

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

Francisco Durán¹ and José Meseguer²

¹ Universidad de Málaga, Spain.

² University of Illinois at Urbana-Champaign, IL, USA.

Abstract. The (ground) 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 above-mentioned 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 *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 [2] shows that *any* computable algebraic data type can be specified by means of a finite set of ground-Church-Rosser and operationally 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, namely, that the normal form will have the least possible sort among those of all other equivalent terms.

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 [8]. 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 often 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 diverges or gets 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.

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 ground-Church-Rosser, although it can of course 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 [5]. 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 ground-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 in fact 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 [10]); we also explain there that the issue of finding general inductive proof techniques for proving the ground-Church-Rosser property is at the moment an interesting open problem.

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 associativity and commutativity [6]. Identity axioms and associativity without commutativity are handled using the variant-based theory transformations presented in [9]. As pointed out in [9], the transformation cannot be used in practice 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. We refer the reader to [9] for further details, but the idea is that if for each operator in a module we cannot narrow on any equation 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, 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 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 directions on how to use the tool and illustrates it with some examples. Section 4 concludes and presents some future work.

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

In this section we introduce the notion of Church-Rosser order-sorted specification [13]. We assume specifications of the form $\mathcal{R} = (\Sigma, R \cup A)$ where Σ is an A -preregular order-sorted signature and R is A -coherent. Let us start by introducing the notions of A -preregularity and A -coherence, where A is a set of equational axioms that are both regular and linear.³

An order-sorted signature (Σ, S, \leq) consists of a poset of sorts (S, \leq) and an $S^* \times S$ -indexed family of sets $\Sigma = \{\Sigma_{s_1 \dots s_n, s}\}_{(s_1 \dots s_n, s) \in S^* \times S}$ 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 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.⁴

We denote by \triangleright the proper subterm relation. Then, given an order \succ , we denote by $\succ_{st} = (\succ \cup \triangleright)^+$ the smallest ordering that contains \succ and \triangleright . We denote $\mathcal{P}(t)$ the set of positions of a term t , and $t|_p$ the subterm of t at position p

³ An equational axiom $u = v$ is regular if $\text{vars}(u) = \text{vars}(v)$, and linear if there are no repeated variables in either u or v .

⁴ The Maude system automatically checks the A -preregularity of a signature Σ for A any combination of associativity/commutativity/identity (see [5, Chapter 22.2.5]).

(with $p \in \mathcal{P}(t)$). A term t with its subterm $t|_p$ replaced by the term t' is denoted $t[t']_p$.

Given a set of 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 [4]), 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 [16]) $\rightarrow_{R,A} = \bigcup_n \rightarrow_{R,A,n}$ 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:

$$\begin{array}{ccc} u & \xrightarrow{\quad} & v \\ & \searrow & \downarrow \text{R/A} \\ & & v' \\ & \nearrow & \downarrow \text{A} \\ & & v' \\ & \searrow & \downarrow \text{R,A} \\ & & v' \end{array}$$

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:

$$\begin{array}{ccc} u & \xrightarrow{\quad} & v \\ A \parallel & & \downarrow \text{R/A} \\ u' & \xrightarrow{\quad} & v' \\ & & \downarrow \text{A} \\ & & v' \end{array}$$

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 \tilde{R} by a Knuth-Bendix-like completion procedure (see, e.g., [14, 17, 11]). 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., [10]).

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 $\mathcal{R} = (\Sigma, R \cup A)$, with Σ A -preregular and R A -coherent, and given conditional rewrite rules $l \rightarrow r$ **if** C and $l' \rightarrow r'$ **if** C' in R such that $(\text{vars}(l) \cup \text{vars}(r) \cup \text{vars}(C)) \cap (\text{vars}(l') \cup \text{vars}(r') \cup \text{vars}(C')) = \emptyset$ and $l|_p \sigma =_A l' \sigma$, for some nonvariable position $p \in \mathcal{P}(l)$ and A -unifier σ , 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 [3].

Given a specification $\mathcal{R} = (\Sigma, R \cup A)$, a critical pair $C \Rightarrow t = t'$ is more general than another critical pair $C' \Rightarrow u = u'$ if there exists a substitution σ such that $t \sigma =_A u$, $t' \sigma =_A u'$, and $C \sigma =_A C'$.

Then, given a specification \mathcal{R} , let $\text{MCP}(\mathcal{R})$ denote the set of most general critical pairs between rules in \mathcal{R} that, after simplifying both sides of the critical pair using the equations 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 $\text{MCP}(\mathcal{R}) = \emptyset$, we are guaranteed that the specification \mathcal{R} is *confluent*—in the standard 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 $\text{MCP}(\mathcal{R}) \neq \emptyset$ with \mathcal{R} still confluent, because all the conditions in the critical pairs may be unsatisfiable or joinable for all instantiations satisfying the condition, but establishing that fact may require additional reasoning. More importantly for our purposes, even in the unconditional case we can have $\text{MCP}(\mathcal{R}) \neq \emptyset$ with \mathcal{R} *ground-confluent*, that is, confluent for all ground terms. Therefore, assuming termination, $\text{MCP}(\mathcal{R}) = \emptyset$ 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 satisfiable, the tool can currently discard those that are *context-joinable* or *unfeasible*, based on a result by Avenhaus and Loría-Sáenz [1], which we generalize here to the order-sorted modulo A case. Let us first introduce some notation.

A conditional order-sorted Maude functional specification can be transformed into an equivalent DTRS 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 [10] for a detailed algorithm).

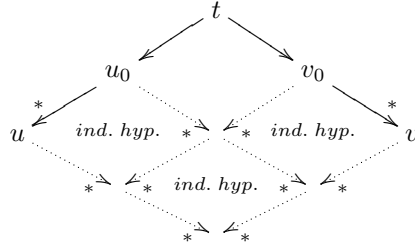
Let a *context* $C = \{u_1 \rightarrow v_1, \dots, u_n \rightarrow v_n\}$ be a set of oriented equations. We denote by \bar{C} the result of replacing each variable x in C by a new constant \bar{x} . And given a term t , \bar{t} results from replacing each variable $x \in \text{vars}(C)$ by the constant \bar{x} .

Definition 2. Let $\mathcal{R} = (\Sigma, R \cup A)$ be a DTRS that is quasi-reductive w.r.t. an A -compatible order well-founded relation \succ , and let $C \Rightarrow s = t$ be a critical pair resulting from $l_i \rightarrow r_i$ if C_i for $i = 1, 2$, and $\sigma \in \text{Unif}_A(l_1|_p, l_2)$. We call $C \Rightarrow s = t$ unfeasible if there is some $u \rightarrow v$ in C such that $\bar{u} \rightarrow_{R \cup \bar{C}, A} \bar{w}_1$, $\bar{u} \rightarrow_{R \cup \bar{C}, A} \bar{w}_2$, and $\text{Unif}_A(w_1, w_2) = \emptyset$ and w_1 and w_2 are strongly irreducible with R modulo A . We call $C \Rightarrow s = t$ context-joinable if $\bar{s} \downarrow_{R \cup \bar{C}} \bar{t}$.

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

Proof. We prove the result by noetherian induction on $\succ_{st} = (\succ \cup \triangleright)^+$. Suppose that we have the critical peak $u \xleftarrow{*} t \xrightarrow{*} v$. All nontrivial cases to consider are of the form $u \xleftarrow{*} u_0 \leftarrow t \rightarrow v_0 \xrightarrow{*} v$ and such that the redexes for u_0 and v_0 overlap. If they do not overlap at the top of t for one of them, then the induction hypothesis can be applied, since there will be a smaller subterm for which we have the result.

Also, since $t \succ u_0$ and $t \succ v_0$, the confluence for u, v will follow from that for u_0, v_0 :



Therefore, we may reduce to an *instance* of a critical pair $C \Rightarrow s = t$ by a substitution σ such that σC holds. Now, if $C \Rightarrow s = t$ is context-joinable, the result follows from [1, Lemma 4.2].

So we have only left the case when $C \Rightarrow s = t$ is unfeasible and αC holds. This means that $\alpha\sigma C_1$ and $\alpha\sigma C_2$ hold for the conditions of $l_1 \rightarrow r_1$ if C_1 and $l_2 \rightarrow r_2$ if C_2 . Therefore, since $\rightarrow_{R, A} \subseteq \succ$, and for each $u_i \rightarrow v_i$ in C_1 , $u'_j \rightarrow v'_j$ in C_2 we have $\alpha\sigma u_i \xrightarrow{*}_{R, A} \alpha\sigma v_i$ and $\alpha\sigma u'_j \xrightarrow{*}_{R, A} \alpha\sigma v'_j$, we have that for each $u \rightarrow v$ in C we have $\alpha u \xrightarrow{*}_{R, A} \alpha v$, and therefore, by Definition 2, we have $\alpha\sigma u_i \text{ st} \prec \alpha\sigma l_1 \succeq_{st} \alpha\sigma l_2 \succ \alpha\sigma u'_j$.

We may assume that we have $\sigma\bar{u}_i \rightarrow_{R \cup \bar{C}, A}^* \bar{w}_1$ and $\sigma\bar{u}_i \rightarrow_{R \cup \bar{C}, A}^* \bar{w}_2$ with $Unif_A(w_1, w_2) = \emptyset$ and w_1 and w_2 strongly irreducible. Then, since α is a solution of C , by [1, Lemma 4.2], we have $\alpha\sigma u_i \rightarrow_{R, A}^* \alpha w_1 \rightarrow_{R, A}^* \alpha\downarrow(w_1)$ and $\alpha\sigma u_i \rightarrow_{R, A}^* \alpha w_2 \rightarrow_{R, A}^* \alpha\downarrow(w_2)$, where $\alpha\downarrow$ is just *some canonical form* of the substitution α , which exists by the termination assumption regardless of whether R is confluent or not. But since w_1, w_2 are strongly irreducible, then $\alpha\downarrow(w_1)$ and $\alpha\downarrow(w_2)$ are in canonical form (modulo A) and are *different* (by $Unif_A(w_1, w_2) = \emptyset$). But since $\alpha\sigma l_1 \succ_{st} \alpha\sigma u_1$, the confluence induction hypothesis applies to $\alpha\sigma u_1$, and therefore it is confluent, which is in contradiction with $\alpha\downarrow(w_1) \neq_A \alpha\downarrow(w_2)$.

The case where the unfeasibility problem arises in $\alpha\sigma u'_j$ is entirely similar, since $\alpha\sigma l_1 \succeq_{st} \alpha\sigma l_2 \succ_{st} \alpha\sigma u'_j$. \square

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 \bar{X} .
- (ii) New *ground* rewrite rules \bar{C} plus an equality operator eq with rules $eq(x, x) \rightarrow tt$ are added to the rules R . Call this theory $\hat{\mathcal{R}}_{\bar{C}}$.
- (iii) In $\hat{\mathcal{R}}_{\bar{C}}$, we search $eq(\bar{s}, \bar{t}) \Rightarrow^+ tt$ 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 $u_i \rightarrow v_i$, we perform in $\hat{\mathcal{R}}_{\bar{C}}$ the search $\bar{u}_i \Rightarrow^! x : [k]$. Let $\bar{w}_1 \dots \bar{w}_m$ be the terms one obtains. If $m = 1$, then discard this term u_i and look for the next condition $u_{i+1} \rightarrow v_{i+1}$. *Otherwise*, try to find *two* different terms w_i, w_j such that (a) $Unif_A(w_1, w_2) = \emptyset$, and (b) w_i and w_j are *strongly irreducible* with \mathcal{R} modulo A . If we succeed in finding a condition $u_{i+1} \rightarrow v_{i+1}$ for which associated w_i, w_j satisfy (a) and (b), then the conditional critical pair $C \Rightarrow s = t$ is *unfeasible*.⁵

2.3 The Descent Property

For an order-sorted specification it is not enough to be confluent. 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 specification $\mathcal{R} = (\Sigma, R \cup A)$ 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 equational specification $\mathcal{R} = (\Sigma, R \cup A)$ ground-Church-Rosser modulo A iff for each ground term t we have $ls[t]_A \geq ls[t\downarrow_{\mathcal{R}}]_A$.*

⁵ Several optimizations, not currently available in the CRC tool are described in [10].

Note that these notions are more general and flexible than the requirement of confluence and *sort-decreasingness* [15, 12]. 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.

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

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

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

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

called a *specialization* of $X = (\bar{X}, \mu')$ into $\rho(X) = (\bar{X}, \mu)$. 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 $(\forall X, l \rightarrow r \text{ if } C)$ is a rule $(\forall \rho(X), \rho(l) \rightarrow \rho(r) \text{ if } \rho(C))$ where ρ is a specialization of X .

Thus, being *A-sort-decreasing* means that, for each rewrite rule $l \rightarrow r$ 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 equation $t = t' \text{ if } C$ is defined as*

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

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

2.4 The Result of the Check

Let $\text{MMA}(\mathcal{R})$ 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 $\langle \text{MCP}(\mathcal{R}), \text{MMA}(\mathcal{R}) \rangle$. 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*.⁶ In fact, for terminating unconditional specifications this check

⁶ A detailed proof of this result will be presented elsewhere. In essence, it is a generalization of the result by Avenhaus and Loría-Sáenz [1, Theorem 4.1] to the order-sorted and modulo A case. For related results in membership equational logic see [3].

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 $\langle \text{MCP}(\mathcal{R}), \text{MMA}(\mathcal{R}) \rangle \neq \langle \emptyset, \emptyset \rangle$ 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, several possibilities exist. The user can prove inductively that the critical pairs are ground-joinable, and that the membership assertions are ground-rewritable to a term with the required sort; however, at present the methods available for such proofs are quite limited. Alternatively, the user can instead modify the specification with the purpose of either correcting a found mistake, or modifying the already correct specification into a variant that the tool will hopefully certify Church-Rosser when resubmitted.

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.

Of course, the tool can only guarantee success when the user's specification is unconditional and Church-Rosser, and not just ground-Church-Rosser. That is, not generating any proof obligations is only a *sufficient* condition. But in some cases of interest—particularly for specifications with nontrivial sort—the specification may be *ground* Church-Rosser, but not Church-Rosser, so that a collection of critical pairs and of membership assertions will be returned by the tool as proof obligations.

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 thought* about 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 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.

Since the user’s specification usually 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. Therefore, after having introduced some modifications that may already eliminate some of the critical pairs and membership assertions generated by the tool, the user may be left with proof obligations for which the best approach is not any further modification of the specification, but, instead, an inductive proof. Provided that no equational lemmas are used in the proof, the inductive theorem prover presented in [7] can be used for this purpose, and in fact can succeed in proving these proof obligations. See [10] for some examples.

Inductive proof of the joinability of critical pairs is a thornier issue, for which we lack at present good methods. The problem is that, if we have an unconditional critical pair $t = t'$ generated by the tool for a module M , this already shows that $M \vdash t = t'$ and, a fortiori, that $M \vdash_{\text{ind}} t = t'$. But this is useless for ensuring joinability. What must instead be proved inductively is $\mathbf{M} \vdash_{\text{ind}} t \downarrow t'$, where \mathbf{M} is the *rewrite theory* associated to the (oriented) equational theory M . But this requires inductive methods for rewriting logic that, as far as we know, are much less developed.

A related unresolved methodological issue is what to do with *conditional* critical pairs or 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. If we already *knew* that the specification was Church-Rosser, we could use standard induction methods to discard them. But this is precisely what we need to prove. It is quite possible that a modular/hierarchical approach could be used, in conjunction with new inductive proof methods, to establish the unsatisfiability of such conditions and then discard the corresponding proof obligations. But such an approach has still to be developed.

We give in the following sections examples illustrating the use of the tool. The examples were chosen trying to highlight those features not simultaneously

supported by previous similar tools, namely, order-sortedness, conditional equations, and rewriting modulo.

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 [5, 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 two overloaded associative-commutative operators `_ , _`, one of them a constructor taking a set and a magma, and the other one taking two magmas. The commutative operator `_equiv_` checks whether two magmas are equivalent. Then, there are operations `_in_`, to check whether a magma is in another magma, and `_<=_` to check whether a magma is contained in another magma. A magma is contained in another magma if all its sets are included; a non-empty set is a subset of a magma if all its element are in the other magma.

```
(fmod HF-SETS is
  sorts Magma Set .
  subsort Set < Magma .
  op '_ , _' : Set Magma -> Magma [ctor assoc comm] .
  op '_ , _' : Magma Magma -> Magma [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]: {M} in {M'} = M in M' .
  ceq [09]: S in S', N = true if S in S' = true .
  ceq [10]: S in S', N = S in N if S in S' = false .
  ceq [11]: S, M in M' = M in M' if S in M' = true .
  ceq [12]: S, M in M' = false if S in M' = false .

  op _equiv_ : Magma Magma -> Bool [comm] .
  eq [13]: M equiv M = true .
  eq [14]: {} equiv {M} = false .
  eq [15]: {M} equiv {M'} = M equiv M' .
  eq [16]: S equiv S, M' = S equiv M' .
  ceq [17]: S equiv M = false if S in {M} = false .
  ceq [18]: S, M equiv M' = false if S in {M'} = false .
  ceq [19]: M equiv M' = false if {M} in {{M'}} = false .

  op _<=_ : Magma Magma -> Bool .
  eq [20]: {} <= M = true .
  ceq [21]: {M} <= M' = true if M in M' = true .
  ceq [22]: {M} <= M' = false if M in M' = false .
  ceq [23]: S, M <= M' = M <= M' if S <= M' = true .
  ceq [24]: S, M <= M' = false if S <= M' = false .
endfm)
```

Notice the labeling of the equations. The critical pairs returned by the tool will use them to provide information on the equations they come from.

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 13 and 17
    true = false
    if S:Set in {S:Set} = false .
  ccp for 13 and 18
    true = false
    if S:Set in {M:Magma, S:Set} = false .
The specification is sort-decreasing.
```

The tool generates 1241 critical pairs. Many of them are joinable, and therefore discarded. From the remaining 28 critical pairs, all of which are conditional, 26 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 critical pair:

```
ccp for 01 and 09
  S:Set in (#1:Magma, S':Set) = true
  if S:Set in S':Set = true .
```

It is context joinable. 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 `#S in (##1, #S')` and `true`, with `##1` a new constant of sort `Magma`, can be joined.

The following critical pair is discarded because it is unfeasible.

```
ccp 09 and 10
  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 can conclude that the critical pair is unfeasible.

The rest of the 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, and at present we do not have methods to eliminate them. But we can still introduce new equations, that should be inductively deducible from the specification, or replace the ones we have, to eliminate the critical pairs.

Let us start with the first critical pair. Given its condition, we may think about adding the following equation 25.

```
(fmod HF-SETS-EXTRAS-1 is pr HF-SETS .
  var S : Set .
  eq [25]: S in {S} = true .
endfm)
```

The tool gives us two critical pairs again, although in this case one of them is unconditional.

```
Maude> (check Church-Rosser HF-SETS-EXTRAS-1 .)

Church-Rosser checking of HF-SETS-EXTRAS-1
Checking solution:
The following critical pairs cannot be joined:
cp for 08 and 25
  #1:Magma in {#1:Magma} = true .
ccp for 13 and 18
  true = false
  if S:Set in {M:Magma, S:Set} = false .
The specification is sort-decreasing.
```

By iterating this process several times, we arrive at the following module:

```
(fmod HF-SETS-EXTRAS-6 is
pr HF-SETS .
vars M M' M'' : Magma .
eq [25']: M in {M} = true .
eq [26]: M in {M, M'} = true .
eq [27]: M in {M}, M' = true .
eq [28]: M in {M, M'}, M'' = true .
endfm)
```

The tool does not give any critical pair for it:

```
Maude> (check Church-Rosser HF-SETS-EXTRAS-6 .)

Church-Rosser checking of HF-SETS-EXTRAS-6
Checking solution:
All critical pairs have been joined.
The specification is locally-confluent.
The specification is sort-decreasing.
```

Once proved operationally terminating, we can conclude that the module HF-SETS-EXTRAS-6 is confluent. The interested reader can find the complete process in [10].

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 [9]), 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 [10] for details on the use of this transformation in the CRC, and to [9] for a detailed description of the variant transformation used.

4 Conclusions and Future Work

We have presented the foundations, design, and use of the new Maude CRC tool, showing how it can deal effectively with complex equational 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 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 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 tool together with its documentation is available at <http://maude.lcc.uma.es/CRChC>. After some further optimizations and a more complete documentation we plan to make it publicly available in the Maude web page before WRLA 2010.

References

1. J. Avenhaus and C. Loria-Sáenz. On conditional rewrite systems with extra variables and deterministic logic programs. In F. Pfenning, editor, *Logic Programming and Automated Reasoning, 5th International Conference, LPAR'94, Kiev, Ukraine, July 16-22, 1994, Proceedings*, volume 822 of *Lecture Notes in Computer Science*, pages 215–229. Springer, 1994.
2. J. Bergstra and J. Tucker. Characterization of computable data types by means of a finite equational specification method. In J. W. de Bakker and J. van Leeuwen, editors, *Seventh Colloquium on Automata, Languages and Programming*, volume 81 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 1980.
3. A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoretical Computer Science*, 236(1):35–132, 2000.
4. R. Bruni and J. Meseguer. Semantic foundations for generalized rewrite theories. *Theoretical Computer Science*, 351(1):286–414, 2006.
5. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude - A High-Performance Logical Framework: How to Specify, Program, and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.
6. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. Maude 2.4 manual. Available in <http://maude.cs.uiuc.edu>, November 2008.
7. M. Clavel, F. Durán, S. Eker, and J. Meseguer. Building equational proving tools by reflection in rewriting logic. In K. Futatsugi, A. T. Nakagawa, and T. Tamai, editors, *Cafe: An Industrial-Strength Algebraic Formal Method*, pages 1–31. Elsevier, 2000. <http://maude.csl.sri.com/papers>.
8. F. Durán, S. Lucas, and J. Meseguer. MTT: The Maude termination tool (system description). In A. Armando, P. Baumgartner, and G. Dowek, editors, *Automated*

- Reasoning 4th International Joint Conference, IJCAR 2008 Sydney, Australia, August 12-15, 2008 Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 313–319. Springer, 2008.
9. F. Durán, S. Lucas, and J. Meseguer. Termination modulo combinations of equational theories. In S. Ghilardi and R. Sebastiani, editors, *Frontiers of Combining Systems, 7th International Symposium, FroCoS 2009, Trento, Italy, September 16-18, 2009. Proceedings*, volume 5749 of *Lecture Notes in Computer Science*, pages 246–262. Springer, 2009.
 10. F. Durán and J. Meseguer. CRC 3: A Church-Rosser checker tool for conditional order-sorted equational Maude specifications. Available at <http://maude.lcc.uma.es/CRCChC>, 2009.
 11. J. Giesl and D. Kapur. Dependency pairs for equational rewriting. In *Proceedings of the 12th International Conference on Rewriting Techniques and Applications (RTA'01)*, volume 2051 of *Lecture Notes in Computer Science*, pages 93–108. Springer, 2001.
 12. I. Gnaedig, C. Kirchner, and H. Kirchner. Equational completion in order-sorted algebras. *Theoretical Computer Science*, 72:169–202, 1990.
 13. J. Goguen and J. Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105:217–273, 1992. Also as Technical Report SRI-CSL-89-10, July, 1989.
 14. J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15(4):1155–1194, 1986.
 15. C. Kirchner, H. Kirchner, and J. Meseguer. Operational semantics of OBJ3. In T. Lepistö and A. Salomaa, editors, *Proceedings of 15th International Coll. on Automata, Languages and Programming*, volume 317 of *Lecture Notes in Computer Science*, pages 287–301. Springer, 1988.
 16. G. Peterson and M. Stickel. Complete sets of reductions for some equational theories. *Journal of ACM*, 28(2):233–264, 1981.
 17. P. Viry. Equational rules for rewriting logic. *Theoretical Computer Science*, 285(2):487–517, 2002.