Order-sorted Term Rewriting

A. J. J. DICK* AND P. WATSON†

* *Informatics Dept, Systems Engineering Division, Rutherford Appleton Laboratory, Chilton, Didcot, Oxon OX14 4LZ.*‡

† *RHBNC, University of London (now at University of Glasgow).*

In this article we look at the motivation for order-sorted term rewriting by considering some of the very natural equational theories in which single-sorted (unsorted) and many-sorted rewriting lack sufficient expressiveness, for various reasons. However, order-sorted rewriting is not without problems of its own, and we consider some of these along with a brief description in each case of some current approaches to their solution.

Received August 1990

1. UNSORTED OR SINGLE-SORTED REWRITING

We assume some familiarity with the general idea of term rewriting, and in particular with the Knuth–Bendix completion algorithm, as found in, for example, Refs 1 or 2. A brief summary follows.

We assume that we have a set of *equations* E, whose terms are composed of *constants*, *variables*, and *function symbols*, all of which are defined in our signature. Let V be our set of variables and T our set of terms. A *ground term* is one which contains no variables.

In unsorted term rewriting, all constants and variables are considered to belong to some set S and a function symbol f of arity n is defined:

$$f: S^n \to S$$

We *orient* each equation into a *rewrite rule* in accordance with some ordering $>$ on terms. Each rule has a left hand side, l, and a right hand side, r, chosen so that $l > r$. Then we write the rule:

$$l \Rightarrow r$$

These rules may now be used in two ways. A rule may *reduce* a term in another rule or equation if some term t in that rule or equation is *matched* by l, i.e. there exists a *substitution*

$$\sigma: V \to T$$

such that σ is defined on every variable in l, and

$$\sigma l = t$$

Then we replace t by σr at the position p where t occurred in the rule or equation. Suppose t is a subterm of s in the equation

$$s = u$$

Then we write

$$s' = s[p \leftarrow \sigma r] = \sigma u$$

We will try to choose a well-founded ordering so that no term can be reduced infinitely often. For a review of termination orderings see Ref. 3.

A term which cannot be reduced by any rule is said to be in *normal form*.

The other use we make of rules is in generating new equations which we add to E. Consider the case where there exist rules

$$l_1 \to r_1$$

and

$$l_2 \to r_2$$

and a term t which can be reduced by the first rule at position p_1 and by the second rule at position top. Then t rewrites to both

$$t_1 = t[p_1 \leftarrow \sigma_1 r_1]$$

and

$$t_2 = \sigma_2 r_2$$

for some substitutions σ_1, σ_2. Since the rules are really equations, we deduce that $t_1 = t_2$. We say t_1 and t_2 form a *critical pair*. If t_1 and t_2 can be reduced to the same normal form by the rules they form a *convergent* critical pair. Otherwise $t_1 = t_2$ is added to E as a new equation.

When all critical pairs are convergent and every equation has been ordered into a rule we say the set of rules is *confluent*.

For many sets of equations, no confluent set of rules can be found. This can be due to the confluent set being infinite, or it can be because some equation cannot be oriented.

The process of finding critical pairs involves *unification*, which we do not discuss further here – see Ref. 1 for an introduction to this highly complex area.

1.1 Problems of unsorted term rewriting

There are many natural sets of equations which unsorted term rewriting cannot handle satisfactorily. The most commonly seen example is that of a stack, the signature of which follows.

emp: $\to S$
(emp is a constant, representing the empty stack)

$V = \{x, x1, x2, \ldots, y, y1, y2, \ldots\}$
(V is the set of variables)

push: $S \times S \to S$

pop: $S \to S$

top: $S \to D$
(The three usual functions on stacks)

The equations are:

$\text{pop}(\text{push}(x, y)) = y$

$\text{top}(\text{push}(x, y)) = x$

there are two equations in E

‡ Now at: Bull SA, 68 route de Versailles, 78430 Louveciennes, France.

Now already we have a problem. If we attempt to apply top or pop to the empty stack, we should get an error. So we introduce the additional function symbol error and additional equations as follows:

$\text{error}: \rightarrow \#$

$\text{pop}(\text{emp}) = \text{error}$

$\text{top}(\text{emp}) = \text{error}$

$\text{push}(x, \text{error}) = \text{error}$

$\text{top}(\text{error}) = \text{error}$

$\text{pop}(\text{error}) = \text{error}$

This does not yet solve our problem. Consider the term

$\text{top}(\text{push}(x, \text{error}))$

This rewrites to both x and error, so we have the critical pair

$$x = \text{error}$$

which is not what we intended.

The problem arises because error is qualitatively different from all other terms, just as stacks and stack elements are qualitatively different. This leads to the idea of many-sorted term rewriting.

2. MANY-SORTED TERM REWRITING

The problems we faced in the example above were caused by our attempting to apply functions to constants to which they should not be applied, in other words generating errors.

We solve these problems by defining every function to have a *domain*, which includes one or more *sorts*, and never applying the function outside this domain. Each sort will have its own set of variables, and we restrict substitutions so that any variable can only be substituted by a term of the same sort. Let S be our (finite) set of sorts.

Then the example of stacks can be expressed as follows:

$S = \{\text{STACK, E, NSTACK}\}$

(representing the three sorts of stacks, elements and non-empty stacks, respectively)

$V_{\text{E}} = \{e, e1, e2, \ldots\}$

$V_{\text{STACK}} = \{y, y1, y2, \ldots\}$

$V_{\text{NSTACK}} = \{z, z1, z2, \ldots\}$

(representing the (disjoint) sets of variables for the three sorts)

$\text{emp}: \rightarrow \text{STACK}$

$\text{pop}: \text{NSTACK} \rightarrow \text{STACK}$

$\text{top}: \text{NSTACK} \rightarrow \text{E}$

$\text{push}: \text{E} \times \text{STACK} \rightarrow \text{NSTACK}$

$\text{pop}(\text{push}(e, y)) = y$

$\text{top}(\text{push}(e, y)) = e$

We have introduced the sort NSTACK to avoid the possibility of applying top or pop to emp, which would lead to the same problems as in the unsorted case. However, now we cannot push elements onto a non-empty stack. For example, $\text{push}(e1, \text{push}(e2, y))$ is ill-sorted because $\text{push}(e2, y)$ is of sort NSTACK, which lies outside the domain of push. Clearly we need a further refinement, in order to make terms of sort NSTACK be of sort STACK as well. Thus we arrive at an *order-sorted* signature.

3. ORDER-SORTED TERM REWRITING

Sorts are defined as before, except that now we specify a partial order $>_s$ between sorts (not to be confused with the term ordering $>$). If A and B are sorts and $A >_s B$, we say that B is a *subsort* of A. The reflexive closure of $>_s$ is denoted $\geqslant_s$. We denote $A >_s B$ diagrammatically by:

$$\begin{array}{c} A \\ | \\ B \end{array}$$

If B is a subsort of A, then every variable or constant of sort B is also a variable or constant of sort A. If a function f is defined on domain A, then f is defined on B and $f(b) = f(a)$, where b is a term of sort B and a is that term regarded as a term of sort A. This extends easily to the case where f takes more than one argument.

We make the further requirement that substitutions are *sort-preserving*, i.e. if v is a variable of sort A, t must be a term of some sort $B \leqslant_s A$ if t is to be substituted for v.

We mention in passing that whereas two terms in the unsorted and many-sorted cases can be unified in at most one way, terms in the order-sorted case can be unified in finitely many ways. The reader is referred to Ref. 4 for details.

Now we can specify operations on stacks as follows.

$S = \{\text{STACK, NSTACK, E}\}$

$\text{NSTACK} <_s \text{STACK}$
(NSTACK is a subsort of STACK)

$V_{\text{E}} = \{e, e1, e2, \ldots\}$

$V_{\text{STACK}} = \{y, y1, y2, \ldots\}$

$V_{\text{NSTACK}} = \{z, z1, z2, \ldots\}$

$\text{emp}: \rightarrow \text{STACK}$

$\text{pop}: \text{NSTACK} \rightarrow \text{STACK}$

$\text{top}: \text{NSTACK} \rightarrow \text{E}$

$\text{push}: \text{E} \times \text{STACK} \rightarrow \text{NSTACK}$

$\text{pop}(\text{push}(e, y)) = y$

$\text{top}(\text{push}(e, y)) = e$

It seems as though everything is in order, as z is a variable of a subsort of STACK, so it is also a variable of STACK.

However, we still cannot write

$\text{pop}(\text{pop}(\text{push}(e1, \text{push}(e2, \text{emp}))))$

for example. One solution to this is to introduce a *top* or *universal* sort U, with the intention that $A <_s U$ for every other sort A, and every function f is defined on U.[5] This has the result that even semantically meaningless terms become syntactically well-sorted, which is obviously

undesirable. Since the solution which we will propose to the problem of sort-decreasingness (see later) also solves the above problem, but without defining functions on the universal sort, we do not pursue the idea of the universal sort here.

4. PROBLEMS OF ORDER-SORTED REWRITING

We now consider some of the problems of order-sorted rewriting, and some current approaches to their solutions.

In general, these problems arise because of the considerable difference between the syntactic world of term rewriting and the semantic world of the underlying algebra, so we begin by defining the initial or Σ-algebra of an order-sorted term rewriting system. In this section we follow Ref. 6.

Definition

Let Σ be an order-sorted signature, with sorts S. Let E be our set of equations. Then the *initial algebra* $\mathscr{A}$ consists of *denotations* ξ^A and f^A for the sort and function symbols of Σ, respectively. Let

$\bar{s} = \{t \mid s = t$ is provable in E and t is a ground Σ-term$\}$

Then

$$\xi^A = \{\bar{t} \mid t \text{ is a ground } \Sigma\text{-term of sort } \xi\}.$$

$C_A = \bigcup\{\xi^A \mid \xi \in S\}$ is called the carrier of $\mathscr{A}$.

f^A is a mapping $D_f^A \to C_A$ whose domain D_f^A is a subset of $C_A^{|f|}$, where $|f|$ is the arity of f.

$$\text{If } f: \xi_1 \times \ldots \times \xi_n \to \xi \in \Sigma \quad \text{and} \quad a_i \in \xi_i \quad \text{for } i = 1, \ldots, n$$

Then $(\bar{a}_1, \ldots, \bar{a}_n) \in D_f^A$ and $f^A(\bar{a}_1, \ldots, \bar{a}_n) \in \xi^A$

Further, $f^A(\bar{a}_1, \ldots, \bar{a}_n) = \overline{f(a_1, \ldots, a_n)}$

if $a_1, \ldots, a_n$ are all ground terms.

We are particularly interested in initial algebras because they have:

no junk

for every $t \in \xi$ there is exactly one $\bar{t}$ in ξ^A

no confusion

$\bar{t} = \bar{s}$ in $\mathscr{A}$ iff $t = s$ is deducible from E.

4.1 Regularity

The first problem we find with our syntax for order-sorted rewriting is that it allows multiple function declarations while in the algebra every mapping has only one denotation. This results in problems such as the following example.

$S = \{S_1, S_2, S_3\}$

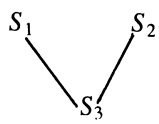
$S_1 >_s S_3$

$S_2 >_s S_3$

$f: S_1 \to S_1$

$f: S_2 \to S_2$

$a: \to S_3$



What is the sort of $f(a)$? $f(a)$ belongs to both S_1 and S_2 and to no lower sort. We have to conclude that $f(a)$ has no least sort.

Definition

A signature Σ is *regular* iff the subsort order $>_s$ is a partial order and every Σ-term t has a unique *least sort* $\mathrm{LS}(t)$ such that:

(i) if t is a Σ-term, t is a term of sort $\mathrm{LS}(t)$
(ii) if t is a Σ-term of sort ξ, $\mathrm{LS}(s) \leqslant_s \xi$.

Note that the least sort of a term t can be found effectively as it only depends on the domains and ranges of functions and the least sort of subterms of t.

In particular, in the example above we might define

$$f: S_3 \to S_3$$

to make the signature regular.

In order-sorted term rewriting we are only concerned with regular signatures, for the practical reason that unification in regular signatures is finitary, while in non-regular signatures it is infinitary (in general).

4.2 Void sorts

If a sort is void, i.e. contains no ground terms, we very quickly run into problems, as the following example from Ref. 7 shows.

The signature is:

$S = \{\mathrm{VOID}, \mathrm{BOOL}\}$

$\mathrm{TRUE}: \to \mathrm{BOOL}$

$\mathrm{FALSE}: \to \mathrm{BOOL}$

$V_{\mathrm{VOID}} = \{x, x1, x2, \ldots\}$

$f: \mathrm{VOID} \to \mathrm{BOOL}$

Let the rewrite rules be:

$f(x) \Rightarrow \mathrm{TRUE}$

$f(x) \Rightarrow \mathrm{FALSE}$

Because $f(x)$ rewrites to both TRUE and FALSE, we immediately deduce that TRUE = FALSE. However, this is not true in the initial algebra because $f(x)$ has no ground instantiation.

We shall adopt the simplest solution to this problem; we forbid void sorts. An alternative solution is given in Ref. 7.

4.3 Sort-preserving rewriting

A more serious problem is that order-sorted rewriting is not complete in the order-sorted case. Consider the following example from Ref. 6.

Signature:

$S = \{A, B\}$ with $B >_s A$

$a: \to A$

$a': \to A$

$b: \to B$

$f: A \to A$

Rules:

$a \Rightarrow b$

$a' \Rightarrow b$

In the initial algebra $\bar{a} = \bar{b} = \bar{a'}$

Therefore $\overline{f(a)} = \overline{f(a')}$

must also hold.

However, we cannot deduce that $f(a) = f(a')$ by rewriting because the term $f(b)$ is ill-sorted. The problem is that $\Rightarrow$ does not preserve monotonicity. Namely, $t \Rightarrow t'$ does not imply $f(t) \Rightarrow f(t')$ unless

$$\mathrm{LS}(t) \geqslant_s \mathrm{LS}(t')$$

Until recently, this restriction on term rewriting was accepted, and only signatures in which every rule $l \Rightarrow r$ was *sort-decreasing*, i.e. $\mathrm{LS}(l) \geqslant_s \mathrm{LS}(r)$, were considered. Recently, however, a number of groups have been working independently on this problem, including Gallier & Isakowitz,[8,9] Watson & Dick,[10] Ganzinger[11] and Duporcheel.[12]

Again the problem is caused by the difference between syntax and semantics. We have already introduced the syntactic measure of the least sort of a term t, $\mathrm{LS}(t)$, but in fact every term also has a semantic sort. In the initial algebra we define $\bar{t}$ for every ground Σ-term t because we want to do substitution of equals for equals – thus $\bar{t}$ is properly regarded as a congruence class. In order to substitute equals for equals within this congruence class, the terms which belong to this class must in some sense have the same sort, in order that a congruence class lies either entirely inside or entirely outside the domain of any given function.

So we define the *semantic sort* of t, $S_{\mathrm{SEM}}(t)$, to be

$$\cap \{\mathrm{LS}(s) \mid \bar{s} = \bar{t} \text{ in } \mathscr{A}\}$$

Now of course $\bar{t} = \bar{t}$, so we see that $S_{\mathrm{SEM}}(t) \leqslant_s \mathrm{LS}(t)$, for every t. Thus our notation becomes inappropriate, and we shall rename the syntactic (least) sort of t to be $S_{\mathrm{SYN}}(t)$.

4.3.1 Dynamic sorts

To avoid the restrictions of sort-decreasing rewrite rules we would much rather use the semantic sort S_{SEM} in rewriting. Then, rewriting a term never causes it to become ill-sorted. Unfortunately this is not possible, as the semantic sort of a term cannot be found effectively, in general. This is easily seen because for any t, $\bar{t}$ may represent infinitely many terms.

The idea of Ref. 10 is to approximate the semantic sort of a term using *dynamic sorting*. At stage $s+1$ in the Knuth–Bendix algorithm, after generating more equations by the critical pairs procedure, we calculate the *dynamic sort* of each term which occurs in the rules and equations, defined to be

$$S^{s+1}_{\mathrm{DYN}}(t) = \cap \{S^{s}_{\mathrm{DYN}}(t') \mid t = t' \text{ in } E'\}$$

where E' is the set of all rules generated up to stage s, $S^{0}_{\mathrm{DYN}}(t) = S_{\mathrm{SYN}}(t)$ for every t.

Note that $S^{s+1}_{\mathrm{DYN}} \leqslant_s S^{s}_{\mathrm{DYN}}(t)$ for every term t, stage s.

Now by the completeness of a fair Knuth–Bendix procedure which does not fail,

$$\mathrm{Lim}_{s \to \infty} S^{s}_{\mathrm{DYN}}(t) = S_{\mathrm{SEM}}(t)$$

The use of dynamic sorts ensures that at no stage do we rewrite a term to an ill-sorted term. The use of dynamic sorts also has consequences for unification, proof by contradiction and other facets of term-rewriting, the details of which are covered in Ref. 10, where a formalism (with inference rules) is given in the style of Ref. 13.

The method developed independently by Gallier & Isakowitz[8,9] is based on the same idea, except that they do not explicitly change the sort of a term during rewriting, but rather allow rewriting to (syntactically) ill-sorted terms, which they justify with a rigorous proof.

This method is fast and brutal compared to ours; the normal form of a term may be ill-sorted, for instance, and no new information about the semantic sort of a term is produced during the completion procedure. On the other hand their method is obviously much easier to implement, and quicker than the method of dynamic sorts.

Both methods remove the requirement that order-sorted rewrite rules must be sort-preserving.

5. CONCLUSION

We have attempted to introduce the essential aspects of order-sorted term rewriting in a well-motivated semi-formal fashion. We have highlighted some of the particular problems such an approach creates, and discussed possible solutions. The reader is referred to Ref. 6 for a thorough grounding in order-sorted rewriting.

REFERENCES

1. G. Huert and D. C. Oppen, Equations and rewrite rules – a survey. In Formal Languages: Perspectives and Problems, edited R. Book, Academic Press, London (1980).
2. A. J. J. Dick, An introduction to Knuth–Bendix completion. *Computer Journal* **34** (1), 2–15 (1991).
3. N. Dershowitz, Termination of rewriting. *J. of Symbolic Computation* **3**, 69–116 (1987).
4. R. J. Cunningham and A. J. J. Dick, Rewrite systems on a lattice of types. *Acta Informatica* **22**, 149–169 (1985).
5. J. A. Goguen and J. Meseguer, Order-sorted algebra, I. Partial and over loaded operators, errors and inheritance.
6. G. Smolka, W. Nutt, J. A. Goguen and J. Meseguer, Order-sorted equational completion. SEKI report SR-87-14. Universität Kaiserslautern, West Germany (1987), also in Proceedings of Colloquium on Resolution of Equations in Algebraic Structures, Austin, Texas (May 1987).
7. J. A. Goguen and J. Meseguer, Completeness of many-sorted equational logic. SIGPLAN Notices —16 (7), 24–32 (1981).
8. J. H. Gallier and T. Isakowitz, Rewriting in order-sorted equational logic. In *Logic Programming*, Proceedings of the Fifth International Conference and Symposium, edited R. A. Kowalski and K. A. Bowen, vol. 1, pp. 280–294, MIT Press (1988).
9. T. Isakowitz and J. H. Gallier, Congruence closure in order-sorted algebra. Technical Report, Computer and Information Sciences Department, University of Pennsylvania, Philadelphia, PA (1987).
10. P. Watson and A. J. J. Dick, Least sorts in order-sorted term rewriting. University of London, Royal Holloway and Bedford New College, Technical Report TR-CSD-606 (Jan. 1989).
11. H. Ganzinger, Order-sorted completion: the many-sorted way. Bericht, Nr. 274. Forschungsberichte des Fachbereichs Informatik der Universität Dortmund (1988).
12. L. Duporcheel, 'Typed Algebra (Back to the Future)'. Unpublished lecture notes, Alacatel Bell Telephone, 1989
13. L. Bachmair, Proof methods for equational theories. Ph.D. thesis, University of Illinois at Urbana-Champaign (1987).