Example 5. *Consider the unsorted theory $\mathcal{E}$ (already used in Section 1) using a defined binary AC-function symbol f, a binary constructor g and a constructor constant a. The single rule is $f(g(x,y), a) \to g(f(x,y), g(a,a))$.*

We have $f \blacktriangleright_\mathcal{E} g$ but not $g \blacktriangleright_\mathcal{E} f$. Hence, we compare the arguments of $flat(f(g(x,y),a), f) = f(g(x,y),a)$ and $flat(f(x,y), f) = f(x,y)$. We have $\{g(x,y), a\} \rhd^{mul} \{x, y\}$.

[5] Available as technical report from `http://www.logic.at/people/schernhammer/` and `TODO`

[6] But note that we can have $f\colon s_1, \ldots, s_n \to s \in \mathcal{D}$ and another $f\colon s'_1, \ldots, s'_n \to s' \in \Omega$, as illustrated for $f = \_;\_$ by our running example.

[7] $A \rhd^{tup} B$ (where $\rhd$ is the proper subterm relation of terms) for multisets A and B of terms means that $A = A' \cup C$, $B = B' \cup C$, $A' \neq \emptyset$ and there is a (possibly partial) surjective mapping $\phi\colon A' \to B'$, such that $\phi(a) = b$ implies $a \rhd b$.

Indeed, $\mathcal{E}$ is not AC-terminating:

$$\begin{aligned} f(g(f(a,a),g(a,a)),a) &\to \\ \to\ g(f(f(a,a),g(a,a)),g(a,a)) &=_{AC} \\ =_{AC}\ g(f(f(g(a,a),a),a),g(a,a)) &\to \\ \to\ g(\underline{f(g(f(a,a),g(a,a)),a)},g(a,a)) & \end{aligned}$$

Note however that $\{g(x,y),a\} \not>^{tup} \{x,y\}$.

Now we are ready to define well-founded recursive OS theories modulo axioms. Ultimately, our goal is to show that well-founded recursive OS theories are compatible with a recursive path ordering that is compatible with C and AC axioms (i.e. an $ACRPO$). Due of the presence of rules like $c(c(x,y),z) \to c(x,c(y,z))$ (cf. e.g. Example 9 and Section 5 below) and commutative function symbols, it is crucial to compare arguments of some functions lexicographically and others by multiset orders. Moreover, for rules involving subsort-overloaded AC function symbols, sorts may or may not be crucial and it may even be advantageous to ignore sorts of certain function symbols (cf. Example 6 below). Hence, the notion of well-founded recursive OS theory modulo axioms is *parameterized* by two status functions $stat$ and $stat_{ac}$. Note, however, that this does not compromise the syntactic and easy-to-check character of well-founded recursion, since the possible choices for these status functions are finite for each finite OS theory. However, finding working status functions may be computationally hard. To solve this problem we show in Section 4 below that these status functions can be computed incrementally when checking hierarchies of order-sorted theories for being well-founded recursive. Hence, provided that the theory extensions in such a hierarchy are small, the task of choosing suitable status functions is feasible.

Definition 8 (well-founded theories)**.** *Let $\mathcal{E} = (\Sigma, B_0, R)$ be an OS theory with constructors $\Omega \subseteq \Sigma$ where the structural axioms B_0 are either AC or C axioms. Let $stat\colon \Sigma \to \{lex, mul\}$ be a status function where $stat(f) = mul$ for all $f \in \Sigma_C \cup \Sigma_{AC}$ and where $f \blacktriangleright_\mathcal{E} g, g \blacktriangleright_\mathcal{E} f$ implies $stat(f) = stat(g)$ and. Let $stat_{ac}\colon \Sigma_{AC} \to \{s, us\}$[8] be a status function where $f, g \in \Sigma_{AC}$ and $f \blacktriangleright_\mathcal{E} g, g \blacktriangleright_\mathcal{E} f$ implies $stat_{ac}(f) = stat_{ac}(g)$.*

$\mathcal{E}$ is well-founded recursive iff (i) $g \blacktriangleright_\mathcal{E} h$ and $h \blacktriangleright_\mathcal{E} g$ ($h, g \in \Sigma$) implies that h and g are either both AC symbols or both non-AC-symbols, and (ii) there are status function $stat$ and $stat_{ac}$ such that for each rule $l \to r$ and each specialization θ the following properties hold:

1. *Either l is a linear constructor term, or if not then l is a pattern in case $root(l) \in \Sigma \setminus \Sigma_{AC}$ and $flat(l, root(l))$ is a pattern in case $root(l) \in \Sigma_{AC}$.*
2. *If l is a constructor term, then so is r.*
3. *For every (not necessarily proper) subterm $r|_p$ of r,*

   $$root(\overline{r\theta}|_p) \blacktriangleright_{\overline{\mathcal{E}}} root(\overline{l\theta})$$

   (resp. $root(r|_p) \blacktriangleright_\mathcal{E} root(l) \in \Sigma_{AC}$ and $stat_{ac} = us$) implies that

   $$\overline{l\theta} \to \overline{r\theta} \text{ is } root(\overline{r\theta}|_p) \text{ argument decreasing (w.r.t. } stat)$$

   (resp. that $l \to r$ is $root(r|_p)$ argument decreasing).
4. *Assume $root(l) = root(r|_p)$ for some $p \in Pos(r)$, $root(r|_p) \blacktriangleright_\mathcal{E} root(l)$ and $stat_{ac}(root(l)) = s$ and consider the multiset S of arguments of $root(\overline{l\theta})$ in the term $\overline{l\theta}$ as well as the multiset T of arguments of $root(\overline{r|_p\theta})$ in the term $\overline{r|_p\theta}$. For every variable $x \in T \setminus S$, there exists a term $s \in S \setminus T$, such that $ls(s) > ls(x)$. Moreover, $lab(root(\overline{l\theta})) \geq^{mul} lab(root(\overline{r|_p\theta}))$.*
5. *If l is a constructor term and $root(l)$ is associative and commutative, then*

   $$root(flat(l, root(l))|_p) \not\blacktriangleright_\mathcal{E} root(l)$$

   for all positions $p \in Pos_\Sigma(flat(l, root(l)))$ with $p > \epsilon$.

The status function $stat$ in Definition 8 determines whether arguments of function symbols are compared lexicographically or by multiset comparison. Mutually recursive function symbols must have the same status. Moreover, arguments of commutative function symbols may only be compared by multiset orders. Hence, the problem of finding suitable statuses for non-commutative functions is very similar to the problem of finding suitable statuses for functions when checking TRSs for RPOS compatibility for which efficient methods exist (cf. e.g. [28]). These methods can be used to determine the status function when checking theories for being well-founded recursive. The other status function $stat_{ac}$ determines whether sorts are taken into account when comparing arguments of AC-function symbols. The reason why we make this distinction is that in the presence of AC function symbols it is not always desirable to take sorts into account, because the labels of AC-function symbols appearing in equations may change through instantiations.

Example 6. *Consider an OS theory containing two sorts A, B with $A < B$, an AC function symbol $g\colon B, B \to B$, two unary function $h\colon B \to B$ and $t\colon B \to A$ and a constant $a : B$. Consider a rule*

$$g(h(x), h(y)) \to g(x, y).$$

We have

$$root(\overline{g(h(x), h(y))}) = g_{\{B,B\}}$$

and indeed $root(\overline{g(h(x), h(y))\sigma}) = g_{\{B,B\}}$ for every substitution σ. On the other hand, e.g. $root(\overline{g(x,y)\sigma}) = g_{\{B,B,B\}}$ if $x\sigma = g(a,a), y\sigma = y$. Hence, when instantiating the rule, there may be an increase in the multiset of sorts of the root symbol of the right-hand side compared to that of the root symbol of the left-hand side of the rule. In this case it is preferable to consider labeled occurrences of g as equal, since there is a decrease in the arguments of the recursive function call. We would have $stat_{ac}(g) = us$ in this case.

On the other hand, consider a rule

$$g(a, a) \to g(h(a), h(a)).$$

We have

$$\begin{aligned} root(\overline{g(a,a)}) &= g_{\{B,B\}} \\ root(\overline{g(h(a), h(a))}) &= g_{\{A,A\}}. \end{aligned}$$

Thus, in order to orient this rule, e.g. by an $(AC)RPO$, it is preferable to consider the symbols $g_{\{B,B\}}$ and $g_{\{A,A\}}$ as different ones, so that $g_{\{B,B\}}$ can be larger in the precedence of function symbols than $g_{\{A,A\}}$. We would have $stat_{ac}(g) = s$ in this case.

The concrete value of $stat_{ac}$ for AC functions can be determined by checking a theory for possible increases in the multisets of sorts (w.r.t. the multiset extension of the subsort ordering) of the arguments in recursive calls to other (or the same) AC functions. If there are no such increases $stat_{ac}$ of the corresponding function should be set to s, otherwise it should be set to us.

Example 7. *Consider the functional Maude module* `NATURAL` *of the running example of Section 1. It contains an identity axiom so*

[8] The function $stat_{ac}$ determines for an AC function symbol f whether sorts are taken into account (s for sorted) or not (us for unsorted) when comparing labeled versions $f_\Psi, f_{\Psi'}$ of this function in the $ACRPO$ we are going to use to prove termination of well-founded recursive theories (cf. Theorem 1 below).

it is outside the scope of well-founded recursive theories. However, by the semantics-preserving theory transformation described in Section 5 below, we obtain the following module which, considered as an OS theory modulo C, is sort-decreasing and well-founded recursive.

```
fmod TR-NATURAL is pr TRUTH-VALUE .
 sort Nat .
 op 0 : -> Nat [ctor] .
 op s : Nat -> Nat [ctor] .
 op _+_ : Nat Nat -> Nat [comm] .
 ops even odd : Nat -> Bool .
 vars N M : Nat .
 eq N + 0 = N .
 eq s(N) + s(M) = s(s(N + M)) .
 eq even(0) = true .
 eq odd(0) = false .
 eq odd(s(N)) = even(N) .
 eq even(s(N)) = odd(N) .
endfm
```

Note that, since there is only one sort in this module, there are no non-trivial sort specializations and thus the equations of the labeled theory are identical with those of the unlabeled one (modulo names of function symbols; cf. Definition 5). Moreover, there are no AC axioms. Hence, the status function $stat_{ac}$ is irrelevant. Finally, the choice of the status function $stat$ is completely arbitrary since the module is well-founded recursive w.r.t. every choice of the status function.

As for $stat$, mutually recursive AC function symbols have to agree on $stat_{ac}$ in well-founded recursive OS theories. In the presence of AC function symbols $f \in \Sigma$ with $stat_{ac}(f) = s$ in a well-founded recursive OS theory $\mathcal{E}$, two additional complications, compared to non-AC function symbols or those with a $stat_{ac}$ of us, may arise, but, as we explain below, these two potential complications do not cause any problems.

First, since Σ^{os} is infinite, there might be infinite decreasing $\blacktriangleright_{\overline{\mathcal{E}}}$ chains that are not looping. However, by Item (4) of Definition 8 we have $\Psi >^{mul} \Psi'$ whenever, $f_\Psi \blacktriangleright_{\overline{\mathcal{E}}} g_{\Psi'}$ ($f_\Psi \neq g_{\Psi'}$ and $f, g \in \Sigma_{AC}$) where $<$ is the (well-founded) subsort ordering. Hence, there are no infinite non-looping $\blacktriangleright_{\overline{\mathcal{E}}}$ chains.

The second potential complication is that the labeling is not-stable under substitutions as illustrated by Example 6. Item (4) of Definition 8 ensures that this stability is restored by ensuring that the sort of every variable occurring directly under an AC function symbols in the (subterm of the) right-hand side in question is dominated by a larger sort in the left-hand side.

The key result of this section is that well-founded recursive sort-decreasing OS theories modulo axioms are terminating. Therefore, our notion of well-founded recursive OS-theories provides: (i) a new formal definition that extends to the order-sorted and modulo C and AC cases the intuitive notion of "specification of a set of well-founded recursive functions"; (ii) a machine-checkable way of ascertaining whether a specification is indeed well-founded recursive; and (iii) a proof that such specifications are always terminating. Furthermore, as shown in Section 4, the checking that a specification is well-founded recursive can be made in a modular and *incremental* way. In practice, of course, we will want our well-founded recursive specifications to be also *confluent*, and *sufficiently complete* with respect to their constructors. These extra properties can also be checked incrementally, as explained in Section 4.

Theorem 1. *Let $\mathcal{E} = (\Sigma, B_0, R)$ be a sort-decreasing well-founded recursive OS theory where the structural axioms B_0 are either AC or C axioms. Then $\mathcal{E}$ is B_0-terminating.*

Note that the sort-decreasingness requirement is essential in Theorem 1, as shown by the following example.

Example 8. *Consider the following OS theory $\mathcal{E}$ without structural axioms. We have sorts s_1 and s_2 where $s_1 < s_2$. Moreover, there is a unary function symbol f typed $f\colon s_2 \to s_2$, another unary function symbol g typed $g\colon s_2 \to s_1$ and a constant a of sort s_2. The rules are*

$$\begin{array}{rcl} f(a) & \to & f(g(a)), \\ g(x) & \to & x \end{array}$$

where x is a variable of sort s_2. This theory is well-founded recursive. For the problematic first rule we have

$$\overline{f(a)} = f_{s_2}(a_\epsilon)$$

and

$$\overline{f(g(a))} = f_{s_1}(g_{s_2}(a_\epsilon)).$$

Moreover, $f_{s_1} \not\blacktriangleright_{\overline{\mathcal{E}}} f_{s_2}$ and $g_{s_2} \not\blacktriangleright_{\overline{\mathcal{E}}} f_{s_2}$.

However the theory is non-terminating as is witnessed by the cyclic reduction sequence

$$f(a) \to f(g(a)) \to f(a).$$

The problem here is that $\mathcal{E}$ is not sort-decreasing, since $ls(g(x)) = s_1 \not\geq s_2 = ls(x)$ for the second rule if x is of sort s_2.

3.1 Many-Sorted Well-Founded Functions as a Special Case of Well-Founded Theories

To further explain the generality of our notion of well-founded recursive theories we show in detail how it captures as a special case a very general notion of well-founded recursive definition in the many-sorted case without axioms. In a sense this is the most general comparison we can make with previous notions, since to the best of our knowledge the notion has not been previously studied in the order-sorted and modulo cases.

To simplify the exposition we focus on the case of recursive definitions without mutual recursion. It is well-known that by adding extra data constructors, such as product types, several mutually recursive functions can be expressed as a single function.

Definition 9. *Let Ω be a many sorted signature of constructors. A* well-founded recursive tower *is a sequence*

$$(f_1\colon s_1^1 \dots s_{n_1}^1 \to s_1, R_{f_1}) \dots (f_m\colon s_1^m \dots s_{n_m}^m \to s_m, R_{f_m})$$

consisting of fresh function symbols $f_1, \dots, f_m$ not in Ω and of sets of rules R_{f_i}, $1 \leq i \leq m$, defining each f_i such that the rules in R_{f_i} are of the form

$$f_i(t_1, \dots, t_{n_i}) \to C[f_i(u_1^1, \dots, u_{n_i}^1) \dots f_i(u_1^k, \dots, u_{n_i}^k)]$$

with $t_1, \dots, t_{n_i}$ Ω-terms and where:

(i) $f_i(t_1, \dots, t_{n_i})$ is linear and the right-hand side of the rule involves only variables occurring in $f(t_1, \dots, t_{n_i})$.
(ii) The context C is a term in the signature $\Omega \cup \{f_1, \dots, f_{i-1}\}$.
(iii) $k \geq 0$, and for each $1 \leq j \leq k$ the set $\{1, \dots, n_i\}$ can be split into two disjoint subsets $A \uplus B = \{1, \dots, n_i\}$ such that $1 \in A$, and
  (1) for each $a \in A$, $t_a \trianglerighteq u_a^j$;
  (2) for each $b \in B$, either $u_b^j \in \mathcal{T}(\Omega \cup \{f_1, \dots, f_{i-1}\}, \mathcal{X})$ or $u_b^j = f_i(v_1^b, \dots, v_{n_i}^b)$ with $(t_1, \dots, t_{n_i}) \rhd (v_1^b, \dots v_{n_i}^b)$; and
  (3) there is an $a \in A$ with $a < min(B)$ such that $(t_1, \dots, t_a) \rhd (u_1^j, \dots, u_a^j)$

where by definition $(t_1, \dots, t_l) \rhd (t'_1, \dots, t'_l)$ iff for each $1 \leq i \leq l$, $t_i \trianglerighteq t'_i$, and there is a $j \in \{1, \dots, l\}$ such that $t_j \rhd t'_j$.

Note that this definition includes as a special case all primitive recursive functions, where the terms $f_i(u_1^j, \ldots, u_{n_i}^j)$ are such that $(t_1, \ldots, t_{n_i}) \rhd (u_1^j, \ldots, u_{n_i}^j)$. Note also that the equations for Ackerman's function in the introduction are a special instance of the above definition. In practice, two more conditions are required of such recursive towers:

(1) *disjoint patterns*, that is, if $f_i(t_1, \ldots, t_{n_i})$ and $f_i(t'_1, \ldots, t'_{n_i})$ are two different left-hand sides in R_{f_i}, which we may assume have distinct variables, then the patterns do not unify.

(2) *sufficient completeness*, that is, the collection of patterns

$$\{(t_1, \ldots, t_{n_i}) \mid \exists t' s.t.\ f_i(t_1, \ldots, t_{n_i}) = t' \in R_{f_i}\}$$

cover the product sort $s_1^i \times \ldots \times s_{n_i}^i$, in the sense that any ground term in that product is an instance of one of the patterns.

Those are precisely the conditions of confluence and sufficient completeness that we show how to check incrementally in Section 4. The main result is now:

Theorem 2. *For any well-founded recursive tower as in Definition 9, the equational theory $(\Omega \cup \{f_1, \ldots, f_m\}, R_{f_1} \cup \ldots \cup R_{f_m})$ is a well-founded recursive many-sorted theory.*

4. Verifying Properties of OS Theories Incrementally

For well-founded recursive OS theories modulo axioms we can check important properties like termination, confluence, sort-decreasingness and sufficient completeness incrementally in the presence of theory hierarchies that satisfy reasonable conditions. These conditions are formalized in the notion of *fair extension*. The basic idea of fair extensions is that extending modules do not interfere with their base modules, i.e., they do not introduce new constructors of sorts of the base module and they do not redefine existing functions.

Definition 10 (fair extension)**.** *Assume that $\mathcal{E}_1 = (\Sigma_1, B_0^1, R_1)$ and $\mathcal{E}_2 = (\Sigma_1 \cup \Sigma_2, B_0^1 \cup B_0^2, R_1 \cup R_2)$ are OS theories where the B_0^is are C or AC axioms for $i \in \{1, 2\}$. Σ_1 and $\Sigma_1 \cup \Sigma_2$ are order-sorted signatures. We write $\Sigma_1 = (S_1, <_1, F_1)$ and $\Sigma_1 \cup \Sigma_2 = (S_1 \cup S_2, <_1 \cup <_2, F_1 \cup F_2)$. Furthermore, F_i is divided into constructors Ω_i and defined function symbols $\mathcal{D}_i$ for both $i \in \{1, 2\}$. $\mathcal{E}_2$ is a* fair extension *of $\mathcal{E}_1$ iff:*

1. *every function symbol f from Σ_1 is AC (resp. C) in $\mathcal{E}_2$ iff it is AC (resp. C) in $\mathcal{E}_1$;*
2. *Σ_2 does not introduce subsorts of sorts of Σ_1, i.e. $s \in S_1 \wedge s' <_1 \cup <_2 s$ for some $s' \in S_1 \cup S_2$ implies $s' <_1 s$;*
3. *Σ_2 does not contain new constructors of some sort of S_1, i.e. $f\colon s_1, \ldots, s_n \to s \in \Omega_2$ implies $s \notin S_1$;*
4. *for every rule $l \to r \in R_2$ and every function symbol $f\colon s_1, \ldots, s_n \to s \in F_1$, l and $f(x_{s_1}^1, \ldots, x_{s_n}^n)$ do not unify in an order-sorted fashion modulo axioms (where x_s denotes a variable of sort s).*
5. *if f is a defined AC symbol in $\mathcal{E}_1$ and $f \blacktriangleright_{\varepsilon_1} g$, $g \not\blacktriangleright_{\varepsilon_1} f$, then $g \not\blacktriangleright_{\varepsilon_2} f$.*
6. *if $c \in \Omega_1$ and there is a rule $l \to r$ from R_1 such that $root(l) = c$, then l does not overlap (order-sorted modulo axioms) with the left-hand side of any rule $l' \to r'$ of R_2 in case $root(l')$ is a defined symbol in $\Sigma_1 \cup \Sigma_2$, and c does not occur below the root of l' in case $root(l')$ is an associative-commutative constructor.*

The first item of Definition 10 ensures that overloaded function symbols have the same set of attached axioms. Items 2 – 4 ensure that no new subsorts and constructors of sorts of the base module are introduced and no functions of the base module are redefined. Item 5 makes sure that no additional mutual recursive dependency of AC symbols is introduced by the extending module, and item 6 is needed to prevent overlaps of rules from R_1 that have constructor terms as left-hand sides with rules from R_2.

In the rest of this section we denote by $\mathcal{E}_1 = (\Sigma_1, B_0^1, R_1)$ an OS theory modulo axioms B_0^1 and by $\mathcal{E}_2 = (\Sigma_1 \cup \Sigma_2, B_0^1 \cup B_0^2, R_1 \cup R_2)$ a fair extension of $\mathcal{E}_1$. By $\mathcal{E}_2'$ we denote the OS theory $(\Sigma_1 \cup \Sigma_2, B_0^1 \cup B_0^2, R_2)$. First, we show modularity of sort-decreasingness. Then we show that the property of being well-founded recursive itself is modular, provided that the base and extending theory agree on the status functions.

Theorem 3 (modularity of sort-decreasingness)**.** *If $\mathcal{E}_1$ and $\mathcal{E}_2'$ are both sort-decreasing, then so is $\mathcal{E}_2$.*

Theorem 4 (modularity of well-founded recursion)**.** *If $\mathcal{E}_1$ and $\mathcal{E}_2'$ are well-founded recursive w.r.t. to compatible functions $stat^1, stat_{ac}^1$ and $stat^2, stat_{ac}^2$, then so is $\mathcal{E}_2$*[9]*.*

Note that we require that the status functions of the base theory and the extending theory are compatible. In a naive mechanization of incremental checks for well-founded recursiveness this could necessitate backtracking, i.e., modifying the status function of a base module depending on an extending theory. To avoid this backtracking, we propose to compute the status functions incrementally in a "by need" fashion. This means that a specific status is assigned to a function symbol (resp. an AC symbol) only if this status is crucial for the theory in question to be well-founded recursive. Otherwise, the status is left open, so that it can be set later when incrementally checking an extending theory for well-founded recursiveness. For example, consider the theory of Example 7. It is well-founded recursive w.r.t. every status function $stat$. Hence, the status of functions can later be set arbitrarily when checking an extending module. A fully general implementation of this idea could, for example, compute a set of status functions for which a module is well-founded recursive. Then when checking an extending module for well-founded recursiveness one could choose suitable status functions from this set of possible ones. Next we show that confluence is modular for fair extensions of well-founded recursive theories.

Theorem 5 (modularity of confluence)**.** *Assume $\mathcal{E}_1$ and $\mathcal{E}_2'$ are well-founded recursive w.r.t. to compatible functions $stat^1, stat_{ac}^1$ and $stat^2, stat_{ac}^2$. If $\mathcal{E}_1$ and $\mathcal{E}_2'$ are confluent then so is $\mathcal{E}_2$.*

Note that for sufficient completeness the adequate notion of modular check consists of checking the property only for *new* defined function symbols.

Theorem 6 (modularity of sufficient completeness)**.** *Assume $\mathcal{E}_1$ and $\mathcal{E}_2'$ are well-founded recursive w.r.t. to compatible functions $stat^1, stat_{ac}^1$ and $stat^2, stat_{ac}^2$. If $\mathcal{E}_1$ is sufficiently complete and for every function $f\colon s_1, \ldots, s_n \to s \in \mathcal{D}_2 \setminus \mathcal{D}_1$ and every ground substitution σ mapping variables to irreducible constructor terms, $f(x_{s_1}^1, \ldots, x_{s_n}^n)\sigma$ is either $\mathcal{E}_2$-reducible or a constructor term (x_s denotes a variable of sort s), then $\mathcal{E}_2$ is sufficiently complete.*

This way of incrementally checking sufficient completeness is compatible with existing automated methods to check the property. Roughly, the idea of these methods is to check whether ground terms rooted by a defined function symbol and having only constructor terms as proper subterms are either reducible, or constructor terms (which is possible as the root symbol might be subsort overloaded). This is done by describing the respective languages of terms by (propositional) tree automata and then reducing the prob-

[9] Compatibility of two functions f_1 and f_2 here means that $f_1(a) = f_2(a)$ whenever $a \in Dom(f_1) \cap Dom(f_2)$.

lem to an emptiness problem for tree automata (we refer to [15] and [16] for more details). The method is suitable for incremental checks following Theorem 6, since it can easily be adapted to consider only terms rooted by defined function symbols of the extending theory instead of all.

Corollary 1. *Assume $\mathcal{E}_1$ and $\mathcal{E}_2'$ are well-founded recursive w.r.t. to functions $stat^1, stat^1_{ac}$ and $stat^2, stat^2_{ac}$ that are compatible. If $\mathcal{E}_1$ and $\mathcal{E}_2'$ are sort-decreasing and confluent and moreover, $\mathcal{E}_1$ is sufficiently complete and for every function $f\colon s_1, \ldots, s_n \to s \in \mathcal{D}_2 \setminus \mathcal{D}_1$ and every irreducible ground substitution σ (that maps variables only to constructor terms) $f(x^1_{s_1}, \ldots, x^n_{s_n})\sigma$ is $\mathcal{E}_2$-reducible (or a constructor term), then $\mathcal{E}_2$ is sort-decreasing, well-founded recursive (thus terminating), confluent and sufficiently complete.*

Example 9. *Consider the running example of Section 1. In order to apply our methods to the modules of this example, the identity axioms and those axioms specifying associativity for a non-commutative function symbol have to be eliminated. Indeed, we can eliminate these problematic axioms by the theory transformation presented in Section 5. This transformation yields the module of Example 7 for the module* NATURAL *and the following two transformed theories for the modules* MSET-NAT *and* LIST-MSAT-NAT*.*

```
fmod TR-MSET-NAT is pr TR-NATURAL .
 sort MSet .
 subsort Nat < MSet .
 op _,_ : MSet MSet -> MSet [ctor assoc comm] .
 op null : -> MSet [ctor] .
 op card : MSet -> Nat .
 var MS : MSet .
 var N : Nat .
 var X : [MSet] .
 eq X , null = X .
 eq card(null) = 0 .
 eq card(N) = s(0) + card(null) .
 eq card(N,MS) = s(0) + card(MS) .
endfm

fmod TR-LIST-MSET-NAT is pr TR-MSET-NAT .
 sorts List NeList .
 subsorts MSet < NeList < List .
 op nil : -> List [ctor] .
 op _;_ : List List -> List .
 op _;_ : MSet NeList -> NeList [ctor] .
 op U : List -> MSet .
 var MS : MSet .
 var NL : NeList .
 var L : List.
 var Y : [List] .
 eq Y ; nil = Y .
 eq nil ; Y = Y .
 eq (MS ; NL) ; L = MS ; (NL ; L) .
 eq U(nil) = null .
 eq U(MS) = MS .
 eq U(MS ; NL) = MS, U(NL) .
endfm
```

We already established that the TR-NATURAL *module is sort-decreasing and well-founded recursive in Example 7. Moreover, it is non-overlapping and thus (by termination) confluent. Sufficient completeness can automatically be verified by the Maude sufficient completeness checker (cf. e.g. [15]). The module* TR-MSET-NAT*, restricted to equations explicitly defined in the module and particularly not including the ones from the* TR-NATURAL *module, is sort-decreasing and well-founded recursive as well. This is seen for instance by using the status functions $stat(f) = mul$ for all f and $stat_{ac}(\_,\_) = us$. Confluence of the equations of* TR-MSET-NAT *follows again from non-overlappingness. All ground instances of $card(x)$ are reducible. Furthermore,* TR-MSET-NAT *is a fair extension of* TR-NATURAL*. Hence, it is sort-decreasing, well-founded recursive, confluent and sufficiently complete.*

Finally, consider the module TR-LIST-MSET-NAT *restricted to equations explicitly defined in the module and particularly not including the ones from the* TR-MSET-NAT *module. It is sort-decreasing and well-founded recursive (e.g. $stat(;) = lex$ and $stat(f) = mul$ for all other functions f). Furthermore, it is confluent because all critical pairs are joinable. All ground instances of $(x_1; x_2)$ are either reducible or constructor terms and all ground instances of $U(x)$ are reducible. As* TR-LIST-MSET-NAT *is a fair extension of* TR-MSET-NAT *it is thus sort-decreasing, well-founded recursive (thus terminating), confluent and sufficiently complete.*

5. A Variant-Based Theory Transformation

So far, our incremental methods for checking the sort-decreasingness, confluence, termination, and sufficient completeness of order-sorted well-founded recursive specifications modulo B have been developed for the case where B can only have commutativity and/or associativity-commutativity axioms. But we are interested in checking the confluence, termination, and sufficient completeness of more general order-sorted specifications $\mathcal{E} = (\Sigma, B, R)$ where B can have any combination of associativity and/or commutativity and/or identity axioms (with some restrictions on the case of associativity without commutativity as explained below). The extension of our method to this more general case is accomplished by an automatic theory transformation $(\Sigma, B, R) \mapsto (\Sigma, B_0, \widehat{R} \cup \Delta)$ such that: (i) B_0 only involves commutativity and associativity-commutativity axioms; (ii) the theories $R \cup B$ and $B_0 \cup \widehat{R} \cup \Delta$ are semantically equivalent (as inductive theories, see below); and (iii) (Σ, B, R) is confluent, terminating, and sufficiently complete for Ω modulo B iff $(\Sigma, B_0, \widehat{R} \cup \Delta)$ has the same properties modulo B_0. Here we summarize and extend the basic ideas of the transformation and refer to [9] for further details.

The first key idea is to decompose B as a disjoint union $B = B_0 \cup \Delta$ so that (Σ, B_0, Δ) is confluent and terminating modulo B_0, and Δ contains all its B_0-extensions (cf. e.g. [26, Definition 10.4]). The second key idea is to generate the transformed rules $\widehat{R}$ by computing the most general Δ, B-*variants* ([7]) of the left-hand sides l for the rules $l \to r$ in R. Given a term t, a Δ, B-variant of t is a Δ, B-canonical form u of an instance of t by some substitution θ; more precisely, it is a pair (u, θ). Some variants are more general than others, so that variants form a preorder in a natural way. The set $\widehat{R}$ then consists of all rules $\widehat{l} \to r\theta$ such that $(\widehat{l}, \theta)$ is a maximal variant of l for $l \to r$ a rule in R. Our transformation $(\Sigma, B, R) \mapsto (\Sigma, B_0, \widehat{R} \cup \Delta)$ is actually the composition of two simpler transformations of this kind:

$$(\Sigma, B, R) \mapsto (\Sigma, B_1, \widehat{R}_1 \cup \Delta_1) \mapsto (\Sigma, B_0, \widehat{R} \cup \Delta)$$

where B_1 is obtained by removing *all identity axioms*[10] Δ_1 from B, and B_0 is obtained by removing from B_1 all axioms that are associative but not commutative, so that Δ is the union of Δ_1 and such associativity axioms oriented (in one of the two directions) as rules. In this way, B_0 only contains commutativity and/or associativity-commutativity axioms. We then incrementally check the confluence, termination, and sufficient completeness of (Σ, B, R) modulo B by checking the same properties modulo B_0 for the semantically equivalent theory $(\Sigma, B_0, \widehat{R} \cup \Delta)$ according to the methods already developed in Sections 3 and 4.

[10] By adding a fresh top sort to each connected component as explained in Footnote 2, we only need to add a pair of identity rules $f(x, e) \to x$ and $f(e, x) \to x$, with x of sort $[s]$, for each connected component $[s]$ involving such axioms.

For the first transformation $(\Sigma, B, R) \mapsto (\Sigma, B_1, \widehat{R}_1 \cup \Delta_1)$ we are always guaranteed that the set of rules $\widehat{R}_1$ is finite if R is (see [9]). However, for the second transformation $(\Sigma, B_1, \widehat{R}_1 \cup \Delta_1) \mapsto (\Sigma, B_0, \widehat{R} \cup \Delta)$, which removes associative but not commutative axioms from B_1, we cannot in general guarantee that $(\Sigma, B_0, \widehat{R} \cup \Delta)$ is a finite theory. However, the *use of subsorts* can make it often the case in practice that $(\Sigma, B_0, \widehat{R} \cup \Delta)$ is finite. We can illustrate this interesting phenomenon with our running example. The first transformation, removing identities, leaves the equation `U(MS ; NL) = MS, U(NL)` unchanged because, since `NL` has sort `NeList`, the identity rules for `_;_` cannot be applied to any instance of `MS ; NL`. By orienting the associativity axiom as a rule `(L ; P); Q` $\rightarrow$ `L; (P ; Q)`, the only variant of the equation `U(MS ; NL) = MS, U(NL)` is itself, since the left-hand side of the associativity rule fails to have an order-sorted unifier with the subterm `MS ; NL`. Therefore, the second transformation also succeeds in our running example (for the resulting transformed modules see Examples 7 and 9).

For well-founded recursive specifications containing operators f that are associative but not commutative (with or without identity) we need to impose some conditions on such f and slightly modify the version in [9] of the second transformation $(\Sigma, B_1, \widehat{R}_1 \cup \Delta_1) \mapsto (\Sigma, B_0, \widehat{R} \cup \Delta)$. There should be only one such operator per connected component, with only two overloadings, which must be

1. either of the form $f : List\ List \rightarrow List$, $f : Elt\ NeList \rightarrow NeList\ [ctor]$, with $Elt < NeList < List$,
2. or of the form $f : List\ List \rightarrow List$, $f : NeList\ Elt \rightarrow NeList\ [ctor]$, with $Elt < NeList < List$.

Moreover, there may be no other constructors of sort $List$ or lower except those of sort Elt or lower. The names Elt, $NeList$, and $List$ are immaterial and are only used to respectively suggest sorts for list elements, nonempty lists, and general lists. Furthermore, in order to make sure that the associativity equations introduced by the second transformation have constructor patterns below their top function symbol (so that the conditions in Section 3 apply to the transformed theory $(\Sigma, B_0, \widehat{R} \cup \Delta)$), instead of introducing an associativity rule

$$f(f(L, P), Q) \rightarrow f(L, f(P, Q))$$

for case (1) (resp. $f(L, f(P, Q)) \rightarrow f(f(L, P), Q)$) for case (2)) with L, P, Q of sort $List$, we introduce a more restricted rule

$$f(f(E, NL), Q) \rightarrow f(E, f(NL, Q))$$

for case (1) (resp. $f(Q, f(NL, E)) \rightarrow f(f(Q, NL), E)$) for case (2)) with E of sort Elt, NL of sort $NeList$, and Q of sort $List$. It is then easy to check that: (i) the left-hand sides of these more restricted rules have constructor patterns below and have no nontrivial overlaps with themselves; (ii) f so defined is sufficiently complete; and (iii) the unrestricted associativity equations are *inductive theorems* of the specification based on the more restricted associativity equations; that is, with this modified second transformation the theories $(\Sigma, B_1, \widehat{R}_1 \cup \Delta_1)$ and $(\Sigma, B_0, \widehat{R} \cup \Delta)$, although no longer equivalent as OS theories, are nevertheless *inductively equivalent* in the sense that their initial algebras $\mathcal{T}_{\Sigma, B_1 \cup \widehat{R}_1 \cup \Delta_1}$ and $\mathcal{T}_{\Sigma, B_0 \cup \widehat{R} \cup \Delta}$ are isomorphic. Indeed, we have

Lemma 1. *Under the above restrictions on the first typing of an associative operator f, the associativity equation $f(f(L, P), Q) = f(L, f(P, Q))$ is an inductive consequence of the restricted associativity equation*

$$f(f(E, NL), Q) = f(E, f(NL, Q))$$

. Likewise, under the second typing the associativity equation $f(L, f(P, Q)) = f(f(L, P), Q)$ is an inductive consequence of the restricted associativity equation

$$f(Q, f(NL, E)) = f(f(Q, NL), E))$$

.

In practice these restrictions are not too strong, since we can automatically ensure typings (1) or (2) by introducing them through a *parameterized module for lists*. Furthermore, the restriction of having only one typing of type (1) or (2) per connected component for each associative f can be relaxed to allow several such typings, provided that the corresponding sorts $Elt < NeList < List$ and $Elt' < NeList' < List'$ involved in two different typings are incomparable.

Example 10. *We use our running example to illustrate the two theory transformations*

$$(\Sigma, B, R) \mapsto (\Sigma, B_1, \widehat{R}_1 \cup \Delta_1) \mapsto (\Sigma, B_0, \widehat{R} \cup \Delta)$$

The first transformation, adding identity axioms as explicit equations and computing the variants of rules with respect to identities, gives us the modules:

```
fmod TR1-NATURAL is pr TRUTH-VALUE .
 sort Nat .
 op 0 : -> Nat [ctor] .
 op s : Nat -> Nat [ctor] .
 op _+_ : Nat Nat -> Nat [comm] .
 ops even odd : Nat -> Bool .
 vars N M : Nat .
 eq N + 0 = N .
 eq s(N) + s(M) = s(s(N + M)) .
 eq even(0) = true .
 eq odd(0) = false .
 eq odd(s(N)) = even(N) .
 eq even(s(N)) = odd(N) .
endfm

fmod TR1-MSET-NAT is pr TR1-NATURAL .
 sort MSet .
 subsort Nat < MSet .
 op _,_ : MSet MSet -> MSet [ctor assoc comm] .
 op null : -> MSet [ctor] .
 op card : MSet -> Nat .
 var MS : MSet .
 var N : Nat .
 var X : [MSet] .
 eq X , null = X .
 eq card(null) = 0 .
 eq card(N)= s(0) + card(null) .
 eq card(N,MS)= s(0) + card(MS) .
endfm

fmod TR1-LIST-MSET-NAT is pr TR1-MSET-NAT .
 sorts List NeList .
 subsorts MSet < NeList < List .
 op nil : -> List [ctor] .
 op _;_ : List List -> List [assoc] .
 op _;_ : MSet NeList -> NeList [ctor assoc] .
 op U : List -> MSet .
 var MS : MSet .
 var NL : NeList .
 var L : List.
 var Y : [List] .
 eq Y ; nil = Y .
 eq nil ; Y = Y .
 eq U(nil) = null .
 eq U(MS) = MS .
 eq U(MS ; NL) = MS, U(NL) .
endfm
```

The way the variants of an equation with respect to the identities modulo the C and AC axioms are computed can be illustrated by the equation `card(N,MS)= s(0) + card(MS)` *in the original module* `MSET-NAT`. *Since the variable* `MS` *could collapse by instantiating it to the identity element* `null`*, the equation's left-hand side has two most general variants: (i) itself, so that the original equation is kept, and (ii) the term* `card(N)`*, leading to the new variant equation* `card(N)= s(0) + card(null)` *added to* `TR1-MSET-NAT`. *The result of the second stage of the theory transformation, denoted above as* $(\Sigma, B_1, \widehat{R}_1 \cup \Delta_1) \mapsto (\Sigma, B_0, \widehat{R} \cup \Delta)$*, has already been described in detail in Examples 7 and 9. Note that, since they do not involve associative but not commutative axioms, the modules* `TR1-NAT` *and* `TR1-MSET-NAT` *are not changed by the second transformation.*

6. Related Work and Conclusions

Our work is related to modularity methods for confluence and/or termination of TRSs. A very good survey of the literature on such methods up to 2002 can be found in [24]. One key difference is that, to the best of our knowledge, such work does not address sorts and subsorts, nor (except for, e.g., [19, 23]) rewriting modulo axioms. Another difference is that in some cases the modularity conditions imposed are quite strong, requiring for example disjointness, which is relatively rare in practical module hierarchies. Perhaps the earliest work most closely related to ours is the work on *proper extensions* of term rewriting systems of [24] (cf. also [8] and [27]). The basic idea behind proper extensions is that calls to functions f in right-hand sides of rewrite rules $l \to r$ where $root(l)$ and f are mutually recursive, do not involve defined function symbols from the base theory (or from the extending theory that recursively depend on functions from the base theory) in the arguments of the function call. Our notion of fair extensions of well-founded recursive theories is even more restrictive in this respect, since the arguments of calls to functions in right-hand sides have to be constructor terms if the function in question is mutually recursive with the root of the left-hand side of the rule. Note however, that the advantage of our more restrictive definition is not just its ability to deal with sorts and structural axioms, but also in our case general termination is modular instead of the weaker notion of $C_{\mathcal{E}}$-termination as for proper extensions.

Our work is also related to the hierarchical termination approach of Urbain and Marché ([23, 29]), with their notion of hierarchical extension being similar to ours of fair extension. In some ways our notion is more general, since for us function symbols can appear in both a submodule and a supermodule, but of course our incremental conditions are in other ways stronger so as to ensure termination, whereas in [23, 29] a modular approach to dependency pairs is developed. Furthermore [23] covers the AC case. There is also a rich body of related work on rewriting modulo axioms, e.g. [2, 17, 18, 22, 26, 31]. For termination modulo, related papers include, e.g., [1, 9, 12, 23].

When using well-founded recursive OS theories and fair extensions to create hierarchies of theories, one can verify important properties such as sort-decreasingness, termination, confluence and sufficient completeness incrementally. Hence, at a practical level, when developing equational programs (such as functional modules in Maude), one can follow a programming discipline ensuring that modules are well-founded recursive and module extensions are fair extensions. Sticking to this programming discipline then guarantees that the verification complexity of the properties in question grows roughly linearly with the number of distinct modules. This is a significant improvement compared to existing methods used for the verification of, e.g., termination where experiments show that in practice the verification complexity grows rapidly with increasing size of theories (see also [29]).

Obvious future work includes the mechanization of all the incremental checks described above in a tool, experimentation with such a tool, and the extension of our results to conditional and context-sensitive theories, which are also supported in Maude. Moreover, recent developments in the termination analysis of rewrite systems modulo axioms (cf. e.g. [1]) might allow us to relax the conditions in the notion of well-founded recursion, thus making our approach more widely applicable.

References

[1] B. Alarcón, S. Lucas, and J. Meseguer. A dependency pair framework for A∨C-termination. In P. Ölveczky editor, Proceedings of the 8th International Workshop on Rewriting Logic and its Applications (WRLA'10), *LNCS* 6381, pages 35–51. Springer, 2010.

[2] L. Bachmair and N. Dershowitz. Completion for rewriting modulo a congruence. *Theoretical Computer Science*, 67(2&3):173–201, 1989.

[3] F. Baader and T. Nipkow. *Term rewriting and All That*. Cambridge University Press, 1998.

[4] M. Bezem, J. Klop, and R. Vrijer, editors. *Term Rewriting Systems*. Cambridge Tracts in Theoretical Computer Science 55. Cambridge University Press, 2003.

[5] P. Borovanský, C. Kirchner, H. Kirchner, and P.-E. Moreau. ELAN from a rewriting logic point of view. *Theoretical Computer Science*, 285:155–185, 2002.

[6] M. Clavel, F. Durán, S. Eker, J. Meseguer, P. Lincoln, N. Martí-Oliet and C. Talcott. *All About Maude – A High-Performance Logical Framework*. LNCS 4350, 2007.

[7] H. Comon-Lundh and S. Delaune. The finite variant property: how to get rid of some algebraic properties. In J. Giesl editor, Proceedings of the 16th International Conference on Rewriting Techniques and Applications (RTA'05), LNCS 3467, 294–307, 2005.

[8] N. Dershowitz. Hierarchical termination. In N. Dershowitz and N. Lindenstrauss editors, Proceedings of the 4th International Workshop on Conditional and Typed Rewriting Systems (CTRS-94), *LNCS 968*, pages 89–105. Springer, 1995.

[9] F. Durán, S. Lucas, and J. Meseguer. Termination modulo combinations of equational theories. In S. Ghilardi and R. Sebastiani editors, Proceedings of the 7th International Symposium on Frontiers of Combining Systems (FroCoS'09), *LNAI* 5749, pages 246–262. Springer, 2009.

[10] F. Durán and J. Meseguer. A Church-Rosser checker tool for conditional order-sorted equational Maude specifications. In P. Ölveczky editor, Proceedings of the 8th International Workshop on Rewriting Logic and its Applications (WRLA'10), *LNCS* 6381, pages 69–85. Springer, 2010.

[11] K. Futatsugi and R. Diaconescu. *CafeOBJ Report*. World Scientific, AMAST Series, 1998.

[12] J. Giesl and D. Kapur. Dependency pairs for equational rewriting. In A. Middeldorp editor, Proceedings of the 12th International Conference on Rewriting Techniques and Applications (RTA'01), *LNCS* 2051, pages 93–108. Springer, 2001.

[13] J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In U. Furbach and N. Shankar editors, Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06), *LNCS* 4130, pages 281–286. Springer, 2006.

[14] J. Goguen and J. Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105:217–273, 1992.

[15] J. Hendrix, J. Meseguer, and H. Ohsaki. A sufficient completeness checker for linear order-sorted specifications modulo axioms. In U. Furbach and N. Shankar editors, Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06), *LNCS* 4130, pages 151–155. Springer, 2006.

[16] J. Hendrix, H. Ohsaki, and J. Meseguer. Sufficient completeness checking with propositional tree automata. Technical report,

CS Department University of Illinois at Urbana-Champaign, 2005. http://www.ideals.illinois.edu/handle/2142/11096.

[17] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the Association for Computing Machinery*, 27:797–821, 1980. Preliminary version in *18th Symposium on Mathematical Foundations of Computer Science*, 1977.

[18] J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15(4):1055–1094, Nov. 1986.

[19] J.-P. Jouannaud and Y. Toyama. Modular Church-Rosser modulo: the complete picture. *International Journal of Software and Informatics*, 2(1):61–75, 2008.

[20] D. Kapur and G. Sivakumar. Proving associative-communicative termination using RPO-compatible orderings. In *Selected Papers from Automated Deduction in Classical and Non-Classical Logics LNCS 1761* pages 39–61, Springer 2000.

[21] C. Kirchner, H. Kirchner, and J. Meseguer. Operational semantics of OBJ3. In T. Lepistö and A. Salomaa editors, Proceedings of the 15th International Colloquium on Automata, Languages and Programming (ICALP'88), *LNCS 317*, pages 287–301. Springer, 1988.

[22] C. Marché. Normalised rewriting and normalised completion. In Proceedings of the 9th Annual Symposium on Logic in Computer Science (LICS'94), pages 394–403. IEEE, 1994.

[23] C. Marché and X. Urbain. Modular and incremental proofs of AC-termination. *Journal of Symbolic Computation*, 38(1):873–897, 2004.

[24] E. Ohlebusch. *Advanced Topics in Term Rewriting*. Springer Verlag, 2002.

[25] P. Ölveczky and O. Lysne. Order-sorted termination: The unsorted way. In M. Hanus and M. Rodriguez-Artalejo editors, Proceedings of the 5th International Conference on Algebraic and Logic Programming (ALP'96), *LNCS 1139*, pages 92–106. Springer, 1996.

[26] G. E. Peterson and M. E. Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM*, 28(2):233–264, 1981.

[27] M. R. K. K. Rao. Modular proofs for completeness of hierarchical term rewriting systems. *Theoretical Computer Science*, 151:487–512, 1995.

[28] P. Schneider-Kamp, R. Thiemann, E. Annov M. Codish and J. Giesl. Proving Termination using Recursive Path Orders and SAT Solving. In B. Konev and F. Wolter editors, Proceedings of the 6th International Symposium on Frontiers of Combining Systems (FroCoS'07), *LNCS 4720*pages 267–282, 2007.

[29] X. Urbain. Modular & incremental automated termination proofs. *J. Autom. Reasoning*, 32(4):315–355, 2004.

[30] A. van Deursen, J. Heering, and P. Klint. *Language Prototyping: An Algebraic Specification Approach*. World Scientific, 1996.

[31] P. Viry. Equational rules for rewriting logic. *Theoretical Computer Science*, 285:487–517, 2002.