A Church-Rosser Checker Tool for Conditional Order-Sorted Equational Maude Specifications

Francisco Durán[1] and José Meseguer[2]

[1] Universidad de Málaga, Spain
[2] University of Illinois at Urbana-Champaign, IL, USA

Abstract. The Church-Rosser property, together with termination, is essential for an equational specification to have good executability conditions, and also for having a complete agreement between the specification's initial algebra, mathematical semantics, and its operational semantics by rewriting. Checking this property for expressive specifications that are order-sorted, conditional with possibly extra variables in their condition, and whose equations can be applied modulo different combinations of associativity, commutativity and identity axioms is challenging. In particular, the resulting *conditional critical pairs* that cannot be joined have often an intuitively unsatisfiable condition or seem intuitively joinable, so that sophisticated tool support is needed to eliminate them. Another challenge is the presence of different combinations of associativity, commutativity and identity axioms, including the very challenging case of associativity without commutativity for which no finitary unification algorithms exist. In this paper we present the foundations and illustrate the design and use of a completely new version of the Maude Church-Rosser Checker tool that addresses all the abovementioned challenges and can deal effectively with complex conditional specifications modulo axioms.

1 Introduction

The goal of *executable* equational specification languages is to make *computable* the abstract data types specified in them by initial algebra semantics. In practice this is accomplished by using specifications that are Church-Rosser (or at least ground Church-Rosser) and terminating, so that the equations can be used from left to right as simplification rules; the result of evaluating an expression is then the canonical form that stands as a unique representative for the equivalence class of terms equal to the original term according to the equations. This approach is fully general; indeed, a well-known result of Bergstra and Tucker [5] shows that *any* computable algebraic data type can be specified by means of a finite set of ground-Church-Rosser and terminating equations, perhaps with the help of some auxiliary functions added to the original signature. For order-sorted specifications, being Church-Rosser and terminating means not only confluence—so that a unique normal form will be reached—but also a *descent* property ensuring that the normal form will have the least possible sort among those of all other equivalent terms.

P.C. Ölveczky (Ed.): WRLA 2010, LNCS 6381, pp. 69–85, 2010.

Therefore, for computational purposes it becomes very important to know whether a given specification is indeed (ground-)Church-Rosser and terminating. A nontrivial question is how to best support this with adequate tools. One can prove the operational termination of his/her (possibly conditional) Maude equational specification by using the MTT tool [14]. A thornier issue is what to do for establishing the ground-Church-Rosser property for a terminating specification. The problem is that a specification with an initial algebra semantics can be ground-Church-Rosser even though some of its critical pairs may not be joinable. That is, the specification can often be ground-Church-Rosser without being Church-Rosser for arbitrary terms with variables. In such a situation, blindly applying a completion procedure that is trying to establish the Church-Rosser property for arbitrary terms may be both quite hopeless—the procedure may diverge or get stuck because of unorientable rules, and even with success may return a specification that is quite different from the original one—and even unnecessary, if the specification was already ground-Church-Rosser. As we further explain in Section 3, several methods that do not alter the mathematical semantics of the original specification may allow us to either prove that the specification is ground Church-Rosser, or to transform it into an equivalent one that is Church-Rosser; typically with minimal changes.

Here, we present CRC, a Church-Rosser checker to check whether a (possibly conditional) order-sorted equational specification modulo equational axioms satisfies the Church-Rosser property. Our Church-Rosser checker tool is particularly well-suited for checking specifications with an initial algebra semantics that have already been proved terminating and now need to be checked to be Church-Rosser, or at least ground-Church-Rosser. Of course, the CRC tool can also be used to check the Church-Rosser property of conditional order-sorted specifications that do not have an initial algebra semantics, such as, for example, those specified in Maude functional theories [9]. Since, for the reasons mentioned above, user interaction will typically be quite essential, completion is not attempted. Instead, if the specification cannot be shown to be Church-Rosser by the tool, proof obligations are generated and are given back to the user as a guide in the attempt to establish the ground-Church-Rosser property. Since this property is in fact inductive, in some cases the Maude inductive theorem prover can be enlisted to prove some of these proof obligations. In other cases, the user may have to modify the original specification by carefully considering the information conveyed by the proof obligations. We give in Section 3 some methodological guidelines for the use of the tool, and illustrate the use of the tool with some examples (additional examples can be found in [17]).

The present CRC tool accepts order-sorted conditional specifications, where each of the operation symbols has either no equational attributes, or any combination of associativity/commutativity/identity. To deal with the various combinations of associativity, commutativity, and identity axioms we make use of different techniques now available. Maude 2.4 supports unification modulo commutativity and modulo associativity and commutativity [10]. Identity axioms and associativity without commutativity are handled using the variant-based

theory transformations presented in [15]. As pointed out in [15], the transformation cannot be used in general for the associativity without commutativity case because it does not have the finite variant property. However, the alternative semi-algorithm given there can be used in many practical situations in which the lefthand sides do have a finite set of variants. We refer the reader to [15] for further details, but the idea is that if for each operator in a module we cannot narrow on any equation's lefthand side using one of the two possible orientations of the associativity equation, then the only variant of the term is the term itself, and we can handle it just by adding the corresponding associativity equation. All this means that in practice we can often handle specifications whose operators can have *any* combination of associativity and/or commutativity and/or identity axioms. See Section 3.2 for an example.

Furthermore, it is assumed that such specifications do not contain any built-in function, do not use the `owise` attribute,[1] and that they have already been proved (operationally) terminating. The tool attempts to establish the ground-Church-Rosser property *modulo* the equational axioms specified for each of the operators by checking a sufficient condition. Therefore, the tool's output consists of a set of critical pairs and a set of membership assertions that must be shown, respectively, ground-joinable, and ground-rewritable to a term with the required sort.

The CRC tool has been implemented as an extension of Full Maude [16,13], as other tools in the Maude formal environment [11,20], and can be used on any Full Maude module satisfying the above restrictions, including structured modules, parameterized modules, etc.

The rest of the paper is structured as follows. Section 2 introduces the notion of Church-Rosser conditional order-sorted specifications modulo axioms. Section 3 presents some guidelines on how to use the tool and illustrates them with some examples. Section 4 concludes and presents some future work. Proofs of technical results are not included here for space reasons. They can be found in [19].

2 Church-Rosser (Conditional) Order-Sorted Specifications Modulo Axioms

In this section we introduce the notion of Church-Rosser order-sorted specification [23] and some standard notation on conditional rewriting (see, e.g., [29] for further details). We assume specifications of the form $\mathcal{R} = (\Sigma, A, R)$ where Σ is an A-preregular order-sorted signature, A is a set of equational axioms that are both regular and linear,[2] and R is an A-coherent set of (possibly conditional) rewrite

[1] In Maude, the `owise` attribute can be used to specify otherwise equations, i.e., equations that will be applied only if no other equation for that symbol can be applied.

[2] An equational axiom $u = v$ is regular if $\mathcal{V}ar(u) = \mathcal{V}ar(v)$, and linear if there are no repeated variables in either u or v.

rules. Let us start by introducing the notions of A-preregularity and A-coherence.

An order-sorted signature $(\Sigma, S, \leq)$ consists of a poset of sorts $(S, \leq)$ and an $S^* \times S$-indexed family of sets $\Sigma = \{\Sigma_{s_1 \ldots s_n, s}\}_{(s_1 \ldots s_n, s) \in S^* \times S}$ of function symbols. Given an S-sorted set $\mathcal{X} = \{\mathcal{X}_s \mid s \in S\}$ of *disjoint* sets of variables, the set $\mathcal{T}(\Sigma, \mathcal{X})_s$ denotes the Σ-algebra of Σ-terms of sort s with variables in $\mathcal{X}$. We denote $[t]_A$ the A-equivalence class of t.

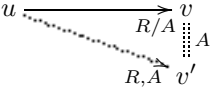
We call an order-sorted signature *A-preregular* if for each term w the set of sorts $\{s \in S \mid \exists w' \in [w]_A \text{ s.t. } w' \in \mathcal{T}(\Sigma, \mathcal{X})_s\}$ has a least upper bound, denoted $ls[w]_A$, which can be effectively computed.[3]

We denote by $\mathcal{P}(t)$ the set of positions of a term t, and by $t|_p$ *the subterm of t at position p* (with $p \in \mathcal{P}(t)$). A term t with its subterm $t|_p$ replaced by the term t' is denoted by $t[t']_p$.

Given a set of equational axioms A, a substitution σ is an *A-unifier* of t and t' if $t\sigma =_A t'\sigma$, and it is an *A-match* from t to t' if $t' =_A t\sigma$.

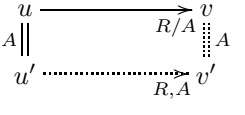
Given a rewrite theory $\mathcal{R}$ as above, we define the relation $\rightarrow_{R/A}$, either by the inference system of rewriting logic (see [8]), or by the usual inductive description: $\rightarrow_{R/A} = \bigcup_n \rightarrow_{R/A,n}$, where $\rightarrow_{R/A,0} = \emptyset$, and for each $n \in \mathbb{N}$, we have $\rightarrow_{R/A,n+1} = \rightarrow_{R/A,n} \cup \{(u, v) \mid u =_A l\sigma \rightarrow r\sigma =_A v \wedge l \rightarrow r \text{ if } \bigwedge_i u_i \rightarrow v_i \in R \wedge \forall i, u_i\sigma \rightarrow^*_{R/A,n} v_i\sigma\}$. In general, of course, given terms t and t' with sorts in the same connected component, the problem of whether $t \rightarrow_{R/A} t'$ holds is undecidable. For this reason, a much simpler relation $\rightarrow_{R,A}$ is defined, which becomes decidable if an A-matching algorithm exists. We define (see [30]) $\rightarrow_{R,A} = \bigcup_n \rightarrow_{R,A,n}$ where $\rightarrow_{R,A,0} = \emptyset$, and for each $n \in \mathbb{N}$ and any terms u, v with sorts in the same connected component the relation $u \rightarrow_{R,A,n+1} v$ holds if either $u \rightarrow_{R,A,n} v$, or there is a position p in u, a rule $l \rightarrow r$ if $\bigwedge_i u_i \rightarrow v_i$ in R, and a substitution σ such that $u|_p =_A l\sigma$, $v = u[r\sigma]_p$, and $\forall i, u_i\sigma \rightarrow^*_{R,A,n} w_i$ with $w_i =_A v_i\sigma$.

Of course, $\rightarrow_{R,A} \subseteq \rightarrow_{R/A}$, but the question is whether any $\rightarrow_{R/A}$-step can be simulated by a $\rightarrow_{R,A}$-step. We say that $\mathcal{R}$ satisfies this *A-completeness* property if for any u, v with sorts in the same connected component we have:



where here and in what follows dotted lines indicate existential quantification.

It is easy to check that A-completeness is equivalent to the following (strong) A-coherence property:



[3] The Maude system automatically checks the A-preregularity of a signature Σ for A any combination of associativity/commutativity/identity (see [9, Section 22.2.5]).

If a theory $\mathcal{R}$ is not coherent, we can try to make it so by completing the set of rules R to a set of rules $\widetilde{R}$ by a Knuth-Bendix-like completion procedure (see, e.g., [25,33,21]). For theories A that are combinations of associativity, commutativity, and identity axioms, we can make any specification A-coherent by using a very simple procedure (see, e.g., [17]).

The problem, then, is to check whether our specification $\mathcal{R}$, satisfying the above requirements, has the Church-Rosser property. As said above, for order-sorted specifications, being Church-Rosser and terminating means not only confluence, but also a descent property. After giving some auxiliary definitions, we introduce the notion of Church-Rosser conditional order-sorted specifications, and describe the sufficient conditions used by our tool to attempt checking the Church-Rosser property.

2.1 The Confluence Property

We say that a term t *A-overlaps* another term with distinct variables t' if there is a nonvariable subterm $t'|_p$ of t' for some position $p \in \mathcal{P}(t')$ such that the terms t and $t'|_p$ can be A-unified.

Definition 1. *Given an order-sorted equational specification $\mathcal{R} = (\Sigma, A, R)$, with Σ A-preregular and R A-coherent, and given conditional rewrite rules $l \to r$ if C and $l' \to r'$ if C' in R such that $(\mathcal{V}ar(l) \cup \mathcal{V}ar(r) \cup \mathcal{V}ar(C)) \cap (\mathcal{V}ar(l') \cup \mathcal{V}ar(r') \cup \mathcal{V}ar(C')) = \emptyset$ and $l|_p\sigma =_A l'\sigma$, for some nonvariable position $p \in \mathcal{P}(l)$ and A-unifier σ of $l|_p$ and l', then the triple*

$$C\,\sigma \wedge C'\sigma \Rightarrow l\sigma[r'\sigma]_p = r\sigma$$

is called a (conditional) critical pair.

In the uses we will make of the above definition we will always assume that the unification and the comparison for equality have been performed *modulo* A. Note also that the critical pairs accumulate the substitution instances of the conditions in the two rules, as in [7].

Given a rewrite theory $\mathcal{R} = (\Sigma, A, R)$, a critical pair $C \Rightarrow u = v$ is more general than another critical pair $C' \Rightarrow u' = v'$ if there exists a substitution σ such that $u\sigma =_A u'$, $v\sigma =_A v'$, and $C\,\sigma =_A C'$, where $C\,\sigma =_A C'$, with $C = \bigwedge_{i=1..n} u_i \to v_i$ and $C' = \bigwedge_{i=1..m} u'_i \to v'_i$, iff $n = m$ and $u_i\sigma =_A u'_i$ and $v_i\sigma =_A v'_i$ for every $i \in [1..n]$.

Then, given a specification $\mathcal{R}$, let $\mathrm{MCP}(\mathcal{R})$ denote the set of most general critical pairs between rules in $\mathcal{R}$ that, after simplifying both sides of the critical pair using the equational rules in $\mathcal{R}$, are not identical critical pairs modulo A of the form $C \Rightarrow t = t$. Under the assumption that the order-sorted equational specification $\mathcal{R}$ is operationally terminating, then, if $\mathrm{MCP}(\mathcal{R}) = \emptyset$, we are guaranteed that the specification $\mathcal{R}$ is *confluent* modulo A—in the obvious sense that if t can be rewritten modulo A to u and v using the rules in $\mathcal{R}$, then u and v can be rewritten modulo A to some w up to A-equality—and therefore, each term t has a unique canonical form modulo A $t{\downarrow_{\mathcal{R}}}$. Note that, due to the presence

of conditional equations, we can have $\mathrm{MCP}(\mathcal{R}) \neq \emptyset$ with $\mathcal{R}$ still confluent, but establishing that fact may require additional reasoning. More importantly for our purposes, even in the unconditional case we can have $\mathrm{MCP}(\mathcal{R}) \neq \emptyset$ with $\mathcal{R}$ *ground*-confluent, that is, confluent for all ground terms. Therefore, assuming termination, $\mathrm{MCP}(\mathcal{R}) = \emptyset$ will ensure the confluence and, a fortiori, the ground-confluence of $\mathcal{R}$, but this is only a sufficient condition.

2.2 Context-Joinability and Unfeasible Conditional Critical Pairs

From those conditional critical pairs which are not joinable, our tool can currently discard those that are either *context-joinable* or *unfeasible*, based on a result by Avenhaus and Loría-Sáenz [2], which we generalize here to the order-sorted modulo A case. Let us first introduce some notation.

A rule $l \to u_{n+1}$ *if* $\bigwedge_{i=1..n} u_i \to v_i$ is said to be *deterministic* if $\forall j \in [1..n+1], \mathcal{V}ar(u_j) \subseteq \mathcal{V}ar(l) \cup \bigcup_{k<j} \mathcal{V}ar(v_k)$. A conditional rewrite theory is *deterministic* if each of its rules is deterministic. Given a rewrite theory $\mathcal{R}$, a term t is called *strongly irreducible* with respect to R modulo A (or *strongly* R, A-*irreducible*) if $t\sigma$ is a normal form for every normalized substitution σ. A rewrite theory $\mathcal{R}$ is called *strongly deterministic* if for every rule $l \to r$ *if* $\bigwedge_{i=1..n} u_i \to v_i$ in R each v_i is strongly R, A-irreducible.

An admissible conditional order-sorted Maude functional specification can be transformed into an equivalent deterministic rewrite theory by a very simple procedure, in which equations are turned into rewrite rules and equational conditions (ordinary and matching equations) are turned into rewrites (see [17] for a detailed algorithm).

We denote by $\rhd$ the proper subterm relation. Then, given an order $\succ$, we denote by $\succ_{st} = (\succ \cup \rhd)^+$ the smallest ordering that contains $\succ$ and $\rhd$. A partial ordering $\succ$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is *well founded* if there is no infinite sequence $t_0 \succ t_1 \succ \ldots$. A partial ordering $\succ$ is *compatible with substitutions* if $u \succ u'$ implies $u\sigma \succ u'\sigma$ for any substitution σ. A partial ordering $\succ$ is *compatible with the term structure* if $u \succ u'$ implies $t[u]_p \succ t[u']_p$ for any term t and position p in t. A partial ordering $\succ$ is *compatible with the axioms* A if $v =_A u \succ u' =_A v'$ implies $v \succ v'$ for all terms u, u', v, and v' in $\mathcal{T}_\Sigma(\mathcal{X})$. A partial ordering $\succ$ is A-*compatible* if it is compatible with substitutions, compatible with the term structure, and compatible with the axioms A. Then, a *reduction ordering* is a partial ordering that is well founded and A-compatible.

A deterministic rewrite theory $\mathcal{R}$ is *quasi-reductive* w.r.t. a reduction ordering $\succ$ on $\mathcal{T}_\Sigma(\mathcal{X})$ if for every substitution σ, every rule $l \to u_{n+1}$ *if* $\bigwedge_{i=1\ldots n} u_i \to v_i$ in R, and every $i \in [1..n]$, $u_j\sigma \succeq v_j\sigma$ for every $j \in [1..i]$ implies $l\sigma \succ_{st} u_{i+1}\sigma$.

Let a *context* $C = \{u_1 \to v_1, \ldots, u_n \to v_n\}$ be a set of oriented equations. We denote by $\overline{C}$ the result of replacing each variable x in C by a new constant $\overline{x}$. And given a term t, $\overline{t}$ results from replacing each variable $x \in \mathcal{V}ar(C)$ by the new constant $\overline{x}$.

Definition 2. *Let $\mathcal{R} = (\Sigma, A, R)$ be a deterministic rewrite theory that is quasi-reductive w.r.t. an A-compatible well-founded relation $\succ$, and let $C \Rightarrow s = t$ be a*

critical pair resulting from $l_i \to r_i$ *if* C_i *for* $i = 1, 2$, *and* $\sigma \in Unif_A(l_1|_p, l_2)$. *We call* $C \Rightarrow s = t$ unfeasible *if there is some* $u \to v$ *in* C *such that* $\overline{u} \to_{R \cup \overline{C}, A} \overline{w}_1$, $\overline{u} \to_{R \cup \overline{C}, A} \overline{w}_2$, *and* $Unif_A(w_1, w_2) = \emptyset$ *and* w_1 *and* w_2 *are strongly irreducible with* R *modulo* A. *We call* $C \Rightarrow s = t$ context-joinable *if* $\overline{s}{\downarrow_{R \cup \overline{C}}}\overline{t}$.

Theorem 1. *Let* $\mathcal{R} = (\Sigma, A, R)$ *be a strongly deterministic rewrite theory that is quasi-reductive w.r.t. an* A*-compatible well-founded relation* $\succ$. *If every critical pair* $C \Rightarrow s = t$ *of* $\mathcal{R}$ *is either unfeasible or context-joinable, then* $\mathcal{R}$ *is confluent.*

Once all critical pairs are computed, the tool proceeds as follows. It first checks whether each conditional critical pair $C \Rightarrow s = t$ is *context joinable*:

(i) Variables in $C \Rightarrow s = t$ are added as new constants $\overline{X}$.
(ii) New *ground* rewrite rules $\overline{C}$ plus an equality operator eq with rules $eq(x, x) \to tt$ are added to the rules R. Call this theory $\hat{\mathcal{R}}_{\overline{C}}$.
(iii) In $\hat{\mathcal{R}}_{\overline{C}}$, we search $eq(\overline{s}, \overline{t}) \to^+ tt$ up to some predetermined depth (using the `search` command).

If the search is successful, then the conditional critical pair is context joinable. Otherwise, we then check whether $C \Rightarrow s = t$ is unfeasible as follows: For each condition $u_i \to v_i$, we perform in $\hat{\mathcal{R}}_{\overline{C}}$ the search $\overline{u_i} \to^! x : [k]$. Let $\overline{w_1} \ldots \overline{w_m}$ be the terms one obtains. If $m = 1$, then discard this term u_i and look for the next condition $u_{i+1} \to v_{i+1}$. *Otherwise*, try to find *two* different terms w_j, w_k such that (a) $Unif_A(w_j, w_k) = \emptyset$, and (b) w_j and w_k are *strongly irreducible* with $\mathcal{R}$ modulo A. If we succeed in finding a condition $u_i \to v_i$ for which associated w_j, w_k satisfy (a) and (b), then the conditional critical pair $C \Rightarrow s = t$ is *unfeasible*.[4]

2.3 The Descent Property

For an order-sorted specification it is not enough to be confluent for being Church-Rosser. The canonical form should also provide the most complete information possible about the sort of a term. This intuition is captured by our notion of Church-Rosser specifications.

Definition 3. *We call a confluent and terminating conditional order-sorted rewrite theory* $\mathcal{R} = (\Sigma, A, R)$ Church-Rosser *modulo* A *iff it additionally satisfies the following* descent *property: for each term* t *we have* $ls[t]_A \geq ls[t{\downarrow_{\mathcal{R}}}]_A$. *Similarly, we call a ground-confluent and terminating conditional order-sorted rewrite theory* $\mathcal{R} = (\Sigma, A, R)$ ground-Church-Rosser *modulo* A *iff for each ground term* t *we have* $ls[t]_A \geq ls[t{\downarrow_{\mathcal{R}}}]_A$.

Note that these notions are more general and flexible than the requirement of confluence and *sort-decreasingness* [27,22]. The issue is how to find checkable conditions for descent that, in addition to the computation of critical pairs, will ensure the Church-Rosser property. This leads us into the topic of specializations.

[4] Several optimizations, not currently available in the CRC tool are described in [17].

Given an order-sorted signature $(\Sigma, S, \leq)$, a sorted set of variables X can be viewed as a pair $(\hat{X}, \mu)$ where $\hat{X}$ is a set of variable names and μ is a sort assignment $\mu\colon \hat{X} \to S$. Thus, a *sort assignment* μ for X is a function mapping the names of the variables in $\hat{X}$ to their sorts. The ordering $\leq$ on S is extended to sort assignments by

$$\mu \leq \mu' \Leftrightarrow \forall x \in \hat{X}, \mu(x) \leq \mu'(x).$$

We then say that such a μ' *specializes* to μ, via the substitution

$$\rho : (x : \mu(x)) \leftarrow (x : \mu'(x))$$

called a *specialization* of $X = (\hat{X}, \mu')$ into $\rho(X) = (\hat{X}, \mu)$. Note that if the set of sorts is finite, or if each sort has only a finite number of sorts below it, then a finite sorted set of variables has a finite number of specializations.

The notion of specialization can be extended to axioms and rewrite rules. A specialization of an equation $(\forall X, l = r)$ is another equation $(\forall \rho(X), \rho(l) = \rho(r))$ where ρ is a specialization of X. A specialization of a rule $(\forall X, l \to r \text{ if } C)$ is a rule $(\forall \rho(X), \rho(l) \to \rho(r) \text{ if } \rho(C))$ where ρ is a specialization of X.

Thus, being *A-sort-decreasing* means that, for each rewrite rule $l \to r$ and for each specialization substitution ν, we have $ls[r\nu]_A \leq ls[l\nu]_A$. The checkable conditions that we have to add to the critical pairs to test for the descent property are called membership assertions.

Definition 4. *Let $\mathcal{R}$ be an order-sorted specification whose signature satisfies the assumptions already mentioned. Then, the set of* (conditional) membership assertions *for a conditional rule $t \to t'$ if C is defined as*

$$\{\ t'\theta : ls[t\theta]_A \text{ if } C\,\theta \ \mid \theta \text{ is a specialization of } \mathcal{V}ar(t) \text{ and } ls[t'\theta{\downarrow_{\mathcal{R}}}]_A \not\leq ls[t\theta]_A\ \}$$

A membership assertion $t : s$ *if* C is more general than another membership assertion $t' : s'$ *if* C' if there exists a substitution σ such that $t\sigma =_A t'$, $s \leq s'$, and $C\,\sigma =_A C'$.

Example 1. Given a specification of natural numbers and integers with the typical operations and definitions, and in particular a `square` operation defined as

```
op square : Int -> Nat .
eq square(I:Int) = I:Int * I:Int .
```

this equation gives rise to a membership assertion, because the least sort of the term `square(I:Int)` is `Nat`, but it is `Int` for the term in the righthand side. The proof obligation generated by the tool is

```
mb I:Int * I:Int : Nat .
```

This membership assertion must be proved inductively. That is, we have to treat it as the proof obligation that has to be satisfied in order to be able to assert that the specification is ground-decreasing. In this case, we have to prove that (for `I` and `J` variables of sort `Nat`) we have `INT` $\vdash_{\text{ind}}$ `(∀I)(∃J) I * I` $\rightarrow^*$ `J`, where `INT` here denotes the *rewrite theory* obtained from the original equational theory by turning each equation into a rewrite tule. This can be done using the constructor-based methods for proofs of ground reachability described in [32].

2.4 The Result of the Check

Let MMA($\mathcal{R}$) denote the set of most general membership assertions of all of the equations in the specification $\mathcal{R}$. Then, given a specification $\mathcal{R}$, the tool returns a tuple $\langle$ MCP($\mathcal{R}$), MMA($\mathcal{R}$) $\rangle$. A fundamental result underlying our tool is that the absence of critical pairs and of membership assertions in such an output is a sufficient condition for an operationally terminating specification $\mathcal{R}$ to be *Church-Rosser*.[5] In fact, for terminating unconditional specifications this check is a necessary and sufficient condition; however, for conditional specifications, the check is only a sufficient condition, because if the specification has conditional equations we can have unsatisfiable conditions in the critical pairs or in the membership assertions; that is, we can have $\langle$ MCP($\mathcal{R}$), MMA($\mathcal{R}$) $\rangle \neq \langle \emptyset, \emptyset \rangle$ with $\mathcal{R}$ still Church-Rosser. Furthermore, even if we assume that the specification is unconditional, since for specifications with an initial algebra semantics we only need to check that $\mathcal{R}$ is ground-Church-Rosser, we may sometimes have specifications that satisfy this property, but for which the tool returns a nonempty set of critical pairs or of membership assertions as proof obligations.

Of course, in other cases it may in fact be a matter of some error in the user's specification that the tool uncovers. In any case, the user has complete control on how to modify his/her specification, using the proof obligations in the output of the tool as a guide. In fact, as we explain below, several possibilities exist.

3 How to Use the Church-Rosser Checker

This section illustrates with examples the use of the Church-Rosser checker tool, and suggests some methods that—using the feedback provided by the tool—can help the user establish that his/her specification is ground-Church-Rosser.

We assume a context of use quite different from the usual context for *completing* an arbitrary equational theory. In our case we assume that the user has developed an *executable specification* of his/her intended system with an initial algebra semantics, and that this specification has already been *tested* with examples, so that the user is in fact confident that the specification is *ground-Church-Rosser*, and wants only to check this property with the tool.

[5] A detailed proof of this result will be presented elsewhere. In essence, it is a generalization of the result by Avenhaus and Loría-Sáenz [2, Theorem 4.1] to the order-sorted and modulo A case. For related results in membership equational logic see [7].

The tool can only *guarantee* success when the user's specification is unconditional and Church-Rosser and, furthermore, any associativity axiom in A for an operator has a corresponding commutativity axiom. In all other cases, the fact that the tool does not generate any proof obligations is only a *sufficient* condition, so that even when the CRC returns a collection of critical pairs and of membership assertions as proof obligations, the specification may be *ground* Church-Rosser, or for a conditional specification it may even be Church-Rosser.

An important methodological question is what to do, or not do, with these proof obligations. What should *not* be done is to let an automatic completion process add new equations to the user's specification in a mindless way. In some cases this is even impossible, because the critical pair in question cannot be oriented. In many cases it will certainly lead to a nonterminating process. In any case, it will modify the user's specification in ways that can make it difficult for the user to recognize the final result, if any, as intuitively equivalent to the original specification.

The feedback of the tool should instead be used as a guide for *careful analysis* of one's specification. As many of the examples we have studied indicate, by analyzing the critical pairs returned, the user can understand why they could not be joined. It may be a mistake that must be corrected. More often, however, it is not a matter of a mistake, but of a rule that is either *too general*—so that its very generality makes joining an associated critical pair impossible, because no more equations can apply to it—or *amenable to an equivalent formulation* that is unproblematic—for example, by reordering the parentheses for an operator that is ground-associative—or both. In any case, it is the user himself/herself who must study where the problem comes from, and how to fix it by correcting or modifying the specification. Interaction with the tool then provides a way of modifying the original specification and ascertaining whether the new version passes the test or is a good step towards that goal.

If the user's attempts to correct or modify the specification do not yet achieve a complete success, so that some proof obligations are left, *inductive* methods to discharge the remaining proof obligations may be used. Indeed, since the user's specification typically has an *initial* algebra semantics and the most common property of interest is checking that it is *ground* Church-Rosser, the proof obligations returned by the tool are *inductive* proof obligations. There are essentially two basic lines of approach, which may even be combined:

- The user may conjecture that adding a new equation $t = t'$ (or set of equations) to its specification T will make it Church-Rosser. If he can prove termination with the added equation(s) and the CRC does not generate any proof obligations for the extended specification, all is well. The only remaining issue is wether the new equation(s) have changed the module's initial algebra semantics. This can be checked by using a tool such as the Maude ITP (which does not require an equational specification to be Church-Rosser in order to perform sound inductive proofs) to verify that $T \vdash_{\text{ind}} t = t'$. A variant of this method when $t = t'$ is an associativity, or commutativity, or identity axiom, is to add it to T *not* as a simplification rule, but as an axiom.

– The other alternative is to reason inductively about the *ground joinablity* of the critical pairs, and also about the *inductive descent property* of the memberships, returned by the tool. The key point in both cases is that we should reason inductively *not* with the equational theory T (a critical pair is by construction an equational theorem in T), but with the rewrite theory $\overrightarrow{T}$ obtained by turning the equations of T into rewrite rules. An approach to inductive descent proofs with $\overrightarrow{T}$ has already been sketched in Section 2.3. For proving ground joinability, several proof methods, e.g., [31,3,26,28,4,1,6], can be used. In particular, for order-sorted specifications, constructor-based methods such as those described in [32] can be used.

A related unresolved methodological issue is what to do with *conditional* critical pairs, or conditional membership assertions, whose conditions are *unsatisfiable*. We currently discard critical pairs which the tool can show are *unfeasible* or *context-joinable*, but all remaining unjoinable pairs are left to the user. Perhaps a modular/hierarchical approach could be used, in conjunction with the inductive proof methods described above, to establish the unsatisfiability of such conditions and then discard the corresponding proof obligations.

We give in the following sections examples illustrating the use of the tool. The examples have been chosen trying to highlight those features not simultaneously supported by previous similar tools, namely, order-sortedness, conditional equations, and rewriting modulo axioms.

3.1 Hereditarily Finite Sets

The following module `HF-SETS` specifies hereditarily finite sets, that is, sets that are finite and, furthermore, their elements, the elements of those elements, and so on recursively, are all finite sets. It was developed by Ralf Sasse and José Meseguer and is inspired by the generalized sets module in Maude's prelude [9, Section 9.12.5]. It declares sorts `Set` and `Magma`, with `Set` a subsort of `Magma`. Terms of sort `Set` are generated by constructors `{}`, the empty set, and `{_}`, which makes a set out of a term of sort `Magma`. Magmas have an associative-commutative operator `_,_`. The commutative operator `_~_` checks whether two sets are equivalent. The membership relation $\in$ holding between two sets is here generalized by a predicate `_in_` holding between two magmas, and the containment relation $\subseteq$ is here modeled by a predicate `_<=_` holding between two sets.

```
(fmod HF-SETS is
   protecting BOOL-OPS .
   sorts Magma Set .
   subsort Set < Magma .
   op _`,_ : Magma Magma -> Magma [ctor assoc comm] .
   op `{_`} : Magma -> Set [ctor] .
   op `{`} : -> Set [ctor] .

   vars M M' N : Magma .                vars S S' : Set .

   eq [01]: M, M, M' = M, M' .          eq [02]: M, M = M .

   op _in_ : Magma Magma -> Bool .
```

```
  eq [03]: M in {} = false .
  eq [04]: {} in {{M}} = false .
  eq [05]: {} in {{}} = true .
  eq [06]: {} in {{}, M} = true .
  eq [07]: {} in {{M}, N} = {} in {N} .
  eq [08]: S in {S'} = S ~ S' .
 ceq [09]: S in {S' , M} = true if S ~ S' = true .
 ceq [10]: S in {S', M} = S in {M} if S ~ S' = false .
 ceq [11]: S in S', N = true if S in S' = true .
 ceq [12]: S in S', N = S in N if S in S' = false .
 ceq [13]: S, M in M' = M in M' if S in M' = true .
 ceq [14]: S, M in M' = false if S in M' = false .

  op _<=_ : Set Set -> Bool .
  eq [15]: {} <= S = true .
  eq [16]: {M} <= S = M in S .

  op _~_ : Set Set -> Bool .
  eq [17]: S ~ S' = (S <= S') and (S' <= S) .
endfm)
```

Notice the labeling of the equations. The critical pairs returned by the tool will use them to provide information about the equations they come from. Notice also the importation of the predefined module `BOOL-OPS`, where the sort `Bool` is defined with constants `true` and `false`, and boolean operations `_and_`, `_or_`, `_xor_`, `not_`, and `_implies_`. The operators `_and_`, `_or_`, and `_xor_` are declared associative and commutative.

The Church-Rosser check gives the following result:

```
Maude> (check Church-Rosser HF-SETS .)
Church-Rosser checking of HF-SETS
Checking solution:
The following critical pairs cannot be joined:
  ccp for 07 and 09
    S':Set <= {} = true if {} ~ S':Set = true .
  ccp for 07 and 10
    S':Set <= {} = false if {} ~ S':Set = false .
  ccp for 02 and 09
    S:Set <= S':Set and S':Set <= S:Set = true
    if S:Set ~ S':Set = true .
  ccp for 09 and 10
    true = S:Set <= #2:Set and #2:Set <= S:Set
    if S:Set ~ S':Set = false /\ S:Set ~ #2:Set = true .
  ccp for 10 and 09
    S:Set <= S':Set and S':Set <= S:Set = true
    if S:Set ~ S':Set = true /\ S:Set ~ #2:Set = false .
  ccp for 10 and 10
    S:Set <= S':Set and S':Set <= S:Set
    = S:Set <= #2:Set and #2:Set <= S:Set
    if S:Set ~ S':Set = false /\ S:Set ~ #2:Set = false .
The specification is sort-decreasing.
```

The tool generates 3725 critical pairs. Most of them are joinable, and therefore discarded. From the remaining 27 critical pairs, all of which are conditional, 21 are discarded because they can be proved either context-joinable or unfeasible. Let us take a look at some of these.

Let us consider the following context-joinable critical pair:

```
ccp for 16 and 11
  S:Set in (S':Set, N:Magma) = true
  if S:Set in S':Set = true .
```

If we add the condition of this critical pair as an equation with its variables `S` and `S'` turned into constants, of sort `Set`, `#S` and `#S'`, then, the terms `true` and `#S in (##1, #S')`, with `##1` a new constant of sort `Magma`, can be joined.

The following critical pair is discarded because it is unfeasible.

```
ccp for 11 and 12
  true = S:Set in N:Magma
  if S:Set in S':Set = false /\ S:Set in S':Set = true .
```

To prove unfeasibility we focus on the conditions. With the rules

```
#S in #S' = false
#S in #S' = true
```

the term `#S in #S'` can be rewritten both to `false` and `true`. Since they do not unify and are strongly irreducible, we conclude that the critical pair is unfeasible.

Most other critical pairs are discarded for similar reasons. The only ones left are those finally returned by the tool. These critical pairs are neither context-joinable nor unfeasible. However, we can introduce new equations, that should be inductively deducible from the specification, or replace the ones we have by alternative equations, in order to eliminate such critical pairs.

Let us start with the first critical pair in the CRC output. We may argue that if the set `S'` is such that the condition is satisfied, then the term `S' <= {}` should be reducible to `true`, and try to add equations to allow this rewrite. But more easily, we may observe that the critical pair comes from equations `07` and `09` at the top, because `09` is more general than necessary. Since a set is either of the form `{}` or `{M}`, and the `{}` case is covered by equations `06` and `07`, we can eliminate this critical pair by replacing equation `09` with

```
ceq [09']: {M} in {S, M'} = true if {M} ~ S = true .
```

A new execution of the check shows that the critical pair for equations `07` and `10` is no longer given. The critical pair for equations `07` and `10` suggests a similar change for equation `10`:

```
ceq [10']: {M} in {S, M'} = {M} in {M'} if {M} ~ S = false .
```

It is not enough, however. With these new two equations, the tool gives us now four conditional critical pairs. Given these critical pairs, we realize that equations `09'` and `10'` are still problematic. The simplest change is to replace these two equations by one unconditional equation covering the two cases:

```
eq [09-10]: {M} in {S, M'} = {M} ~ S or {M} in {M'} .
```

Replacing `09'` and `10'` by `09-10` now succeeds:

```
Maude> (check Church-Rosser HF-SETS-3 .)
Church-Rosser checking of HF-SETS-3
Checking solution:
All critical pairs have been joined.
The specification is locally-confluent.
The specification is sort-decreasing.
```

Therefore, once proved operationally terminating, we can conclude that the module `HF-SETS-3` is confluent.

3.2 Lists and Sets

Let us consider now the following specification of lists and sets.

```
(fmod LIST&SET is
   sorts MBool Nat List Set .
   subsorts Nat < List Set .
   ops true false : -> MBool .
   ops _and_ _or_ : MBool MBool -> MBool [assoc comm] .
   op 0 : -> Nat .
   op s_ : Nat -> Nat .
   op _;_ : List List -> List [assoc] .
   op null : -> Set .
   op __ : Set Set -> Set [assoc comm id: null] .
   op _in_ : Nat Set -> MBool .
   op _==_ : List List -> MBool [comm] .
   op list2set : List -> Set .

   var  B : MBool .            vars N M : Nat .
   vars L L' : List .          var  S : Set .

   eq [01]: N N = N .
   eq [02]: true and B = B .
   eq [03]: false and B = false .
   eq [04]: true or B = true .
   eq [05]: false or B = B .
   eq [06]: 0 == s N = false .
   eq [07]: s N == s M = N == M .
   eq [08]: N ; L == M = false .
   eq [09]: N ; L == M ; L' = (N == M) and L == L' .
   eq [10]: L == L = true .
   eq [11]: list2set(N) = N .
   eq [12]: list2set(N ; L) = N list2set(L) .
   eq [13]: N in null = false .
   eq [14]: N in M S = (N == M) or N in S .
endfm)
```

It has four sorts: `MBool`, `Nat`, `List`, and `Set`, with `Nat` included in both `List` and `Set` as a subsort. The terms of each sort are, respectively, Booleans, natural numbers (in Peano notation), lists of natural numbers, and finite sets of natural numbers. The rewrite rules in this module then define various functions such as `_and_` and `_or_`, a function `list2set` associating to each list its corresponding set, the set membership predicate `_in_`, and an equality predicate `_==_` on lists. Furthermore, the idempotency of set union is specified by the first equation. The operators `_and_` and `_or_` have been declared associative and commutative, the list concatenation operator `_;_` has been declared associative, the set union operator `__` has been declared associative, commutative and with `null` as its identity, and the `_==_` equality predicate has been declared commutative using the `comm` keyword. This module therefore illustrates how our tool can deal in principle with arbitrary combinations of associativity and/or commutativity and/or identity axioms, even though it may not succeed in some cases when some operators are associative but not commutative.

The tool gives us the following result.

```
Maude> (check Church-Rosser .)
Church-Rosser checking of LIST&SET
Checking solution:
The following critical pairs cannot be joined:
  cp for 01 and 14
    N:Nat == M:Nat = (N:Nat == M:Nat) or N:Nat == M:Nat .
  cp for 01 and 14
```

```
      (N:Nat == M:Nat) or N:Nat in #5:Set
      = (N:Nat == M:Nat) or (N:Nat == M:Nat) or N:Nat in #5:Set .
 The specification is sort-decreasing.
```

These critical pairs are completely harmless. They can in fact be removed by introducing an idempotency equation for the `_or_` operator.

```
(fmod LIST&SET-2 is pr LIST&SET .
   var  B : MBool .
   eq [15]: B or B = B .
 endfm)
```

The tool now tells us that the specification is locally confluent and sort-decreasing, and since it is terminating (see [15]), we can conclude that it is Church-Rosser.

As explained in Section 1, to handle this specification, we apply several transformations on the original module to remove identity attributes and associativity attributes that do not come with commutativity ones. We refer the interested reader to [17] for details on the use of this transformation in the CRC, and to [15] for a detailed description of the variant transformation used.

4 Conclusions and Future Work

We have presented the foundations, design, and use of the Maude CRC 3 tool, showing how it can deal effectively with complex equational specifications that are order-sorted, conditional (possibly with extra variables in their condition), and whose equations can be applied modulo different combinations of associativity, commutativity and identity axioms and are specified in Maude as functional modules or theories. Our tool attempts to prove such specifications Church-Rosser or ground Church-Rosser under the assumption of their operational termination. Besides the much greater generality of our tool when compared with its earlier versions or with other similar tools, two very useful new features are: (i) the capacity to discharge unjoinable critical pairs by proving them to be either unfeasible or context-joinable; and (ii) the capacity to deal in principle with any combination of associativity and/or commutativity and/or identity axioms. The CRC 3 tool together with its documentation is available at `http://maude.lcc.uma.es/CRChC`.

As future work, we would like to remove the current restrictions of the CRC tool, and to provide new methods to handle conditional critical pairs (or conditional membership assertions) whose conditions are unsatisfiable. The integration of the different tools in the Maude formal environment [11], namely the ITP [12], MTT [14], CRC, ChC [18], and SCC [24] tools, in a real formal tool environment is another important pending goal. It will make it easy for the CRC to interact with other tools that can be used to discharge some of its generated proof obligations.

Acknowledgements. F. Durán was supported by Spanish Research Projects TIN2008-03107 and P07-TIC-03184. J. Meseguer was partially supported by NSF Grants CCF-0905584, CNS-07-16038, CNS-09-04749, and CNS-08-34709.

References

1. Avenhaus, J., Hillenbrand, T., Löchner, B.: On using ground joinable equations in equational theorem proving. Journal of Symbolic Computation 36(1-2), 217–233 (2003)
2. Avenhaus, J., Loría-Sáenz, C.: On conditional rewrite systems with extra variables and deterministic logic programs. In: Pfenning, F. (ed.) LPAR 1994. LNCS, vol. 822, pp. 215–229. Springer, Heidelberg (1994)
3. Bachmair, L., Dershowitz, N., Plaisted, D.A.: Completion without failure. In: Kaci, A.H., Nivat, M. (eds.) Resolution of Equations in Algebraic Structures. Rewriting Techniques, vol. 2, pp. 1–30. Academic Press, New York (1989)
4. Becker, K.: Proving ground confluence and inductive validity in constructor based equational specifications. In: Gaudel, M.-C., Jouannaud, J.-P. (eds.) CAAP 1993, FASE 1993, and TAPSOFT 1993. LNCS, vol. 668, pp. 46–60. Springer, Heidelberg (1993)
5. Bergstra, J., Tucker, J.: Characterization of computable data types by means of a finite equational specification method. In: de Bakker, J.W., van Leeuwen, J. (eds.) Seventh Colloquium on Automata, Languages and Programming. LNCS, vol. 81, pp. 76–90. Springer, Heidelberg (1980)
6. Bouhoula, A.: Simultaneous checking of completeness and ground confluence for algebraic specifications. ACM Transactions on Computational Logic 10(3) (2009)
7. Bouhoula, A., Jouannaud, J.-P., Meseguer, J.: Specification and proof in membership equational logic. Theoretical Computer Science 236(1), 35–132 (2000)
8. Bruni, R., Meseguer, J.: Semantic foundations for generalized rewrite theories. Theoretical Computer Science 351(1), 286–414 (2006)
9. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C. (eds.): All About Maude - A High-Performance Logical Framework: How to Specify, Program, and Verify Systems in Rewriting Logic. LNCS, vol. 4350. Springer, Heidelberg (2007)
10. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: Maude 2.4 manual (November 2008), `http://maude.cs.uiuc.edu`
11. Clavel, M., Durán, F., Hendrix, J., Lucas, S., Meseguer, J., Ölveczky, P.: The Maude formal tool environment. In: Mossakowski, T., Montanari, U., Haveraaen, M. (eds.) CALCO 2007. LNCS, vol. 4624, pp. 173–178. Springer, Heidelberg (2007)
12. Clavel, M., Palomino, M., Riesco, A.: Introducing the ITP tool: a tutorial. Journal of Universal Computer Science 12(11), 1618–1650 (2006)
13. Durán, F.: A Reflective Module Algebra with Applications to the Maude Language. PhD thesis, Universidad de Málaga, Spain (June 1999), `http://maude.csl.sri.com/papers`
14. Durán, F., Lucas, S., Meseguer, J.: MTT: The Maude termination tool (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 313–319. Springer, Heidelberg (2008)
15. Durán, F., Lucas, S., Meseguer, J.: Termination modulo combinations of equational theories. In: Ghilardi, S., Sebastiani, R. (eds.) FroCoS 2009. LNCS, vol. 5749, pp. 246–262. Springer, Heidelberg (2009)
16. Durán, F., Meseguer, J.: Maude's module algebra. Science of Computer Programming 66(2), 125–153 (2007)
17. Durán, F., Meseguer, J.: CRC 3: A Church-Rosser checker tool for conditional order-sorted equational Maude specifications (2009), `http://maude.lcc.uma.es/CRChC`

18. Durán, F., Meseguer, J.: A Maude coherence checker tool for conditional order-sorted rewrite theories. In: Olveczky, P.C. (ed.) WRLA 2010. LNCS, vol. 6381, pp. 86–103. Springer, Heidelberg (2010)
19. Durán, F., Meseguer, J.: A Church-Rosser Checker Tool for Conditional Order-Sorted Equational Maude Specifications. In: Ölveczky, P.C. (ed.) 8th Intl. Workshop on Rewriting Logic and its Applications (2010)
20. Durán, F., Ölveczky, P.C.: A guide to extending Full Maude illustrated with the implementation of Real-Time Maude. In: Roşu, G. (ed.) Proceedings 7th International Workshop on Rewriting Logic and its Applications (WRLA 2008). Electronic Notes in Theoretical Computer Science. Elsevier, Amsterdam (2008)
21. Giesl, J., Kapur, D.: Dependency pairs for equational rewriting. In: Middeldorp, A. (ed.) RTA 2001. LNCS, vol. 2051, pp. 93–108. Springer, Heidelberg (2001)
22. Gnaedig, I., Kirchner, C., Kirchner, H.: Equational completion in order-sorted algebras. Theoretical Computer Science 72, 169–202 (1990)
23. Goguen, J., Meseguer, J.: Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. Theoretical Computer Science 105, 217–273 (1992)
24. Hendrix, J., Meseguer, J., Ohsaki, H.: A sufficient completeness checker for linear order-sorted specifications modulo axioms. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 151–155. Springer, Heidelberg (2006)
25. Jouannaud, J.-P., Kirchner, H.: Completion of a set of rules modulo a set of equations. SIAM Journal of Computing 15(4), 1155–1194 (1986)
26. Kapur, D., Narendran, P., Otto, F.: On ground-confluence of term rewriting systems. Information and Computation 86(1), 14–31 (1990)
27. Kirchner, C., Kirchner, H., Meseguer, J.: Operational semantics of OBJ3. In: Lepistö, T., Salomaa, A. (eds.) ICALP 1988. LNCS, vol. 317, pp. 287–301. Springer, Heidelberg (1988)
28. Martin, U., Nipkow, T.: Ordered rewriting and confluence. In: Stickel, M.E. (ed.) CADE 1990. LNCS, vol. 449, pp. 366–380. Springer, Heidelberg (1990)
29. Ohlebusch, E.: Advanced Topics in Term Rewriting. Springer, Heidelberg (2002)
30. Peterson, G., Stickel, M.: Complete sets of reductions for some equational theories. Journal of ACM 28(2), 233–264 (1981)
31. Plaisted, D.: Semantic confluence tests and completion methods. Information and Control 65, 182–215 (1985)
32. Rocha, C., Meseguer, J.: Constructors, sufficient completeness, deadlock states of rewrite theories. Technical Report 2010-05-1, CS Dept., University of Illinois at Urbana-Champaign (May 2010), `http://ideals.illinois.edu`
33. Viry, P.: Equational rules for rewriting logic. Theoretical Computer Science 285(2), 487–517 (2002)