Order-Sorted Completion: The Many-Sorted Way (Extended Abstract)

Harald Ganzinger*
Fachbereich Informatik, Universität Dortmund
D-4600 Dortmund 50, W. Germany
e-mail: hg@informatik.uni-dortmund.de

Order-sorted specifications can be transformed into equivalent many-sorted ones by using injections to implement subsort relations. In this paper we improve a result of Goguen/Jouannaud/Meseguer about the relation between order-sorted and many-sorted rewriting. We then apply recent techniques in completion of many-sorted conditional equations to systems obtained from translating order-sorted conditional equations. Emphasis will be on ways to overcome some of the problems with non-sort-decreasing rules.

1 Introduction

1.1 Operational Semantics for Order-Sorted Specifications

Many-sorted equational logic is a major candidate to serve as a basis for algebraic specifications and logic programming. Order-sorted equational logic originated with [Gog78] and was further elaborated — in different variations — in [Gogo86], [GM87], [SNGM87], [KKM88], and [Poi88], among others. In [GJM85], an operational semantics for order-sorted specifications based on a translation scheme into many-sorted specifications is introduced. The main idea of this translation is to add auxiliary injection operators $i_{s\subset s'}$ to implement subsort containments $s \subset s'$. This provides the ability to use standard many-sorted concepts of rewriting, completion, and theorem proving in implementations of specification languages based on order-sorted logic.

In [GKK87], a completion procedure specifically tailored to *unconditional* order-sorted equations and based on order-sorted rewriting is proposed as an alternative. The main motivation for this approach is that order-sorted rewriting can be more efficient than naive many-sorted rewriting with the translated rules [KKM88].

The purpose of this paper is not to enter a discussion about efficiency of rewrite relations — we believe that both approaches are of interest in their own right — but to improve both previous approaches. There are two directions of improvement which this paper wants to contribute to.

The first problem left open by the approaches of [GJM85] and [GKK87] is the handling of non-sort-decreasing rules. This problem had been overlooked in [GJM85] as most of the results in this paper which relate order-sorted to many-sorted rewriting are only valid for sort-decreasing rules. Hence, standard Knuth-Bendix completion when applied to the many-sorted translation of order-sorted equations fails whenever a non-sort-decreasing rule is encountered. In this case, the operationally awkward injectivity axiom for the injections has to be taken into consideration — a problem which standard Knuth-Bendix completion is not prepared to handle. For a related reason, the order-sorted completion procedure of [GKK87] fails when a non-sort-decreasing rule is generated. Order-sorted replacement of equals by equals is not a complete proof method for order-sorted deduction in this case [SNGM87].

The second problem is to complete order-sorted specifications with *conditional* equations. Fortunately, the state-of-the-art in completion of many-sorted conditional equations which has originated with [Kap84] has been substantially advanced during the last year, mainly by the work of Rusinowitch and Kounalis [KR88], [Rus87], and by this author [Gan87], [Gan88], [BG88]. (Other related relevant work is described in [ZR84], [ZR85] and [KaR87].) In particular our own approach, based on the method of proof orderings

*This work is partially supported by the ESPRIT-project PROSPECTRA, ref#390.

[Bac87], [BDH86], appears to be well-engineered towards practical use due to its power in simplifying and eliminating equations during completion. This advance in technology is the main reason of why this paper again picks up the translation idea of [GJM85] to develop order-sorted completion the many-sorted way.

As one can immediately see, both problem areas are closely related. To handle non-sort-decreasing rules requires to consider the effect on the equational theory of conditional equations such as the injectivity axiom.

1.2 Summary of Main Results

With regard to the relation between order-sorted rewriting and many-sorted rewriting we improve results by Goguen, Jouannaud and Meseguer. In particular we show that one step of order-sorted rewriting with a set of rules R is one step of many-sorted rewriting with the lowest parses $\lambda(R)$ of R modulo the axioms LP of the lowest parse. This result also holds in the case of non-sort-decreasing rules. It could be made the basis of employing the concepts of completion modulo equations for order-sorted specifications — an idea which will, however, not be investigated any further in this paper. A second result is that to any sort-decreasing canonical system of order-sorted *conditional* rewrite rules there exists an equivalent, canonical many-sorted system (consisting of LP and the lowest parses $\lambda(R_S)$ of the sort specializations R_S of the given order-sorted rules R). This result is not completely obvious as the critical pairs lemma is not true for conditional rewrite rules in general [DOS88]. (A related result in [GJM85] which, by the way, is also only true for non-sort-decreasing rules, employs the composed relation $\rightarrow_{\lambda(R_S)} \circ \rightarrow_{LP}^{n,f}$ in the many-sorted world.) Hence, many-sorted unfailing completion [HR87], [Bac87] will generate the reduced version of this system. The proofs are contained in the full version [Gan88b] of this paper and omitted from this extended abstract.

We show that in many practical cases non-sort-decreasing rules can be replaced by sort-decreasing ones without changing the initial algebra. These replacements are rules with extra variables in the condition and in the right side. Fortunately they belong to the class of what we call *quasi-reductive rules.* Quasi-reductive rules are a generalization of reductive conditional rewrite rules and the associated rewrite process is similarly efficient.

We outline an unfailing completion procedure for conditional equations that can handle both nonreductive equations such as injectivity axioms and quasi-reductive equations as they are introduced during replacing non-sort-decreasing rules. It is an extension of the one which has been presented and proved correct in detail in [Gan87] and [Gan88]. The correctness of the extensions (unfailing completion, quasi-reductive equations) is shown in [BG88] and [BG89]. We demonstrate by means of examples that these techniques perform successfully on practical examples of order-sorted specifications.

2 Basic Notions and Notations

We will only introduce the syntactic aspects of order-sorted logic and refer to [GM87] and [SNGM87] for the two main variants of semantics for order-sorted specifications. In this paper, notions and notations mainly follow [SNGM87] and [GKK87].

Every *variable* x comes with a sort s^x which is a *sort symbol.* For every sort symbol there exist infinitely many variables having this sort.

A *subsort declaration* is an expresion of form $s < s'$, where s and s' are sort symbols. A *function or operator declaration* has the form $f : s_1 \dots s_n \rightarrow s_0$, where n is the arity of f and s_i are sort symbols. An *order-sorted* signature, usually denoted Σ^{os}, is a set of sort symbols, subsort and function declarations. A *many-sorted signature*, usually denoted Σ, is a particular case of an order-sorted signature with an empty set of subsort declarations. In a many-sorted signature we do not allow more than one declaration for any function symbol. By S and Ω we denote the set of sorts and operators, respectively, of a many-sorted signature Σ.

The *subsort order* $s \leq_{\Sigma^{os}} s'$ is the least quasi-order on the sort symbols of Σ^{os} generated by the subsort declarations. Throughout this paper we restrict attention to signatures for which $<_{\Sigma^{os}}$ is a partial order. If the signature is clear from the context, we will omit the subscript Σ^{os} in $<_{\Sigma^{os}}$. $\leq_{\Sigma^{os}}$ extends to tuples of sort symbols of the same length by $(\underline{s}_1, \dots, \underline{s}_n) \leq_{\Sigma^{os}} (s_1, \dots, s_n)$, iff $\underline{s}_i \leq_{\Sigma^{os}} s_i$, for $1 \leq i \leq n$. We write $s <_b s'$, if $s < s'$ and if there does not exist any s'' such that $s < s'' < s'$.

Given a set of variables X, a Σ^{os}-*term of sort* s in $\mathcal{T}_{\Sigma^{os}}(X)_s$ is either a variable x such that $s^x \leq s$, or has the form $f(t_1, \dots, t_n)$, where $f : s_1 \dots s_n \rightarrow s_0$ is a function declaration in Σ^{os} such that $s_0 \leq s$ and

$t_i \in T_{\Sigma^{os}}(X)_{s_i}$ is a term of sort s_i, for $1 \leq i \leq n$.

Many-sorted terms are defined like order-sorted ones. The main difference is that the sort of a many-sorted term t is uniquely determined and will be denoted s^t in this paper. Given a many-sorted signature Σ, by $T_\Sigma(X)_s$ we denote the set of many-sorted terms of sort s over the variables in X. For both order-sorted and many-sorted terms t we use the notation $t : s$ to indicate that t is a term of sort s.

A Σ^{os}-*equation* is a pair of Σ^{os}-terms u and v of the same sort written as $u \doteq v$. A *conditional equation over* Σ^{os} is a formula of form $C \Rightarrow u \doteq v$, where $u \doteq v$ is a Σ^{os}-equation and C is a finite conjunction of Σ^{os}-equations $u_1 \doteq v_1 \wedge \ldots \wedge u_n \doteq v_n$, $n \geq 0$. If E is a set of (conditional) Σ^{os}-equations, by $E^=$ we denote the set

$$E^= = \{C \Rightarrow u \doteq v \mid C \Rightarrow u \doteq v \in E \text{ or } C \Rightarrow v \doteq u \in E\}.$$

An *order-sorted (equational) specification* consists of an order-sorted signature Σ^{os} and a set E of conditional Σ^{os}-equations.

If O is a syntactic Σ^{os}-object, i.e. a term or (conditional) equation, by $var(O)$ we denote the set of variables occurring in O.

A Σ^{os}-*substitution* is a function σ from Σ^{os}-terms to Σ^{os}-terms such that

1. if u is a term of sort s, then $\sigma(u)$ is a term of sort s,
2. $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$,
3. $dom(\sigma) := \{x \mid \sigma(x) \neq x\}$, the *domain* of σ is finite.

We will also use the notation $u\sigma$ for $\sigma(u)$.

An order-sorted signature Σ^{os} is called *pre-regular*, if for any Σ^{os}-function symbol f and every string $s_1 \ldots s_n$ of Σ^{os}-sorts the set $\{\overline{s} \mid (f : \overline{s}_1 \ldots \overline{s}_n \to \overline{s}) \in \Sigma^{os},\ s_1 \ldots s_n \leq_{\Sigma^{os}} \overline{s}_1 \ldots \overline{s}_n\}$ is either empty or has a minimum element wrt. the subsort order of Σ^{os}. Σ^{os} is called *regular*, if for any string $s_1 \ldots s_n$ of Σ^{os}-sorts such that there exists a function declaration $(f : \overline{s}_1 \ldots \overline{s}_n \to \overline{s}) \in \Sigma^{os}$ with $s_1 \ldots s_n \leq_{\Sigma^{os}} \overline{s}_1 \ldots \overline{s}_n$, then there exists a least $(\underline{s}_1, \ldots, \underline{s}_n, \underline{s})$ such that $(f : \underline{s}_1 \ldots \underline{s}_n \to \underline{s}) \in \Sigma^{os}$ and $s_1 \ldots s_n \leq_{\Sigma^{os}} \underline{s}_1 \ldots \underline{s}_n$. Regularity of a signature implies pre-regularity.

Pre-regularity is required for the existence of initial algebras in the semantics of [SNGM87]. For the semantics of [GM87], regularity is a sufficient condition for the existence of initial algebras. Although the notion of order-sorted deduction which is used in this paper corresponds to the semantics of [GM87], pre-regulary will already be a sufficient condition for the syntactic properites on which our approach of order-sorted completion is based.

An order-sorted signature is called *coherent*, if each equivalence class of sorts under the equivalence closure of $\leq_{\Sigma^{os}}$ has a maximal element.

From now on we will assume order-sorted signatures to be pre-regular and coherent.

3 Translation of Order-Sorted Specifications

3.1 Many-Sorted Representations of Order-Sorted Terms

Definition 3.1 *Let Σ^{os} be an order-sorted signature. Its* **translation** *into a corresponding many-sorted signature Σ is defined as follows:*

1. *The sorts in S are the sorts in S^{os}.*
2. *If $f : s_1 \ldots s_n \to s_0$ is an operator declaration in Σ^{os}, then $f_{s_1 \ldots s_n \to s_0} : s_1 \ldots s_n \to s_0 \in \Omega$.*
3. *If $s <_b s'$ in Σ^{os}, then $i_{s \subset s'} \in \Omega$.*

The translation disambiguates overloaded function symbols and introduces injections $i_{s \subset s'}$ to represent the containment of s in s'. Injections along chains of subsort relations can be represented by terms in elementary injections. As there may be more than one way of going from an s to an s' we have to order these paths if we want a unique representation. For that purpose we assume the partial subsort order to be extended to an arbitrary but fixed total order $<_t$ on S. The same notation is used for the *lexicographic* extension of $<_t$ to sequences of sorts. (This is different to the extension of $\leq_{\Sigma^{os}}$ to tuples which we have used to define the regularity properties. The latter was defined component-wise.)

Now we define composite injections to proceed along minimal paths in the subsort graph.

Definition 3.2 *Let $s \leq s'$. Let furthermore $s_n s_{n-1} \ldots s_0$, $n \geq 0$, be a sequence of sorts minimal wrt. $<_t$ such that $s_0 = s$, $s_n = s'$, and $s_i <_b s_{i+1}$, for $0 \leq i < n$. Then the* **minimal composite injection**

$$i_{s_{n-1} \subset s_n} \circ i_{s_{n-2} \subset s_{n-1}} \circ \ldots \circ i_{s_0 \subset s_1}$$

from s to s' will be denoted as $I_{s \subset s'}$. (If $s = s'$, $I_{s \subset s'}$ is the identity for which there is no explicit operator symbol in Σ. Hence, terms $I_{s \subset s'}(t)$ and t are not distinguished in this case.)

We can now go on and define mappings between order-sorted terms and their many-sorted representations as terms over Σ. Any many-sorted term t in Σ represents one unique order-sorted term $\omega(t)$ which is obtained by deleting injections and by collapsing the disambiguated operator symbols into the original overloaded symbol.

Definition 3.3 *The mapping $\omega(\_) : T_\Sigma(X) \to T_{\Sigma^{os}}(X)$ is inductively defined as follows:*

1. *If $x : s$ is a variable, then $\omega(x : s) = x : s$.*
2. *$\omega(f_{s_1 \ldots s_n \to s_0}(t_1, \ldots, t_n)) = f(\omega(t_1), \ldots, \omega(t_n))$, for non-injections $f_{s_1 \ldots s_n \to s_0} \in \Omega$.*
3. *$\omega(i_{s \subset s'}(t)) = \omega(t)$, for injections $i_{s \subset s'} \in \Omega$.*

In the reverse direction, $\lambda(\_)$ will compute the lowest parse of an order-sorted term.

Definition 3.4 *The* **lowest parse** *$\lambda(\_) : T_{\Sigma^{os}}(X) \to T_\Sigma(X)$ is inductively defined as follows:*

1. *$\lambda(x : s) = x : s$, for variables $x : s \in X$.*
2. *$\lambda(f(t_1, \ldots, t_n)) = f_{s_1 \ldots s_n \to s_0}(t'_1, \ldots, t'_n)$, where $f : s_1 \ldots s_n \to s_0$ is the operator declaration in Σ^{os} for which $s_0 s_1 \ldots s_n$ is minimal wrt. $<_t$ such that $\lambda(t_i) \in T_\Sigma(X)_{s'_i}$ and $s'_i \leq s_i$, $1 \leq i \leq n$, and where $t'_i = I_{s'_i \subset s_i}(\lambda(t_i))$, $1 \leq i \leq n$.*

As we are putting the codomain s_0 of f at the beginning of the sort sequence $s_0 s_1 \ldots s_n$ when looking for a minimal declaration for f, $\lambda(t)$ will always have a lowest possible sort.

Proposition 3.5 1. $\omega(\lambda(t)) = t$

2. $s^{\lambda(\omega(t))} \leq s^t$

3.2 Computation of Minimal Parses by Rewriting

In the preceeding section we have defined a function $\lambda(\_)$ which produces a many-sorted representation of an order-sorted term. The representation always is a term of a lowest possible sort and, hence, unique for pre-regular signatures. On the other hand, there are usually many different many-sorted terms that represent the same order-sorted term via $\omega(\_)$. In this section we describe a canonical set of rewrite rules over Σ which, for any given many-sorted term, computes the lowest parse of the order-sorted term $\omega(t)$ it represents.

The set of rules consists of rules for computing the minimal path (wrt. $<_t$) between any two sorts $s < s'$ and of rules which represent the inheritance axioms for overloaded function symbols on subsort hierarchies.

Axioms (CI) for composite injections

$$i_{s' \subset s''}(I_{s \subset s'}(x)) = I_{s \subset s''}(x),$$

for $s < s' < s''$, if $I_{s \subset s''} \neq i_{s' \subset s''} \circ I_{s \subset s'}$.

Axioms (INH) for inheritance

$$f_{s_1 \ldots s_n \to s_0}(I_{\tilde{s}_1 \subset s_1}(x_1), \ldots, I_{\tilde{s}_n \subset s_n}(x_n)) = I_{s'_0 \subset s_0}(f_{s'_1 \ldots s'_n \to s'_0}(I_{\tilde{s}_1 \subset s'_1}(x_1), \ldots, I_{\tilde{s}_n \subset s'_n}(x_n))),$$

if $f : s_1 \ldots s_n \to s_0$ and $f : s'_1 \ldots s'_n \to s'_0$ are operator declarations, $s'_0 s'_1 \ldots s'_n <_t s_0 s_1 \ldots s_n$, $s'_0 \leq s_0$, and if $\tilde{s}_i$ are maximal[1] sorts such that $\tilde{s}_i \leq s_i$ and $\tilde{s}_i \leq s'_i$.

[1]To select maximal sorts $\tilde{s}_i$ is not really required. However, (INH)-axioms for maximal $\tilde{s}_i$ subsume (INH)-axioms for non-maximal ones.

We will now prove that the equation system $LP = CI \cup INH$, oriented from left to right, forms a canonical system of rewrite rules. First we will define a precedence on Σ-operators such that the induced recursive path ordering proves the termination of the system.

Definition 3.6 *By $>_I$ we denote the following partial order on Ω:*

1. $i_{s_1 \subset s_2} > i_{s_1' \subset s_2'}$, *iff* $s_2' < s_2$ *or if* $s_2' = s_2$ *and* $s_1' < s_1$, *for injections* $i_{s_1 \subset s_2}$ *and* $i_{s_1' \subset s_2'}$.

2. $f_{s_1 \ldots s_n \to s_0} > i_{s \subset s'}$, *for any order-sorted operator f and any injection* $i_{s \subset s'}$.

3. $f_{s_1 \ldots s_n \to s_0} > f_{s_1' \ldots s_n' \to s_0'}$, *iff* $s_0' s_1' \ldots s_n' <_t s_0 s_1 \ldots s_n$, *for any two declarations* $f : s_1' \ldots s_n' \to s_0'$ *and* $f : s_1 \ldots s_n \to s_0$ *of the same order-sorted operator symbol f.*

By $>_I$ we denote the recursive path ordering on $T_\Sigma(X)$ induced by $>_I$.

Proposition 3.7 *Orienting the equations in LP from left to right into rules $L \to R$, we have $L >_I R$, for any of these rules.*

The confluence of the system will be proved using the following proposition:

Proposition 3.8 *If*

$$J(x) = i_{s_{n-1} \subset s_n} \circ i_{s_{n-2} \subset s_{n-1}} \circ \ldots \circ i_{s_0 \subset s_1}(x)$$

is some composite injection from s_0 to s_n, $n \geq 1$, then $J(x) \to_{CI}^* I_{s_0 \subset s_n}(x)$.

As a consequence we have $J_1(x) \downarrow_{CI} J_2(x)$, for any two composite injections J_1 and J_2 from s to s'.

Lemma 3.9 *For any two Σ-terms $t : s$ and $t' : s'$ such that $s \leq s_0$, and $s' \leq s_0$, we have $\omega(t) = \omega(t')$, iff $I_{s \subset s_0}(t) \equiv_{LP} I_{s' \subset s_0}(t')$ such that the $\equiv_{LP}$-proof only involves intermediate terms smaller than $I_{s \subset s_0}(t)$ or $I_{s' \subset s_0}(t')$ with respect to $>_I$.*

Proposition 3.10 *The set of rules LP is locally confluent, hence confluent by 3.7.*

From 3.7 and 3.10 it follows that LP is canonical. We will now prove that the LP-normalforms of terms t represent the lowest parse of the corresponding order-sorted terms $\omega(t)$. More precisely,

Lemma 3.11 *Let $t \in T_\Sigma(X)_s$ and $s^{\lambda(\omega(t))} =: s' \leq s'' \leq s$.*

1. *$I_{s' \subset s''}(\lambda(\omega(t)))$ is irreducible under $\to_{LP}$.*

2. *$t \geq_I I_{s' \subset s''}(\lambda(\omega(t)))$.*

In particular, from 1, 3.9, 3.7, and 3.10 we have that $t \downarrow_{LP} = I_{s' \subset s}(\lambda(\omega(t)))$.

Altogether we have shown that two terms $t_1, t_2 \in T_\Sigma(X)$ are representations of the same order-sorted terms, iff they are equivalent under LP. Moreover, the equivalence can be decided by rewriting the appropriately injected terms to their $\to_{LP}$-normalforms.

3.3 Order-Sorted Deduction and Rewriting

The notion of order-sorted deduction here is the one for the variant of order-sorted logic in [GM87]. Order-sorted deduction is described by the following set of inference rules, cf. e.g. [GKK87]:

Definition 3.12 (Order-Sorted Deduction) *Let E be a set of order-sorted equations over Σ^{os}.*

Reflexivity

$$E \vdash_X t \doteq t \ ,$$

for any $t \in \mathcal{T}_{\Sigma^{os}}(X)$.

Symmetry

$$\frac{E \vdash_X t \doteq t'}{E \vdash_X t' \doteq t}$$

Transitivity

$$\frac{E \vdash_X t \doteq t',\ E \vdash_X t' \doteq t''}{E \vdash_X t \doteq t''}$$

Congruence

$$\frac{E \vdash_Y \theta(x) \doteq \theta'(x),\ \forall x \in X}{E \vdash_Y \theta(t) \doteq \theta'(t)}\ ,$$

for $\theta, \theta' : X \to \mathcal{T}_{\Sigma^{os}}(Y)$, $t \in \mathcal{T}_{\Sigma^{os}}(X)$

Substitutivity

$$\frac{E \vdash_Y \theta(t_i) \doteq \theta(t_i'),\ 1 \leq i \leq n}{E \vdash_Y \theta(t) \doteq \theta(t')}\ ,$$

for $\eta = \ t_1 \doteq t_1' \wedge \ldots \wedge t_n \doteq t_n' \Rightarrow t \doteq t' \in E$ *and* $\theta : var(\eta) \to \mathcal{T}_{\Sigma^{os}}(Y)$ *a substitution.*

Clearly, $E \vdash_X t \doteq t'$, iff $t \equiv_E^X t'$, where $\equiv_E^X = \cup_{n \in N} \equiv_n^X$, with $\equiv_0^X = \emptyset$ and $t \equiv_n^X t'$, iff $t \equiv_{n-1}^X t'$ or if there exist u_j, u_j' such that $u_j \equiv_{n-1}^X u_j'$ and $t \doteq t'$ can be derived from $u_j \doteq u_j'$ using one of the above inference rules.

We will now extend our notion of lowest parses $\lambda(\_)$ to unconditional equations. Let $t_1 \doteq t_2$ be an order-sorted equation, and assume that $s_i = s^{\lambda(t_i)}$.

$$\lambda(t_1 \doteq t_2) \ = \ I_{s_1 \subset s}(\lambda(t_1)) \doteq I_{s_2 \subset s}(\lambda(t_2)),$$

where s is some minimal supersort of both t_1 and t_2, i.e. $t_1, t_2 \in \mathcal{T}_{\Sigma^{os}}(X)_s$. (Due to the coherence of Σ^{os} such an s exists. There may be more than one choice for s. This, however is irrelevant in our context.) In particular, if $s_2 \leq s_1$, the left side of $\lambda(t_1 \doteq t_2)$ will not have an injection as top symbol.

Let now

$$E^{\#} = CI \cup INH \cup IN \cup \lambda(E),$$

where

$$\lambda(E) = \{\ldots \lambda(t_i \doteq t_i') \ldots \Rightarrow \lambda(t \doteq t') \mid \ldots t_i \doteq t_i' \ldots \Rightarrow t \doteq t' \in E\}$$

are the minimal parses of the equations in E and where

$$IN = \{i_{s \subset s'}(x) \doteq i_{s \subset s'}(y) \Rightarrow x \doteq y \mid i_{s \subset s'} \in \Omega\}$$

is the set of injectivity axioms for the injections in Σ.

The following is the proof-theoretic equivalent of the satisfaction theorem in [GJM85]:

Theorem 3.13 *For* $t_1, t_2 \in T_\Sigma(X)_s$, $t_1 \equiv_{E^{\#}}^X t_2$, *iff* $\omega(t_1) \equiv_E^X \omega(t_2)$.

This theorem proves the equivalence of order-sorted deduction in E with standard many-sorted equational logic in $E^{\#}$.

We now go on and compare order-sorted rewriting to many-sorted rewriting.

Definition 3.14 *An* **order-sorted conditional rewrite rule** *is an order-sorted conditional equation* $C \Rightarrow l \doteq r$ *satisfying* $(var(C) \cup var(r)) \subset var(l)$ *and denoted* $C \Rightarrow l \dot{\to} r$.

Definition 3.15 *A term* $t \in \mathcal{T}_{\Sigma^{os}}(X)$ **rewrites** *to* t' *with a rewrite rule* $\rho = C \Rightarrow l \dot{\to} r$ *in* R *at occurrence* o, *which is denoted* $t \to_{R^X} t' = t[o \leftarrow \sigma(r)]$ *whenever*

1. σ *is a substitution* $\sigma : var(\rho) \to \mathcal{T}_{\Sigma^{os}}(X)$ *such that* $t/o = l\sigma$,
2. *there is a sort* s *such that, for* x *a variable of sort* s, $t[o \leftarrow x]$ *is a well-formed term and* $l\sigma, r\sigma \in \mathcal{T}_{\Sigma^{os}}(X)_s$,
3. *for any* $u \doteq v \in C$ *there exists a term* w *such that* $u\sigma \to_{R^X}^* w$ *and* $v\sigma \to_{R^X}^* w$, *with* $\to_{R^X}^*$ *the reflexive and transitive closure of* $\to_{R^X}$.

If $X = var(t)$, we will also write $\rightarrow_R$ and $\rightarrow_R^*$ for $\rightarrow_{R^X}$ and $\rightarrow_{R^X}^*$, respectively. The smallest fixpoint of this recursion defines $\rightarrow_{R^X}$.

Many-sorted rewriting is defined like order-sorted rewriting in signatures with an empty set of subsort relations. In this case, the second condition of the previous definition becomes trivial.

Theorem 3.16 *$u \rightarrow_{\lambda(R)/LP} v$ iff $\omega(u) \rightarrow_R \omega(v)$.*

This theorem proves that order-sorted rewriting is equivalent to rewriting the many-sorted representations of terms modulo the axioms of the lowest parse LP, using the lowest parses of the order-sorted rules as rewrite rules. If $\rightarrow_R$ is canonical, $\lambda(R)/LP$ is canonical, too. However, rewriting modulo LP does not seem to be very efficient. Fortunately, forming the closure R_S of R by all *specializations* of the rules, will make $\lambda(R_S) \cup LP$ canonical, provided R is sort-decreasing and canonical.

To formally introduce the notion of specialization it is useful to define the notion of a sort assignment. A sort assignment is a map $\alpha : \overline{X} \rightarrow S$, where $\overline{X}$ is the set of names of variables in X. Hence, a sorted set of variables is a pair $(\overline{X}, \alpha)$, denoted X_α. Sort assignments inherit the subsort ordering such that $\alpha \leq \alpha'$, iff $\alpha(x) \leq \alpha'(x)$, for any $x \in \overline{X}$. A specialization is a substitution $\rho : X_\alpha \rightarrow X_{\alpha'}$, where $\alpha' \leq \alpha$, sending $x : \alpha(x)$ to $x : \alpha'(x)$. To *specialize* a order-sorted term or formula ϕ means to apply a specialization to ϕ. If Φ is a set of order-sorted terms of formulas, by Φ_S we denote the set of all specializations of terms or formulas in Φ.

Definition 3.17 *An order-sorted rule $C \Rightarrow s \dot{\rightarrow} t$ is called* **sort-decreasing**, *iff for any specialization ρ, $s\rho \dot{\rightarrow} t\rho$ has a lowest parse such that $s^{\lambda(s\rho)} \geq s^{\lambda(t\rho)}$. A many-sorted rule $C \Rightarrow s \dot{\rightarrow} t$ is called* **sort-decreasing**, *iff the left side s does not carry an injection as its top symbol. A set of rules is sort-decreasing, if each of its members is sort-decreasing.*

An immediate consequence of 3.16 is the following corollary.

Corollary 3.18 *$u \rightarrow_{\lambda(R_S)/LP} v$ iff $\omega(u) \rightarrow_R \omega(v)$.*

Confluence and termination of R carry over to $R_\# = \lambda(R_S) \cup LP$ as we shall see in the next theorem. Let us first make a few remarks about reduction orderings. If we are given a reduction ordering $>$ on $\mathcal{T}_{\Sigma^\infty}(X)$, it can be extended to a reduction ordering $>_{ms}$ on $T_\Sigma(X)$ which is compatible with $\equiv_{LP}$, simply by defining $t >_{ms} t'$, iff $\omega(t) > \omega(t')$. In addition, the transitive closure $\tilde{>}$ of $(>_{ms} \cup >_I)$, where $>_I$ is the recursive path ordering that we have introduced to order the LP axioms, also is a reduction ordering on $T_\Sigma(X)$ which is compatible with LP. This ordering can be used to order $\lambda(R_S) \cup LP$.

Theorem 3.19 *Let R be a set of order-sorted rules.*

1. *R is sort-decreasing, iff $R_\#$ is sort-decreasing.*
2. *Let R be sort-decreasing. R is canonical iff $R_\#$ is canonical with a reduction ordering that is compatible with $\equiv_{LP}$.*
3. *If $R_\#$ is canonical and sort-decreasing, then, $\downarrow_{R_\#} = \equiv_{E^\#}$, i.e. for any two terms $u, v \in T_\Sigma(X)_s$ we have $u \equiv_{E^\#}^X v$, iff $u \downarrow_{R_\#} v$.*

In the case of unconditional rewrite rules R, $R_\#$ is unconditional, too. Unfailing many-sorted completion [HR87], [Bac87] will generate any reduced variant of $R_\#$.

3.4 Elimination of Non-Sort-Decreasing Rules

Theorem 3.19 requires order-sorted rules to be sort-decreasing for the construction of an equivalent canonical system of many-sorted rules. Likewise, order-sorted completion as proposed in [GKK87] requires rules to be sort-decreasing and fails, if non-sort-decreasing rules are generated. We shall see in section 5.1 that translating into many-sorted specifications and applying conditional equation completion (to deal with the injectivity axiom of injections) is successful in simple cases of non sort-decreasing rules. In many interesting cases, like in the subsequent example, the completion procedure which we will describe in section 4 below will not terminate.

Example 3.20

```
sort nzNat < nat, nat < int, nzNat < nzInt, nzInt < int
op
   0 :  nat
   s :  nat -> nzNat
   + :  int*int   -> int, nat*nat     -> nat,  nat*nzNat -> nat
        nzNat*nat -> nat, nzNat*nzNat -> nzNat
   - :  nat -> int, nzNat -> nzInt, int -> int, nzInt -> nzInt
   * :  int*int -> int, nat*nat -> nat
   square :  int*int -> nat
var i:int, j:int, n:nat
axioms
   -(0) = 0
   -(-i) = i
   i+0 = i
   0+i = i
   k+s(m) = s(k+m)
   (-s(k)) + s(m) = (-k) + m
   i + (-j) = -((-i)+j)
   i*0 = 0
   0*i = 0
   i*s(n) = i*n + i
   i*(-j) = -(i * j)
   (-i)*j = -(i * j)
   square(i) = i*i
```

The last axiom, when oriented from left to right, is clearly not sort-decreasing. A specification with the same initial algebra would be the one in which this equation is replaced by

$$i * i \doteq n \Rightarrow square(i) \doteq n,$$

with n a variable of sort *nat*. This equation, when oriented from left to right, is sort-decreasing. However, it has the extra variable n in its condition and right side. The lowest parse of this equation would be

$$i * i \doteq i_{nat \subset int}(n) \Rightarrow square(i) \doteq n.$$

Equations of this kind are usually not admitted as rewrite rules. In fact, we plan to associate a specific operational semantics with it. It should be specifying the replacement of a $square(i)$ by any n which can be obtained from normalizing $i * i$ and type checking the result by matching $i_{nat \subset int}(n)$ with the normalform. If the normalform is unique, this process of finding the substitution for i and n at rewrite-time is completely backtrack-free. Unfortunately, this idea of deterministic oriented goal solving is not a complete goal solving method in general. Fortunately, an adequately designed completion procedure can make it become complete.

Our idea of how one can replace non-sort-decreasing equations by sort-decreasing ones should be obvious, not requiring any further formalization. However, we should be saying something about when this replacement preserves the initial algebra of a specification. We assume to be given a set E of order-sorted equations, as well as its many-sorted equivalent $E^{\#}$.

Definition 3.21 *Let C be a set of unconditional equations, let $t \in T_\Sigma(X)_s$ be a term of sort s, $var(C) \subset X$, and $s' \leq s$. We say that t* **is of type** *s'* **in context** *C, if for any ground substitution σ of the variables in X such that $C\sigma \subset \equiv^{\emptyset}_{E^{\#}}$ there exists a term $u_\sigma \in T_\Sigma(X)_{s'}$ such that $t\sigma \equiv^{\emptyset}_{E^{\#}} I_{s' \subset s}(u_\sigma)$.*

In our above example we have $i * i$ of type *nat* in the empty context as for any ground substitution $i * i$ is equal to $i_{nat \subset int}((i_{nzNat \subset nat} \circ s)^{i*i}(0))$.

Proposition 3.22 *Let $C \Rightarrow I_{s^l \subset s}(l) \doteq I_{s^r \subset s}(r)$ be a conditional equation and let $I_{s^r \subset s}(r)$ be of type s_l in context C. Then, replacing $C \Rightarrow I_{s^l \subset s}(l) \doteq I_{s^r \subset s}(r)$ by*

$$C \wedge I_{s^r \subset s}(r) \doteq I_{s^l \subset s}(x) \Rightarrow l \doteq x,$$

with x a new variable of sort s^l, preserves $\equiv^{\emptyset}_{E^{\#}}$, and hence the initial algebra of the specification.

One half of the proof of this proposition is that paramodulation of $C \Rightarrow I_{s^l \subset s}(l) \doteq I_{s^r \subset s}(r)$ on the condition of the injectivity axioms for the injections will generate the replacement $C \wedge I_{s^r \subset s}(r) \doteq I_{s^l \subset s}(x) \Rightarrow l \doteq x$. We shall see that during completion superpositions of this kind will be performed anyway. Yet, completion very often will not terminate because of other superpositions on the originally given equation. We believe that sufficiently powerful *ground completion procedures*, once they have been developed for conditional equations, can solve this problem in cases where the previous proposition applies. In other words, ground completion of the *original* specification can be expected to terminate in this case.

4 Completion of Many-Sorted Conditional Equations

In this section, we will assume to be given a fixed many-sorted signature Σ. Equations, terms, substitutions, etc. will be taken over this signature, unless specified otherwise. Furthermore, we assume a reduction ordering $>$ to be given on $T_\Sigma(X)$. $>_{st}$ will denote the transitive closure of $> \cup st$, with st the strict subterm ordering. $>_{st}$ is noetherian and stable under substitutions.

4.1 Annotated Equations and Reductive Rewriting

We do not put any restrictions on the syntactic form of conditional equations. In particular, conditions and right sides may have extra variables. To compensate for this permissiveness, the application of equations at rewrite-time will be restricted. Completion will guarantee that this restricted application is complete. Formally, application restrictions can be modelled by considering a given set E of equations as a *generator* for rewrite rules.[2] In particular, the set E^r of *reductive* instances of the equations in E is of interest:

$$E^r = \{C\sigma \Rightarrow s\sigma \dot{\rightarrow} t\sigma \mid C \Rightarrow s \doteq t \in E^{=}, s\sigma > t\sigma, s\sigma > u\sigma, s\sigma > v\sigma, \text{ for any } u \doteq v \in C\}$$

In the general case, $\rightarrow_{E^r}$ can be quite inefficient and require (restricted) paramodulation to solve conditions of equations in E. Furthermore, the computed solutions have to be tested for reductivity. To increase efficiency of rewriting, it can be useful to further restrict application of equations at rewrite-time.

We will annotate equations to specify in which way their use at rewrite-time should be restricted. For the purposes of this paper, an equation can be annotated as *operational* or *nonoperational*. The intuitive meaning is that a nonoperational equation should not contribute at all to the equational theory. Injectivity axioms, for example, should be irrelevant at rewrite-time.

In *operational* equations $C \Rightarrow s \doteq t$, condition equations $u \doteq v \in C$ will be annotated as either *oriented* or *unoriented*. We will use the notation $u \stackrel{=}{=} v$ to indicate the annotation "oriented". For a oriented condition oriented goal solving is wanted. Altogether:

Definition 4.1 *Let E be a set of annotated equations. E is viewed to generate the set E^a of rewrite rules $C\sigma \Rightarrow s\sigma \dot{\rightarrow} t\sigma$ such that*

1. *$C \Rightarrow s \doteq t \in E^{=}$ is annotated as operational and $C\sigma \Rightarrow s\sigma \dot{\rightarrow} t\sigma \in E^r$, i.e. the instance is reductive,*
2. *if $u \stackrel{=}{=} v \in C$, then $v\sigma\sigma'$ is $\rightarrow_{E^a}$-irreducible for any $\rightarrow_{E^a}$-irreducible substitution σ'.*

Clearly, $\rightarrow_{E^a} \subset \rightarrow_{E^r} \subset \rightarrow_E$, where the subset inclusions are proper, hence $\stackrel{*}{\leftrightarrow}_{E^a} \neq \equiv_E$ in general. E is called *complete*, iff $\stackrel{*}{\leftrightarrow}_{E^a} = \equiv_E$ and if $\rightarrow_{E^a}$ is canonical. A completion procedure attempts to complete E in this sense.

In many practical cases, a final system E obtained by completion will have additional properties which make $\rightarrow_{E^a}$ to be efficiently computable. For example, if

$$i * i \stackrel{=}{=} i_{nat \subset int}(n) \Rightarrow square(i) \doteq n.$$

with the condition annotated as oriented, is element of a complete E, $square(i)$ needs only be rewritten for those instances of i for which the $\rightarrow_{E^a}$-normalform of $i * i$ is of form $i_{nat \subset int}(n)$. Moreover, if $i * i$ is smaller in the reduction order than $square(i)$, the replacement n will also always be smaller than $square(i)$, making any application of the equation reductive. No reductivity tests are required at rewrite-time. Equations which have this property will be called *quasi-reductive* below. Note that let-expressions with patterns in functional programming languages such as MIRANDA are another example of equations with oriented conditions, cf. definition of quicksort below.

[2] In [BG88] we develop a more general concept of application restrictions based on a notion of relevant substitutions.

4.2 Quasi-Reductive Equations

To simplify the formal treatment in this section, we can assume that operational equations have oriented conditions only. (If an equation has an unoriented condition $u \stackrel{.}{=} v$, we can replace the latter by the two oriented conditions $u \stackrel{.}{=} x$ and $v \stackrel{.}{=} x$, where x is a new variable.)

In the classical case of unoriented conditions, the class of reductive equations [Kap84], [JW86], allows for efficient rewriting [Kap87]. In particular, conditions of equations are easily proved or disproved, and no goal solving is required. Moreover, there are no reductivity tests required at rewrite-time, as any instance of a reductive equation is reductive.

In the case of oriented goal solving there exists a similarly efficient class of equations. Oriented goal solving $u\sigma \rightarrow_E^* v\sigma$ reduces to normalizing $u\sigma$ and, then, matching $v\sigma$ with the normal form, if any of the variables of u is already bound by the matching of the redex, or by the solution of some other condition equation. To formalize this idea, we will have to look at how variables are bound within an equation. We call a conditional equation $u_1 \stackrel{.}{=} v_1 \wedge \ldots \wedge u_n \stackrel{.}{=} v_n \Rightarrow s \stackrel{.}{=} t$ (with oriented conditions) *deterministic*, if, after appropriately changing the orientation of the consequent and choosing the order of the condition equations, the following holds true:

$$var(u_i) \subset var(s) \cup \bigcup_{1 \leq j < i} (var(u_j) \cup var(v_j)),$$

and

$$var(t) \subset var(s) \cup \bigcup_{j=1}^{n} (var(u_j) \cup var(v_j)).$$

To arrive at a concept for avoiding reductivity proofs at rewrite-time, let us now assume the existence of some enrichment $\Sigma' \supseteq \Sigma$ of the signature such that the given reduction ordering on $T_\Sigma(X)$ can be extended to a reduction ordering on $T_{\Sigma'}(X)$.

Definition 4.2 *A deterministic equation $u_1 \stackrel{.}{=} v_1 \wedge \ldots \wedge u_n \stackrel{.}{=} v_n \Rightarrow s \stackrel{.}{=} t$, $n > 0$, is called* **quasi-reductive**, *if there exists a sequence $h_i(\xi)$ of terms in $T_{\Sigma'}(X)$, $\xi \in X$, such that $s > h_1(u_1)$, $h_i(v_i) \geq h_{i+1}(u_{i+1})$, $1 \leq i < n$, and $h_n(v_n) \geq t$. An unconditional equation $s \stackrel{.}{=} t$ is quasi-reductive, if $s > t$.*

The equation

$$i * i \stackrel{.}{=} i_{nat \subset int}(n) \Rightarrow square(i) \stackrel{.}{=} n$$

becomes quasi-reductive under a recursive path ordering, if $square > *$ in precedence. Choosing, $h_1(\xi) = \xi$, the inequalities $square(i) > i * i$ and $i_{nat \subset int}(n) \geq n$ are obvious. Also quasi-reductive is

$$split(x, l) \stackrel{.}{=} (l_1, l_2) \Rightarrow sort(cons(x, l)) \stackrel{.}{=} append(sort(l_1), cons(x, sort(l_2))),$$

with $h_1(\xi) = f(\xi, x)$, where f is a new auxiliary function symbol. The termination proofs can be given by an appropriately chosen polynomial interpretation. It has to be verified that $f(sort(cons(x, l)), x) > f(split(x, l), x)$ and $f((l_1, l_2), x) \geq append(sort(l_1), cons(x, sort(l_2)))$.

Quasi-reductivity is a proper generalization of reductivity:

Proposition 4.3 *If the equation $u_1 \stackrel{.}{=} u_{n+1} \wedge \ldots \wedge u_n \stackrel{.}{=} u_{2n} \Rightarrow s \stackrel{.}{=} t$ is reductive, then the equation*

$$u_1 \stackrel{.}{=} x_1 \wedge u_{n+1} \stackrel{.}{=} x_1 \wedge \ldots \wedge u_n \stackrel{.}{=} x_n \wedge u_{2n} \stackrel{.}{=} x_n \Rightarrow s \stackrel{.}{=} t,$$

is quasi-reductive, if the x_i are new, pairwise distinct variables.

Lemma 4.4 *Let E be finite and $u_1 \stackrel{.}{=} v_1 \wedge \ldots \wedge u_n \stackrel{.}{=} v_n \Rightarrow s \stackrel{.}{=} t \in E$ a quasi-reductive equation.*

1. *If σ is a substitution such that $u_i\sigma \rightarrow_E^* v_i\sigma$, $1 \leq i \leq n$, then, $s\sigma > t\sigma$.*
2. *If $N' \rightarrow_{E^a} N''$ is decidable for all terms N' such that $N >_{st} N'$, then the applicability of the equation $u_1 \stackrel{.}{=} v_1 \wedge \ldots \wedge u_n \stackrel{.}{=} v_n \Rightarrow s \stackrel{.}{=} t$ in N is decidable.*

Corollary 4.5 *Let E be a set of annotated equations in which any operational equation is quasi-reductive. Then, $\rightarrow_{E^a}$ is decidable.*

For confluent $\rightarrow_{E^a}$, the applicability of a quasi-reductive equation can be decided by matching the left side and, then, for $1 \leq i \leq n$, matching the v_i against the normal forms of the substituted u_i to obtain another part of the substitution. As quasi-reductive equations are deterministic, each variable in u_i is bound at the time when the i-th condition is to be checked. Computing the substitution σ is completely backtrack-free in this case. Moreover, no termination proofs are required at rewrite-time.

4.3 Completion Inferences and Strategies

In this abstract we will only briefly describe the basic inference rules and fairness requirements in completion of annotated, application-restricted equations. For details we refer to the full version [Gan88b] and to [Gan87], [Gan88] and [BG88].

Standard completion CC in the conditional case according to the concepts in [Gan87] and further refined in [BG88] consists of three inference rules for adding consequences, one rule for simplification and one rule for elimination of conditional equations. Conseqences are added by

- paramodulating an equation on the consequent of an equation (i.e. computation of contextual critical pairs)
- paramodulating an equation on some condition of an equation
- resolving some condition of an equation with $x \doteq x$.

Paramodulation is restricted in that terms are never replaced by bigger terms in the reduction ordering. Also, superposition is limited to the nonvariable part of a literal.

Equations $C \Rightarrow s \doteq t$ are simplified by rewriting with quasi-reductive equations, using the (skolemized) condition equations C as additional rewrite rules.

An equation $C \Rightarrow s \doteq t$ can be eliminated, if the current set E of equations admits a proof of $C \vdash s \doteq t$ which is simpler wrt. the proof ordering than the proof in which $C \Rightarrow s \doteq t$ is applied under the identity substitution to the hypotheses C. In practice, the different proofs of $C \vdash_E s \doteq t$ have to be enumerated to a certain depth and their complexities compared against the complexity of the proof which has lead to the creation of $C \Rightarrow s \doteq t$, cf. [Gan88].

The fairness requirements in CC-inference rule application depend on the annotations of the equations. The general case is described in [BG88] and [Gan88b]. As a particularly interesting subcase we mention the following result:

Theorem 4.6 *A CC-derivation $E_0, E_1, \ldots$ is* **fair**, *i.e. the final system $E_\infty = \cup_j \cap_{k \geq j} E_k$ is complete, if the following holds true:*

1. *E_∞ does not contain any unconditional equation annotated as nonoperational.*
2. *Any operational equation in E_∞ is quasi-reductive.*
3. *$\cup_k E_k$ contains all instances of each nonoperational equation $\eta \in E_\infty$ which can be obtained by paramodulating operational equations of E_∞ on one selected condition of η.*
4. *$\cup_k E_k$ contains all instances of each nonoperational equation $\eta \in E_\infty$ which can be obtained by resolving the same selected condition of η by $x \doteq x$.*
5. *$\cup_k E_k$ contains all critical pairs between operational equations in E_∞.*
6. *$\cup_k E_k$ contains all instances of each operational equation $\eta \in E_\infty$ which can be obtained by paramodulating operational equations of E_∞ on the right sides of the oriented condition equations of η.*

5 Order-Sorted Completion: The Many-Sorted Way

In this section we illustrate by means of examples that our techniques of completion for conditional equations can be successfully applied to order-sorted specifications. In the examples, equations will be operational, unless labelled by a "-". Moreover, we rearrange conditions of nonoperational equations such that the first condition is always the one which is selected for superposition. *Operational equations will all be reductive or quasi-reductive with the given orientation of literals and ordering of conditions.*

5.1 Smolka's Example

Our first example is due to Smolka and shows the incompleteness of order-sorted replacement of equals by equals, cf. [SNGM87] or [KKM88], in the case of non sort-decreasing rules.

Example 5.1

```
sort  s1 < s2
op  a:s1, b:s1, d:s2, f:s1 -> s1
axioms
   a = d
   b = d
```

In this example, $f(a) \doteq f(b)$ can be derived by order-sorted deduction, however it cannot be proven by replacement of equals by equals. The many-sorted equivalent $E^{\#}$ consists of the folllowing equations:

Example 5.2

```
1    i(a) = d
2    i(b) = d
3-   i(x) = i(y)  => x = y
```

where $i : s1 \to s2$ is the injection $i_{s1 \subset s2}$. Axiom 3 is the injectivity property of i. Orienting 1 and 2 from left to right creates the following final system of equations:

Example 5.3

```
1    i(a) = d
3-   i(x) = i(y)  => x = y
4-   i(x) = d     => x = a
5    b = a
```

Equation 4 is generated from superposing equation 1 on the condition of the nonoperational injectivity axiom 3 (cf. fairness constraint 3). We have here decided to classify 4 as nonoperational although it becomes a quasi-reductive equation when orienting its literals from right to left. After this, 4 generates equation 5 from superposition with equation 2. If $b > a$ in precedence, equation 5 is reductive, allowing to eliminate equation 2 by reduction. Any other superposition on the condition of 3 or 4 does only generate equations which can later be eliminated by the inference rules of simplification and elimination.

5.2 Squares of Integers

We return to the specification of integers as given in example 3.20. The result of translating into many-sorted and completing this system is the following:

Example 5.4

```
1    -(0) = int(0)
2    - (- (i:int))    = i
2a   - (- (X1:nzInt)) = X1
2b   - (- (X1:nat))   = int(X1:nat)
2c   - (- (X1:nzNat)) = nzInt(X1:nzNat)
3    (i:int)+int(0) = i
3a   (X2:nat)+0     = X2
3b   (X:nzNat)+0    = nat(X:nzNat)
4    int(0)+ (i:int) = i
4a   0+ (X1:nat)     = X1
5    (k:nat)+nat(s(m:nat))              = nat(s((k:nat)+ (m:nat)))
5a   (X2:nzNat)+nat(s(m:nat))           = nat(s((X2:nzNat)+ (m:nat)))
6    int(-s(k:nat))+int(nzInt(s(m:nat))) = - (k:nat)+int(m:nat)
7    (i:int)+ (- (j:int))      = - (- (i:int)+ (j:int))
7a   (i:int)+int(- (X1:nzInt)) = - (- (i:int)+int(X1:nzInt))
7b   (i:int)+int(- (X1:nzNat)) = - (- (i:int)+int(nzInt(X1:nzNat)))
7c   (i:int)+ (- (X1:nat))     = - (- (i:int)+int(X1:nat))
8    (i:int)*int(0)  = int(0)
8a   (X2:nat)*0      = 0
9    int(0)* (i:int) = int(0)
```

```
9a   0* (X1:nat)     = 0
10  (i:int)*int(nzInt(s(m:nat))) = (i:int)*int(m:nat)+ (i:int)
10a (X2:nat)*nat(s(m:nat))       = (X2:nat)* (m:nat)+ (X2:nat)
11  (i:int)* (- (j:int))      = -(i:int)* (j:int)
11a (i:int)*int(- (X1:nzInt)) = -(i:int)*int(X1:nzInt)
11b (i:int)* (- (X1:nat))     = -(i:int)*int(X1:nat)
11c (i:int)*int(- (X1:nzNat)) = -(i:int)*int(nzInt(X1:nzNat))
12  (- (i:int))* (j:int)      = -(i:int)* (j:int)
12a int(- (X1:nzInt))* (j:int) = -int(X1:nzInt)* (j:int)
12b (- (X1:nat))* (j:int)      = -int(X1:nat)* (j:int)
12c int(- (X1:nzNat))* (j:int) = -int(nzInt(X1:nzNat))* (j:int)
13  (i:int)* (i:int) ≅ int(k:nat)          => square(i:int) = k
13a (i:int)* (i:int) ≅ int(nzInt(X:nzNat)) => square(i:int) = nat(X:nzNat)
 -----
I1  -int(X1:nat)     = - (X1:nat)
I2  -int(X1:nzInt)   = int(- (X1:nzInt))
I3  -nat(X1:nzNat)   = int(- (X1:nzNat))
I5  -nzInt(X1:nzNat) = - (X1:nzNat)
I6  int(X2:nat)+int(X1:nat) = int((X2:nat)+ (X1:nat))
I7  nat(X2:nzNat)+ (X1:nat) = (X2:nzNat)+ (X1:nat)
I8  int(X2:nat)*int(X1:nat) = int((X2:nat)* (X1:nat))
I9  int(nat(X:nzNat)) = int(nzInt(X:nzNat))
I10 int(X2:nat)+int(nzInt(X:nzNat)) = int((X2:nat)+nat(X:nzNat))
I11 int(nzInt(X:nzNat))+int(X1:nat) = int((X:nzNat)+ (X1:nat))
I12 int(nzInt(X:nzNat))+int(nzInt(Y:nzNat)) = int((X:nzNat)+nat(Y:nzNat))
I13 int(X2:nat)*int(nzInt(X:nzNat)) = int((X2:nat)*nat(X:nzNat))
I14 int(nzInt(X:nzNat))*int(X1:nat) = int(nat(X:nzNat)* (X1:nat))
I15 int(nzInt(X:nzNat))*int(nzInt(Y:nzNat)) = int(nat(X:nzNat)*nat(Y:nzNat))
 -----
i1-    nat(X:nzNat) = nat(Y:nzNat) => X = Y
i2-    int(X:nat)   = int(Y:nat)   => X = Y
i3-    nzInt(X:nzNat) = nzInt(Y:nzNat) => X = Y
i4-    int(X:nzInt)   = int(Y:nzInt)   => X = Y
i5-    int(nzInt(X:nzNat))  = int(Y:nat) => nat(X:nzNat) = Y
i6-    int(nzInt(X1:nzNat)) = int(nzInt(X:nzNat))
          and int(nzInt(X:nzNat)) = (i1:int)* (i1:int) => X = X1
```

The initial set of many-sorted equations $E^{\#}$ consists of the equations 1–13, $I1$–$I9$, and $i1$–$i4$.[3] The remaining equations are generated during completion. The (INH)-equations have a number in $I1$–$I8$ or $I10$–$I15$, equation $I9$ is the only (CI)-axiom. The (nonoperational) injectivity axioms are the equations $i1$–$i4$. It is sufficient to start completion with just the (INH)-equations $I1$–$I8$ between any two neighboring operators (wrt. $<_t$), as the remaining ones are generated as critical pairs.

The final system also contains the lowest parses of many specializations (indicated by letters $a, b, c, \ldots$) of the initial order-sorted rules. This is in accordance with theorem 3.19. In a *reduced* final system like the one above, however, not all specializations need to be present. Equation $13a$ has been generated from superposing $I9$ on the right side of the condition of 13, cf. 4.6. In this example, completion has just verified the completeness of the initial system. No new order-sorted equation has been generated. The new equations on the many-sorted level serve to synchronize the application of order-sorted rules with the computation of lowest parses.

Acknowledgements. The author is grateful to H. Bertling for many discussions on the subject of this paper.

6 References

[Bac87] Bachmair, L.: Proof methods for equational theories. PhD-Thesis, U. of Illinois, Urbana Champaign, 1987.

[BDH86] Bachmair, L., Dershowitz, N. and Hsiang, J.: Proof orderings for equational proofs. Proc. LICS 86, 346-357.

[3] Injections $i_{s \subset s'}$ are ambiguously denoted by their codomain. s', e.g. int denotes both $i_{nzInt \subset int}$ and $i_{nat \subset int}$. The order-sorted rather than the many-sorted function symbols are used. $x : s$ denotes a variable x of sort s.

[BG88] Bertling, H. and Ganzinger, H.: Completion-time optimization of rewrite-time goal solving. Report M.1.3-R-12.0, PROSPECTRA-Project, FB Informatik, U. Dortmund, 1988.

[BG89] Bertling, H. and Ganzinger, H.: Completion of application-restricted conditional equations. Report, FB Informatik, U. Dortmund, 1989, to appear.

[BGS88] Bertling, H., Ganzinger, H. and Schäfers, R.: CEC: A system for conditional equational completion. User Manual Version 1.4, PROSPECTRA-Report M.1.3-R-7.0, U. Dortmund, 1988.

[DOS88] Dershowitz, N., Okada, M. and Sivakumar, G.: Confluence of conditional rewrite systems. Proc. 1st Int'l Workshop on Conditional Term Rewriting, Orsay, 1987, Springer LNCS 308, 1988, 31–44.

[Gan87] Ganzinger, H.: A Completion procedure for conditional equations. Report 234, U. Dortmund, 1987, also in: Proc. 1st Int'l Workshop on Conditional Term Rewriting, Orsay, 1987, Springer LNCS 308, 1988, 62–83 (revised version to appear in J. Symb. Computation).

[Gan88] Ganzinger, H.: Completion with History-Dependent Complexities for Generated Equations. In Sannella, Tarlecki (eds.): Recent Trends in Data Type Specifications. Springer LNCS 332, 1988, 73–91.

[Gan88b] Ganzinger, H.: Order-sorted completion: the many-sorted way (full version). Report 274, FB Informatik, Univ. Dortmund, 1988.

[GJM85] Goguen, J.A., Jouannaud, J.-P. and Meseguer, J.: Operational semantics for order-sorted algebra. Proc. 12th ICALP, 1985, 221–231.

[GKK87] Gnaedig, I., Kirchner, C. and Kirchner, H.: Equational completion in order-sorted algebra. Proc. CAAP '88, Springer LNCS 299, 1988, 165–184.

[GM87] Goguen, J.A. and Meseguer, J.: Order-sorted algebra I: partial and overloaded operators, errors and inheritance. Comp. Sci. Lab, SRI International, 1987.

[Gog78] Goguen, J.A.: Order-sorted algebra. Semantics and theory of computation. Report No. 14, UCLA Computer Science Department, 1978.

[Gogo86] Gogolla, M.: Partiell geordnete Sortenmengen und deren Anwendung zur Fehlerbehandlung in abstrakten Datentypen. PhD Thesis, FB Informatik, U. Braunschweig, West Germany, 1986.

[HR87] J. Hsiang, M. Rusinowitch: On word problems in equational theories, Int. Coll. on Automata Languages and Programming, Springer LNCS, 1987.

[JW86] Jouannaud, J.P. and Waldmann, B.: Reductive conditional term rewriting systems. Proc. 3rd TC2 Working Conference on the Formal Description of Prog. Concepts, Ebberup, Denmark, Aug. 1986, North-Holland.

[Kap84] Kaplan, St.: Fair conditional term rewrite systems: unification, termination and confluence. Report 194, U. de Paris-Sud, Centre d'Orsay, Nov. 1984.

[Kap87] Kaplan, St.: A compiler for conditional term rewriting. Proc. RTA 1987, LNCS 256, 1987, 25-41.

[KaR87] Kaplan, St. and Remy J.-L.: Completion algorithms for conditional rewriting systems. MCC Workshop on Resolution of Equations in Algebraic Structures, Austin, May 1987.

[KKM88] Kirchner, C, Kirchner, H., and Meseguer, J.: Operational semantics of OBJ3. Proc. ICALP 88, Springer LNCS, 1988.

[KR88] Kounalis, E. and Rusinowitch, M.: On word problems in Horn theories. Proc. 1st Int'l Workshop on Conditional Term Rewriting, Orsay, 1987, Springer LNCS 308, 1988, 144–160.

[Poi88] Poigné, A.: Partial algebras, subsorting, and dependent types. In Sannella, Tarlecki (eds.): Recent Trends in Data Type Specifications. Springer LNCS 332, 1988, 208–234.

[Rus87] Rusinowitch, M.: Theorem-proving with resolution and superposition: an extension of Knuth and Bendix procedure as a complete set of inference rules. Report 87-R-128, CRIN, Nancy, 1987.

[SNGM87] Smolka, G., Nutt, W., Goguen, J.A. and Meseguer, J.: Order-sorted equational computation. SEKI Report SR-87-14, U. Kaiserslautern, 1987.

[WB83] Winkler, F. and Buchberger, B.: A criterion for eliminating unnecessary reductions in the Knuth-Bendix algorithm. Coll. on Algebra, Combinatorics and Logic in Comp. Sci., Györ, 1983.

[ZR85] Zhang, H. and Remy, J.L.: Contextual rewriting. Conf. on Rewriting Techniques and Applications, Dijon 1985, Springer LNCS 202, 46–62.