On the Church-Rosser and Coherence Properties of Conditional Order-Sorted Rewrite Theories[1]

Francisco Durán

Universidad de Málaga, Spain

José Meseguer

University of Illinois at Urbana-Champaign, IL, USA

Abstract

In the effort to bring rewriting-based methods into contact with practical applications both in programing and in formal verification, there is a tension between: (i) *expressiveness and generality*—so that a wide range of applications can be expressed easily and naturally—, and (ii) *support for formal verification*, which is harder to get for general and expressive specifications. This paper answers the challenge of successfully negotiating the tension between goals (i) and (ii) for a wide class of Maude specifications, namely: (a) equational order-sorted conditional specifications $(\Sigma, E \cup A)$, corresponding to functional programs modulo axioms such as associativity and/or commutativity and/or identity axioms; and (b) order-sorted conditional rewrite theories $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$, corresponding to concurrent programs modulo axioms A. For Maude functional programs the key formal property checked is the Church-Rosser property. For concurrent declarative programs in rewriting logic, the key property checked is the coherence between rules and equations modulo the axioms A. Such properties are essential, both for executability purposes and as a basis for verifying many other properties, such as, for example, proving inductive theorems of a functional program, or correct model checking of temporal logic properties for a concurrent program. This paper develops the mathematical foundations on which the checking of these properties (or ground versions of them) is based, presents two tools, the Church-Rosser Checker (CRC) and the Coherence Checker (ChC) supporting the verification of these properties, and illustrates with examples a methodology to establish such properties using the proof obligations returned by the tools.

Key words: Maude, order-sorted conditional specifications, rewriting modulo, formal verification, Church-Rosser property, coherence

[1] This paper is an improved and substantially extended version of [27] and [26].

1. Introduction

In the effort to bring rewriting-based methods into contact with practical applications both in programing and in formal verification, there is a tension between: (i) *expressiveness and generality*—so that a wide range of applications can be expressed easily and naturally—, and (ii) *support for formal verification*, which is harder to get for general and expressive specifications. This paper answers the challenge of successfully negotiating the tension between goals (i) and (ii) for a wide class of Maude specifications, namely, either: (a) conditional order-sorted equational theories (Maude *functional modules*) of the form $(\Sigma, E \cup A)$ specifying *functional programs* modulo axioms A such as associativity and/or commutativity and/or identity of some of the function symbols in the signature Σ, or (b) conditional rewrite theories (Maude *system modules*) $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$ specifying *concurrent programs* modulo axioms A as before, whose *states* are elements of the initial algebra $\mathcal{T}_{\Sigma/E \cup A}$ associated to the underlying order-sorted equational theory $(\Sigma, E \cup A)$, and whose *concurrent transitions* are specified by the rules R, which are applied with some frozenness restrictions ϕ which, as explained in [9], may forbid rewriting under some argument positions of a function symbol.

Of course, different kinds of formal verification may be performed for an equational theory $(\Sigma, E \cup A)$ or a rewrite theory $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$. For example, we may want to prove inductive theorems about a functional program, or to model check temporal logic properties for a concurrent declarative program. However, many verification methods, including the ones just mentioned, rely on two basic properties, namely, that the equational theory $(\Sigma, E \cup A)$ is *Church-Rosser* (or at least ground Church-Rosser) modulo the axioms A; and that the rules R in the rewrite theory $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$ are *coherent* (or at least ground coherent) with the equations E modulo the axioms A. Furthermore, even before any formal verification is attempted, the (ground) Church-Rosser property of $(\Sigma, E \cup A)$ and the (ground) coherence of $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$ are essential *executability requirements*, without which the execution of $(\Sigma, E \cup A)$ as a functional program (resp., of $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$ as a concurrent program) may yield unpredictable results. Indeed, the (ground) Church-Rosser property of a functional program $(\Sigma, E \cup A)$ ensures its *determinism*, so that the final result of evaluating an input expression is unique if it exists. Likewise, the (ground) coherence of a concurrent program $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$ (which assumes that its functional fragment $(\Sigma, E \cup A)$ is ground Church-Rosser) ensures that we can always achieve the effect of rewriting with R modulo $E \cup A$ by intermingling rewriting with both E and R modulo A.

We believe that the generality and expressiveness of specifications with a rich order-sorted type structure, with conditional equations and rules, and with structural axioms such as associativity and/or commutativity and/or identity is enormously useful in practical applications (see, e.g., [10] for many examples). Therefore, we have no doubt that having methods and tools to prove such specifications Church-Rosser (resp., coherent) will be very useful. Furthermore, the more general some methods and tools are, the more *widely applicable* they

become: if a specification happens to be many-sorted or even just unsorted, since these are both special cases of the order-sorted framework, it can *a fortiori* be handled by the methods and tools that we present.

One important design decision has been *not* to support either equational completion of an equational theory $(\Sigma, E \cup A)$, or coherence completion of a rewrite theory $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$. The reason for this decision is that the specifications $(\Sigma, E \cup A)$ (resp., $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$) are *not* arbitrary ones, such as, e.g., an arbitrary collection of equations presenting, say, the theory of groups which one wants to complete into an equivalent confluent and terminating presentation. Instead, such Maude specifications are *programs*, which the user has presumably tested and expects they have the required (ground) Church-Rosser (resp., (ground) coherence) properties. Therefore, the tools we present, namely the Maude Church-Rosser Checker (CRC) tool, and the Maude Coherence Checker (ChC) tool, attempt to *check* such properties without performing any completion on the given specifications. Indeed, attempting completion processes under such circumstances seems ill advised for several reasons. Consider, for example, the case of an equational functional program $(\Sigma, E \cup A)$ that a user has written and tested and now submits to the CRC tool to check that it is Church-Rosser. If the CRC returns with success all is well. But even if the CRC returns with some unresolved proof obligations such as conditional critical pairs that it could not join, or term memberships it could not establish, *all may still be well*, except that some more formal reasoning is required. Of course, some *genuine problem*, such as a failure of confluence, may be uncovered by the returned proof obligations. But this will not be the most common case, and is not an issue that can be *automatically* settled: judicious user intervention is needed to decide whether either: (i) the specification is faulty and should be corrected, or (ii) the specification is correct, but more formal reasoning is needed.

The reasons why, very often, all may be well even though the CRC or ChC tools return unresolved proof obligations are twofold. First, since the specifications are *conditional*, the CRC tool may not be able to automatically check the Church-Rosser property (resp., the ChC tool may not be able to check the coherence property) of the given specification even though the property holds. For example, the CRC tool may return a conditional critical pair $C \Rightarrow s = t$ that could not join, but in fact, by further reasoning we may be able to show that for all substitutions θ such that the condition $C\theta$ holds, the terms $s\theta$ and $t\theta$ are joinable, which is all that is needed. A second set of reasons why all may be well even though the tools return unresolved proof obligations is that, since the specifications are programs operating on concrete data, namely, *ground terms*, all that is needed of a functional program $(\Sigma, E \cup A)$ is that it is *ground* Church-Rosser, and all that is needed of a concurrent program $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$ is that (besides its functional fragment $(\Sigma, E \cup A)$ being ground Church-Rosser), it is *ground* coherent. That is, the proof obligations returned by the tools may hold for the ground case, but their proof may require additional *inductive* reasoning.

This paper has several closely-related goals:

(1) To present the foundations of the CRC tool. This is achieved by presenting a detailed discussion of confluence and descent for order-sorted conditional specifications modulo axioms, and proving a general theorem reducing their confluence (resp., ground confluence) to the joinability of suitable conditional critical pairs under the assumption that such specifications (which may have extra variables in their conditions and righthand sides) are operationally terminating in the sense of [18], that is, terminating in the intuitive sense that an interpreter executing them will terminate for all inputs.

(2) To present likewise the foundations of the ChC tool. This is achieved by defining in detail the notions of coherence and ground coherence for conditional specifications, and proving how checking these properties can be reduced to checking appropriate conditional critical pairs between conditions and rules (plus additional conditions required in non-overlap cases). In the ground coherence case, we show how certain purely equational, inductive proof obligations are sufficient to ensure the property.

(3) To illustrate with examples a methodology that a user can follow in dealing with unresolved proof obligations returned by the CRC and ChC tools, since, as mentioned above, both the conditional nature of the specifications and the fact that often only the ground versions of the properties are really needed imply that subsequent user intervention performing further formal reasoning may sometimes be needed.

(4) To present the CRC and ChC tools and explain their use, so that a reader of this paper gains both the necessary theoretical understanding and all the practical knowledge needed to use the tools.

(5) To place the present work in the context of related work, both on confluence and equational completion methods, including the ground case; and of other work on coherence checking and completion methods.

In addressing points (3) and (4), and also in explaining the foundations mentioned in (1) and (2) above, one more aspect of the CRC and ChC tools becomes clear, namely, their practical effectiveness in dealing with complex specifications. These tools would be *ineffective* in practice if they were to return a large number of unresolved proof obligations. Due to the conditional nature of the input specifications, and the presence of axioms like associativity-commutativity for which a unification problem may have a large number of solutions, this is a real possibility: many conditional critical pairs $C \Rightarrow s = t$, such that we cannot automatically prove $s \downarrow t$, are often generated. The effectiveness of the CRC and ChC tools resides in the *reasoning methods* employed by the tools to discharge many of these unresolved critical pairs, so that in the end a relatively small number of proof obligations is returned to the user. For example, in the hereditary finite sets specification presented in Section 5.1, the CRC generates 1027 critical pairs, from which it can trivially discharge by reduction 1001 critical pairs, leaving 26 left. But further automated reasoning allows the CRC to discharge 20 of these, returning only 6 unresolved proof obligations to the user.

Yet another important feature of the CRC and ChC tools, which we illustrate with examples, such as the lists and sets example in Section 5.2, is their capacity to deal with *any* combination of associativity and/or commutativity and/or identity axioms, even though Maude's built-in order-sorted unification algorithm does not handle associative but not commutative symbols. For combinations where any associative symbol is also commutative, the tool's treatment is fully general. For cases where some symbol is associative but *not* commutative, it is well known that associative unification is not finitary. Yet, the CRC and ChC tools can handle many specifications with associative and not commutative symbols by a simple check which, if successful, allows us to replace an associativity axiom for a symbol f by either the oriented equation $f(f(x,y),z) \to f(x,f(y,z))$, or the oriented equation $f(x,f(y,z)) \to f(f(x,y),z)$ for analysis purposes. The general idea, also applied to identity axioms and borrowed from [21], is to replace a specification $\mathcal{R} = (\Sigma, A \cup B, R)$ where $A \cup B$ is a set of equational axioms by a *semantically equivalent* specification $\mathcal{R} = (\Sigma, B, \vec{A} \cup \widehat{R})$, where the axioms A have been oriented as rules, and the rules $\widehat{R}$ are the $\vec{A}, B$-*variants* of the original rules R.[2]

The CRC and ChC tools, together with their documentation, are publicly available at `http://maude.lcc.uma.es/CRChC`.

The rest of the paper is structured as follows. Section 2 introduces the notion of conditional order-sorted rewriting modulo a set of linear and regular axioms. Section 3 presents the notion of Church-Rosser conditional order-sorted specification modulo axioms, introduces key concepts such as those of strongly deterministic order-sorted equational specification, conditional critical pair, and context-joinable and unfeasible conditional critical pair, and discusses the properties of confluence and descent handled by the CRC tool. Section 4 introduces the notion of coherence of conditional rewrite theories and discusses the theoretical basis of the ChC tool, including the use of the notions of context-joinability and unfeasibility of conditional critical pairs in such a tool, and the very important case of ground coherence. Section 5 presents some guidelines on how to use the tools and illustrates their use with some examples. To wrap up, Section 6 discusses related work, presents some conclusions, and outlines some directions of future work.

2. Conditional order-sorted rewriting modulo axioms

Throughout this paper, we rely on standard terminology and theorems from the field of term rewriting (see, e.g., [51, 3, 15, 57, 14]) and order-sorted algebras [34, 56, 48]. We however introduce in this section some standard notation on conditional order-sorted rewriting modulo axioms.

We assume specifications of the form $\mathcal{R} = (\Sigma, A, R)$ where A is a collection of unconditional equational axioms that are *linear* and *regular*, and R is an A-

[2] See [21] for a definition of the $\vec{A}, B$-*variants* of a rule.

coherent set of (possibly conditional) rewrite rules (see below for further details on these notions).

Let us start by recalling the notions of order-sorted signature, terms, regular and linear equational axioms, and sort-decreasing and sort-preserving equations.

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. Throughout this paper we further assume that Σ is *preregular*, so that each term t has a least sort, denoted $ls(t)$ (see [34]), and that Σ is *kind-complete*, that is, for each sort $s \in S$ its connected component in the poset $(S, \leq)$ has a top sort, denoted $[s]$, and for each $f \in \Sigma_{s_1 \ldots s_n, s}$ there is also an $f \in \Sigma_{[s_1] \ldots [s_n], [s]}$. An order-sorted signature can always be extended to a kind-complete one. Maude automatically checks preregularity and adds a new "kind" sort $[s]$ at the top of the connected component of each sort $s \in S$ specified by the user, and automatically lifts each operator to the kind level.

Given an S-sorted set $\mathcal{X} = \{\mathcal{X}_s \mid s \in S\}$ of *mutually disjoint* sets of variables, the set $\mathcal{T}_\Sigma(\mathcal{X})_s$ denotes the set of Σ-terms of sort s with variables in $\mathcal{X}$. We denote by $\mathcal{P}(t)$ the set of positions of a Σ-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 $t[t']_p$.

For an equation $u = v$ to be well-formed, the sorts of u and v should be in the same connected component of $(S, \leq)$. For E a set equations, $[t]_E$ denotes the equivalence class of t modulo provable E-equality [34]. An equation $u = v$ is called *regular* if $\mathcal{V}ar(u) = \mathcal{V}ar(v)$, and *linear* if there are no repeated variables in either u or v. An equation $u = v$ is called *sort-decreasing* iff for each well-sorted substitution θ we have $ls(u\theta) \geq ls(v\theta)$, and is called *sort-preserving* if both $u = v$ and $v = u$ are sort-decreasing. Using substitutions that specialize variables to smaller sorts (see Section 3.2), sort-decreasingness of an equation can be easily checked. We assume throughout the paper that the equational axioms A in the specification $\mathcal{R} = (\Sigma, A, R)$ are regular, linear, and sort-preserving.[3] Sort-preservingness of A is extremely useful for performing *order-sorted* rewriting modulo A: when A-matching a subterm $t|_p$ against a rule's lefthand side to obtain a matching substitution σ, we need to check that σ is well-sorted, that is, that if a variable x has sort s, then some element in the A-equivalence class $[x\sigma]_A$ has also sort s. But by sort-preservingness of A this is equivalent to checking $ls(x\sigma) \leq s$.

Given a set of equational axioms A, a substitution σ is an A-*unifier* of t

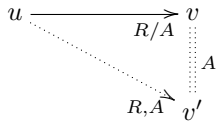
[3] When A is any combination of associativity and/or commutativity axioms, sort-preservingness is equivalent to the A-preregularity condition automatically checked by Maude (see [10, Section 22.2.5]). When A is any combination of associativity and/or commutativity and/or identity axioms, the A-*preregularity* condition checked by Maude is equivalent to the associativity and commutativity axioms being sort-preserving and the identity axioms $f(x, 1) = x$ and $f(1, x) = x$ being sort-decreasing. However, the case of an A-preregular Maude specification $\mathcal{R} = (\Sigma, A, R)$ can be reduced to that of a semantically equivalent specification whose axioms are sort-preserving by either: (i) the signature completion method presented in [36]; or (ii) turning the identity axioms into rules and performing the variant-based theory completion process described in [21].

and t' if $t\sigma =_A t'\sigma$, and it is an *A-match* from t to t' if $t' =_A t\sigma$. $Unif_A(t,t')$ denotes a *complete set of A-unifiers* of t and t'; that is, $Unif_A(t,t')$ is a set of A-unifiers of t and t' such that for any other A-unifier θ of t and t' there is a $\tau \in Unif_A(t,t')$ and a substitution ρ such that for each $x \in \mathcal{V}ar(t) \cup \mathcal{V}ar(t')$, $\theta(x) =_A \rho(\tau(x))$.

Given a rewrite theory $\mathcal{R}$ as above, we define the relation $\rightarrow_{R/A}$, either by the inference system of rewriting logic (see [9]), 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.

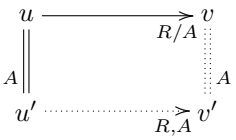
Even if there is an effective A-matching algorithm, the relation $u \rightarrow_{R/A} v$ still remains undecidable in general, since to see if $u \rightarrow_{R/A} v$ involves searching through the possibly infinite equivalence class $[u]_A$ to see whether an A-match is found for a subterm of some $u' \in [u]_A$ and the result of rewriting u' belongs to the equivalence class $[v]_A$. For this reason, a much simpler relation $\rightarrow_{R,A}$ is defined, which becomes decidable if an A-matching algorithm exists. We define (see [52]) $\rightarrow_{R,A} = \bigcup_n \rightarrow_{R,A,n}$ 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 (bi)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:

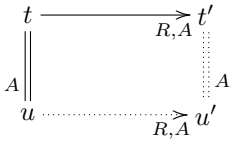


where here and in what follows dotted lines indicate existential quantification.

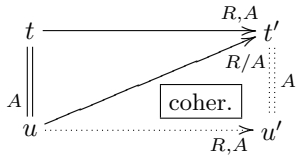
It is easy to check that A-completeness is equivalent to the following (strong) *A-coherence* property (which is really a bisimulation property):



Lemma 1. *For R a set of A-coherent rules, if $t \rightarrow_{R,A} t'$, then*



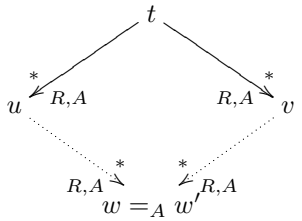
PROOF. Since $t \to_{R,A} t'$ implies $t \to_{R/A} t'$, and $u =_A t$ and $t \to_{R/A} t'$ imply $u \to_{R/A} t'$, we have



as desired. □

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., [37, 59] for the *strong* coherence completion that we use here, and [31] for the equivalent notion of extension completion). For theories A that are combinations of associativity and/or commutativity and/or identity axioms, we can make any specification A-coherent by using a completion procedure which always terminates and has a very simple description (see [52], and [10, Section 4.8] for a more informal explanation).

We say that $\mathcal{R} = (\Sigma, A, R)$ is A-*confluent*, resp. A-*terminating*, if the relation $\to_{R/A}$ is confluent, resp. terminating. If $\mathcal{R}$ is A-coherent, then A-confluence is equivalent to asserting that, for any $t \to^*_{R,A} u$, $t \to^*_{R,A} v$, we have:



and A-termination is equivalent to the termination of the $\to_{R,A}$ relation. We use the notation $t \to^!_{R/A} t'$ (resp., $t \to^!_{R,A} t'$) for a terminating rewrite, that is, a rewrite $t \to^*_{R/A} t'$ (resp., $t \to^*_{R,A} t'$) such that t' is R/A-*irreducible* (resp., R, A-*irreducible*), i.e., $\nexists t''$ such that $t' \to_{R/A} t''$ (resp., $t' \to_{R,A} t''$). We extend this notation to substitutions to write, e.g., $\tau \to^!_{R,A} \tau'$ for rewriting the terms in the assignments of a substitution τ to their normal forms using $\to_{R,A}$, that is, $\tau \to^!_{R,A} \tau'$ means that τ and τ' have the same domain, and for each variable x in that domain, $\tau(x) \to^!_{R,A} \tau'(x)$.

We say that $\mathcal{R} = (\Sigma, A, R)$ is *weakly terminating* modulo A iff for each t there is a t' such that $t \to^!_{R/A} t'$. If $\mathcal{R}$ is A-coherent, this is equivalent to the weak termination of $\to_{R,A}$.

3. Church-Rosser (Conditional) Order-Sorted Specifications Modulo Axioms

For order-sorted specifications, being Church-Rosser means not only confluence, but also a descent property (see Section 3.2), which ensures that for

each term t we have $ls(t) \geq ls(t{\downarrow_{\mathcal{R}}})$, where $t{\downarrow_{\mathcal{R}}}$ denotes a term such that $t \rightarrow^{!}_{R,A} t{\downarrow_{\mathcal{R}}}$, which by confluence is unique up to A-equivalence. In this section we introduce the notion of Church-Rosser order-sorted specification [34], and its generalization to the conditional and modulo case.

3.1. Strongly Deterministic Order-Sorted Equational Specifications

The (oriented and conditional) order-sorted equational specifications modulo axioms A that we consider in this paper are equational theories $(\Sigma, R \cup A)$ that are oriented as rewrite theories of the form $\mathcal{R} = (\Sigma, A, R)$, with A a set of regular, linear, and sort-preserving axioms. The conditional equations R in $(\Sigma, R \cup A)$ are oriented as rewrite rules of the form $l \rightarrow r$ *if* $\bigwedge_{i=1..n} u_i \rightarrow v_i$, and are assumed to be A-coherent. Furthermore, we assume that $\mathcal{R}$ is *strongly deterministic* in the following sense.

Definition 1. *Let $\mathcal{R} = (\Sigma, A, R)$ satisfy the above assumptions. A rule $l \rightarrow r$ if $\bigwedge_{i=1..n} u_i \rightarrow v_i$ in R is said to be* deterministic *iff (i) $\forall j \in [1..n], \mathcal{V}ar(u_j) \subseteq \mathcal{V}ar(l) \cup \bigcup_{k<j} \mathcal{V}ar(v_k)$, and (ii) $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l) \cup \bigcup_{j \leq n} \mathcal{V}ar(v_j)$. $\mathcal{R}$ is* deterministic *iff all its rules are so. A term t is called* strongly irreducible *with respect to R modulo A (or* strongly R, A-irreducible*) iff $t\sigma$ is an R, A-normal form for every normalized substitution σ. A deterministic rewrite theory $\mathcal{R}$ is called* strongly deterministic *iff for every rule $l \rightarrow r$ if $\bigwedge_{i=1..n} u_i \rightarrow v_i$ in R each v_i is strongly R, A-irreducible.*

Note that the above notion of strongly deterministic equational specification essentially corresponds to the notion of an *admissible* Maude functional module in the sense of [10, Section 4.6]. That is, an admissible conditional order-sorted Maude functional specification can be transformed into an equivalent strongly deterministic rewrite theory by a very simple procedure, in which equations are oriented as rewrite rules and equational conditions (ordinary ones and so called matching equations) are transformed into rewrite conditions (see [25] for a detailed algorithm).

The same way that for unconditional specifications, confluence of a set of rewrite rules can be reduced to local confluence under the termination assumption, to reduce the confluence of strongly deterministic rewrite theories to their local confluence we similarly need a suitable conditional termination assumption.

Definition 2. *A strongly deterministic rewrite theory $\mathcal{R} = (\Sigma, A, R)$ is* quasi-decreasing *iff there is a well-founded partial order $\succ$ on $\mathcal{T}_\Sigma(\mathcal{X})$ such that:*

(i) it is A-compatible*, i.e., if $v =_A u \succ u' =_A v'$ then $v \succ v'$ for all terms u, u', v, and v' in $\mathcal{T}_\Sigma(\mathcal{X})$,*

(ii) $\rightarrow_{R,A} \subseteq \succ$ and $\rhd \subseteq \succ$ (where $\rhd$ is the strict subterm relation), and

(iii) for each $l \rightarrow r$ if $\bigwedge_{i=1..n} u_i \rightarrow v_i$ in R, substitution σ, and each $0 \leq i < n$, if $u_j\sigma \rightarrow^{*}_{R,A} w_j$ and $w_j =_A v_j\sigma$, for $1 \leq j \leq i$, then $l\sigma \succ u_{j+1}\sigma$.

Note that, as shown in detail in [44] for the case of $\mathcal{R} = (\Sigma, \emptyset, R)$ unsorted (but the argument easily extends to the order-sorted and modulo cases), quasi-decreasingness is equivalent to operational termination, which is the property checked by Maude's MTT tool [19] to prove the termination of an order-sorted conditional rewrite theory $\mathcal{R} = (\Sigma, A, R)$.

3.2. The Descent Property and Church-Rosser Specifications

For an order-sorted specification $\mathcal{R} = (\Sigma, A, R)$ it is not enough to be confluent: if the canonical form $t{\downarrow_{\mathcal{R}}}$ of a term t exists, then it should provide the most complete information possible about the sort of the equivalence class $[t]_{R \cup A}$. These intuitions are captured by the notions of descent and of Church-Rosser specification.

Definition 3. *Let $\mathcal{R} = (\Sigma, A, R)$ be strongly deterministic and weakly terminating modulo A. We say that $\mathcal{R}$ has the* descent *property (resp.,* ground descent *property) iff for each term (resp., ground term) t there exists a term t' such that $t \rightarrow^{!}_{R,A} t'$ and $ls(t) \geq ls(t')$.*

Definition 4. *Let $\mathcal{R} = (\Sigma, A, R)$ be strongly deterministic and either: (i) is sort-decreasing, or (ii) has the descent property. If, in addition, $\mathcal{R}$ is confluent modulo A (resp., ground confluent modulo A), then we call $\mathcal{R}$* Church-Rosser *(resp., ground* Church-Rosser*) modulo A.*

Note that in a Church-Rosser specification $\mathcal{R}$, for each term t, if there is a term $t{\downarrow_{\mathcal{R}}}$ such that $t \rightarrow^{!}_{R,A} t{\downarrow_{\mathcal{R}}}$, then such a $t{\downarrow_{\mathcal{R}}}$ is unique up to A-equality and $ls(t)_A \geq ls(t{\downarrow_{\mathcal{R}}})$. Note also that the Church-Rosser notion as defined above is more general and flexible than the requirement of confluence and *sort-decreasingness* [40, 32]. The issue is how to find simple sufficient conditions for descent (under some termination assumption modulo A) 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 $(\Sigma, S, \leq)$, 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 $\mu \colon \hat{X} \rightarrow S$. 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 is a *specialization* of 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 subsorts 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 $(\forall X, l = r \text{ if } C)$ (resp., a rule $(\forall X, l \rightarrow r \text{ if } C)$)

is another equation $(\forall \rho(X), l\rho = r\rho \text{ if } C\rho)$ (resp., another rule $(\forall \rho(X), l\rho \rightarrow r\rho \text{ if } C\rho)$) where ρ is a specialization of X. It is easy to check that an equation $(\forall X, l = r \text{ if } C)$ (resp., a rule $(\forall X, l \rightarrow r \text{ if } C)$) is sort-decreasing in the sense explained in Section 2 iff $ls(l\rho) \geq ls(r\rho)$ for each specialization ρ. Obviously, if in a weakly terminating $\mathcal{R} = (\Sigma, A, R)$ all rules in R are sort-decreasing when viewed as unconditional rules, then $\mathcal{R}$ has the descent property. But we are *not* requiring sort decreasingness: we seek some sufficient conditions to ensure descent under the quasi-decreasingness assumption. Such conditions are called membership assertions. We let $\mathcal{R} \vdash t \rightarrow_{R,A} u$ and $\mathcal{R} \vdash t \rightarrow^*_{R,A} u$ respectively denote a one-step rewrite proof and an arbitrary length (but finite) rewrite proof in R from t to u, using the deduction rules in [9].

Definition 5. *Let $\mathcal{R} = (\Sigma, A, R)$ be a quasi-decreasing order-sorted specification satisfying the assumptions in Section 3.1. Then, the set of* (conditional) membership assertions *for a conditional rule $t \rightarrow t'$ if C is defined as*

$$\{\ t'\theta : ls(t\theta) \text{ if } C\,\theta \quad | \quad \theta \text{ is a specialization of } \mathcal{V}ar(t) \text{ and } \nexists u \text{ s.t. } t'\theta \rightarrow^!_{R,A} u \wedge ls(u) \leq ls(t\theta)\ \}$$

By definition, we say that $\mathcal{R}$ satisfies *(resp.,* inductively satisfies*) a conditional membership assertion of the form $w : s$ if D iff for each solution τ of D (resp., each ground solution τ of D) there is a term q such that $w\tau \rightarrow^*_{R,A} q$ and $ls(q) \leq s$, where if $D = \bigwedge_j u_j \rightarrow v_j$, then a substitution (resp., ground substitution) τ is a* solution *(resp.,* ground solution*) of D iff $\mathcal{R} \vdash \bigwedge_j u_j\tau \rightarrow^*_{R,A} v_j\tau$.*

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 $t\sigma =_A t'$, and $C\,\sigma =_A C'$. We denote $\mathrm{MMA}(\mathcal{R})$ the set of most general membership assertions of all of the equations in the specification $\mathcal{R}$. It is easy to show that $\mathcal{R}$ satisfies all its membership assertions iff it satisfies $\mathrm{MMA}(\mathcal{R})$. The importance of $\mathrm{MMA}(\mathcal{R})$ as a set of sufficient conditions whose satisfaction ensures the descent property is explained by the following theorem.

Theorem 1. *Let $\mathcal{R} = (\Sigma, A, R)$ be as in Definition 5. Then $\mathcal{R}$ has the descent (resp., ground descent) property if it satisfies (resp., inductively satisfies) all the conditional membership assertions in* $\mathrm{MMA}(\mathcal{R})$.

Proof. We prove the non-ground case; the proof of the ground case is similar. For simplicity we work with the set of all membership assertions of $\mathcal{R}$ rather than with the semantically equivalent set $\mathrm{MMA}(\mathcal{R})$. By the quasi-decreasingness assumption, the relation $\rightarrow^+_{R,A}$ is a well founded (strict) order. The proof is by well-founded induction on $\rightarrow^+_{R,A}$. If t is R, A-irreducible the result is obvious. Suppose instead that we have a term v such that $t \rightarrow_{R,A} v$. This means that there is a rule $l \rightarrow r$ *if* C in R, a position p, and a substitution σ that solves C, such that $t|_p =_A l\sigma$, and $v = t[r\sigma]_p$. Let ρ be the specialization of $\mathcal{V}ar(l)$ such that for each $x \in \mathcal{V}ar(l)$ the sort of x is now $ls(x\sigma)$. Then $s = ls(l\rho) = ls(l\sigma)$, and we have a substitution τ with domain $\rho(\mathcal{V}ar(l))$ such

that $\tau = \sigma\rho$. Suppose that the rule $l \to r$ *if* C had generated the membership assertion $r\rho : s$ *if* $C\rho$. Since $\mathcal{R}$ satisfies all its membership assertions and τ solves $C\rho$, we have a term w such that $r\rho\tau = r\sigma \to^*_{R,A} w$ and $s \geq ls(w)$. But then, $t[r\sigma]_p \to^*_{R,A} t[w]_p$, and, by A sort-preserving, $ls(t) = ls(t[l\sigma]_p) \geq ls(t[w]_p)$. But since $t \to^+_{R,A} t[w]_p$, applying the induction hypothesis to $t[w]_p$ we get a t' with $t \to^!_{R,A} t'$ and $ls(t) \geq ls(t[w]_p) \geq ls(t')$, as desired. □

Example 1. *Given a specification of natural numbers and integers with the usual operations and including a* `square` *operation defined by:*

```
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 CRC tool is*

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

This membership assertion must be proven 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 satisfies the ground descent property. In this case, we have to prove that we have `INT` $\vdash_{\text{ind}}$ `(∀I)(∃J) I * I` $\to^*$ `J`*, for* `I` *and* `J` *variables of sorts* `Int` *and* `Nat`*, respectively, and where* `INT` *here denotes the* rewrite theory *obtained from the original equational theory by turning each equation into a rewrite rule. This can be done using the constructor-based methods for proofs of ground reachability described in [55, 54].*

3.3. Conditional Critical Pairs and Confluence

We say that a term t *A-overlaps* another term t' with distinct variables 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 6. *Given an order-sorted equational specification $\mathcal{R} = (\Sigma, A, R)$ satisfying the assumptions in Section 3.1, and given (possibly renamed) conditional rewrite rules $l \to r$ if C and $l' \to r'$ if C' in R such that $\mathcal{V}ar(l \to r \text{ if } C) \cap \mathcal{V}ar(l' \to r' \text{ if } C') = \emptyset$ and $l|_p\sigma =_A l'\sigma$, for some nonvariable position $p \in \mathcal{P}(l)$ and A-unifier $\sigma \in Unif_A(l_p, l')$, then the triple*

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

is called a (conditional) critical pair.

Note that the critical pairs accumulate the substitution instances of the conditions in the two rules, as in [8]. 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 each $i \in [1..n]$.

Given a specification $\mathcal{R}$, let $\mathrm{MCP}(\mathcal{R})$ denote the set of most general critical pairs between rules in $\mathcal{R}$, and let $\mathrm{MCP}(\mathcal{R})\downarrow$ denote the set of critical pairs obtained after simplifying both sides of each critical pair using the equational rules in $\mathcal{R}$, and discarding trivially joinable critical pairs modulo A of the form $C \Rightarrow t = t$. As we explain in Corollary 1 below, if $\mathcal{R}$ is quasi-decreasing, satisfies the descent property, and $\mathrm{MCP}(\mathcal{R})\downarrow = \emptyset$, $\mathcal{R}$ is confluent modulo its axioms A. However, even if $\mathcal{R}$ is confluent, $\mathrm{MCP}(\mathcal{R})\downarrow$ may be nonempty. The reason for this is that what a conditional critical pair $C \Rightarrow s = t$ requires to be shown, is not the *trivial joinability* $s\downarrow_{R,A} t$, that is, the existence of terms $w =_A w'$ such that $\mathcal{R} \vdash s \to^*_{R,A} w \wedge t \to^*_{R,A} w'$, but only the joinability $s\tau\downarrow_{R,A} t\tau$ for each *solution* τ of C. That is, all critical pairs $C \Rightarrow s = t$ in $\mathrm{MCP}(\mathcal{R})$ may be joinable in the sense below, but this may not be settled just by checking $s\downarrow_{R,A} t$.

Definition 7. *We say that a conditional critical pair $C \Rightarrow s = t$ is* trivially joinable *iff $s\downarrow_{R,A} t$, and that it is* joinable *(resp.,* ground joinable*) iff for each solution (resp.,* ground solution*) τ of its condition C, we have $s\tau\downarrow_{R,A} t\tau$.*

The theorem below reduces confluence to local confluence of conditional critical pairs. It generalizes to the order-sorted, modulo, and ground cases, and to the weaker termination condition of quasi-decreasingness (instead of the stronger quasi-reductiveness condition in [2]) a similar theorem by Avenhaus and Loría-Sáenz [2].

Theorem 2. *Let $\mathcal{R} = (\Sigma, A, R)$ satisfy the assumptions in Section 3.1, be quasi-decreasing with respect to an A-compatible well-founded order $\succ$, and satisfy (resp., inductively satisfy)* $\mathrm{MMA}(\mathcal{R})$*. $\mathcal{R}$ is confluent (resp., ground confluent) iff all critical pairs in* $\mathrm{MCP}(\mathcal{R})$ *are joinable (resp., ground joinable).*

Proof. The ($\Rightarrow$) implication is trivial, so we focus on proving the ($\Leftarrow$) implication. We prove ($\Leftarrow$) in the general case by well-founded induction on $\succ$. We then explain how the proof can be specialized for the ground confluence case. Without loss of generality we may prove the results for terms t, t', and t'' such that there exist t_1 and t_2 with

$$t' \overset{*}{\underset{R,A}{\longleftarrow}} t_1 \underset{R,A}{\longleftarrow} t \underset{R,A}{\longrightarrow} t_2 \overset{*}{\underset{R,A}{\longrightarrow}} t''$$

Since by quasi-decreasingness $u \to_{R,A} v$ implies $u \succ v$, by the usual well-founded induction argument, it is enough to prove that $t_1\downarrow_{R,A} t_2$. We reason by cases depending on the positions p, q at which the one-step rewrites $t \overset{p}{\to}_{R,A} t_1$, $t \overset{q}{\to}_{R,A} t_2$ take place. The case when p and q are disjoint positions, that is, $p \not\leq q$ and $q \not\leq p$, is easy, as shown in Figure 1.

Let us now suppose the case $p \leq q$ (the case $q \leq p$ is completely symmetric). Since $w \rhd w'$ implies $w \succ w'$, by well-founded induction we may reduce to the case where $p = \Lambda$ (top position). Therefore, we have rules $l \to r$ *if* $\bigwedge_i u_i \to v_i$ and $l' \to r'$ *if* $\bigwedge_j u'_j \to v'_j$ in R such that

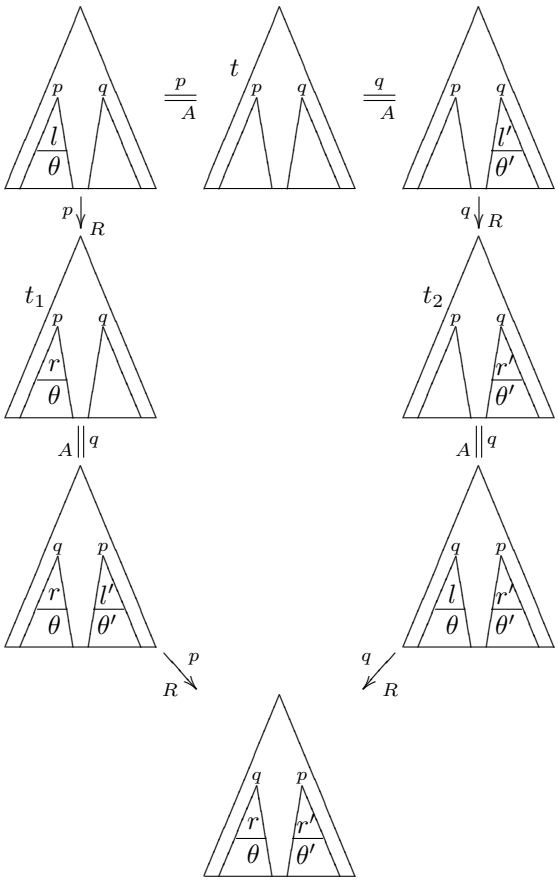


Figure 1: p and q disjoint positions case in Proof of Theorem 2.

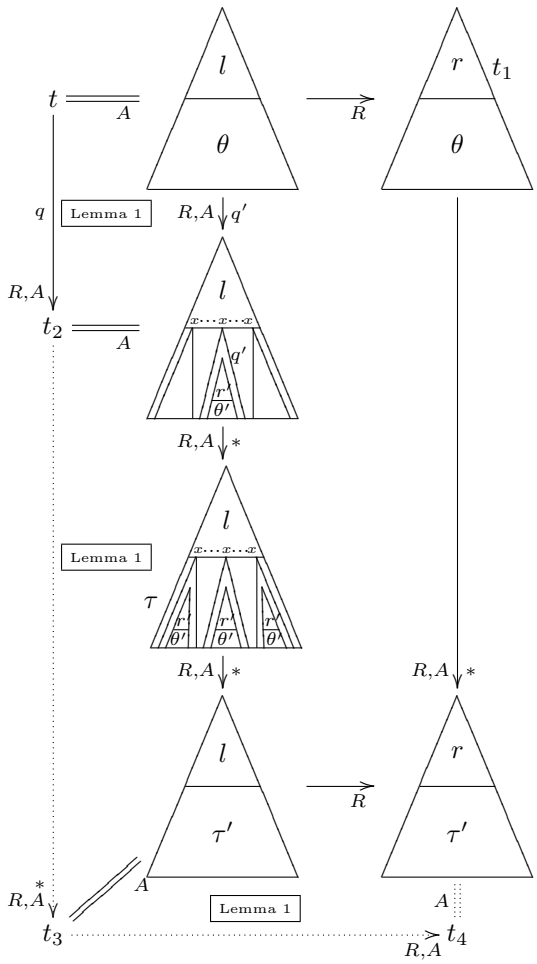
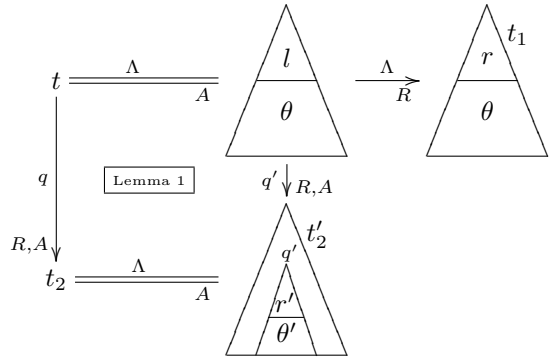


Figure 2: $p \leq q$, non-overlap case in Proof of Theorem 2.

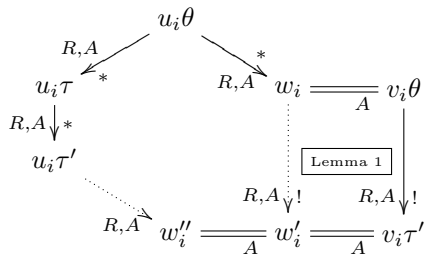


And we can consider two cases:

(a) (non-overlap case) q' is *not* a non-variable position of l, and

(b) (overlap case) q' is a non-variable position of l.

Let us first show the non-overlap case (a), summarized in Figure 2, where q' occurs at the position of a variable x of l, or below such a position. Let $q'' \leq q'$ be the position of the occurrence of x below which q is located. Then, the rewrite $l'\theta' \rightarrow_{R,A} r'\theta'$ induces also a rewrite $x\theta \rightarrow_{R,A} v$ so that $t'_2 = t'[v]_{q''}$. Let τ be the substitution such that $x\tau = v$, and $y\tau = y\theta$ otherwise. Since we do not assume sort-decreasingness, if x originally had sort s, it may be the case that $ls(v) \not\leq s$. But we can always re-type x with, say, the top sort $[s]$ to get a well-typed τ. We then have $t'_2 \rightarrow^*_{R,A} l\tau$. But since $\mathcal{R}$ satisfies MMA($\mathcal{R}$), reasoning exactly as in the proof of Theorem 1 we obtain an R, A-normalized substitution τ' such that $\tau \rightarrow^!_{R,A} \tau'$ with $ls(y\theta) \geq ls(y\tau')$ for each $y \in dom(\theta)$. So we can re-assign the original sort s to the variable x, so that τ' has the same domain as θ. Therefore, since $\theta \rightarrow^!_{R,A} \tau'$, we have $t_1 = r\theta \rightarrow^*_{R,A} r\tau'$; and we also have $t'_2 \rightarrow^*_{R,A} l\tau \rightarrow^!_{R,A} l\tau'$. Therefore, we will be done if we show that $\bigwedge_{i\in[1\ldots n]} u_i\tau' \rightarrow^*_{R,A} w'_i$, with $w'_i =_A v_i\tau'$. But by quasi-decreasingness we have $l\theta \succ u_i\theta$, $1 \leq i \leq n$, and therefore $t =_A l\theta \succ u_i\theta$, $1 \leq i \leq n$. Therefore, since by $\succ$ A-compatible we have $t \succ u_i\theta$, $i \leq i \leq n$, we can apply the confluence induction hypothesis to the $u_i\theta$, $1 \leq i \leq n$. But this means that, since by $l\theta \rightarrow_{R,A} r\theta$ thanks to $\bigwedge_{i\in[1\ldots n]} u_i\theta \rightarrow^*_{R,A} w_i$ with $w_i =_A v_i\theta$, and since v_i is strongly irreducible and τ' is a normalized substitution we have



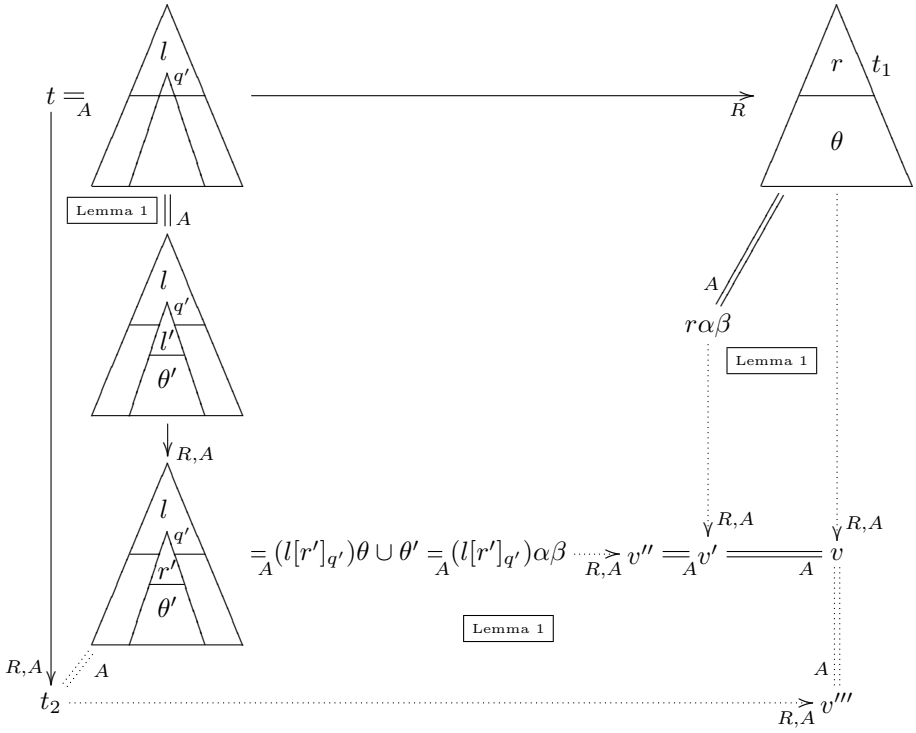


Figure 3: $p \leq q$, overlap case in Proof of Theorem 2.

And therefore $t_1 \downarrow_{R,A} t_2$, as desired.

Let us prove case (b). We have $t'_2 = (l[r']_{q'})(\theta \cup \theta')$ and $t_1 = r\theta = r(\theta \cup \theta')$, and there is a conditional critical pair $C \Rightarrow (l[r']_{q'})\alpha = r\alpha$ and a substitution β such that $\theta \cup \theta' =_A \alpha\beta$, and furthermore, $\theta \cup \theta'$ is a solution of C. Therefore, by Lemma 1, $\alpha\beta$ is also a solution of C and we have $(l[r']_q)\alpha\beta \downarrow_{R,A} r\alpha\beta$. This then gives us $t_1 \downarrow_{R,A} t_2$, as shown in the diagram in Figure 3.

Let us finally see how the above proof specializes to a proof of ground confluence when t, t_1, and t_2 are ground terms, and all critical pairs are ground joinable. Because of the assumption that $\mathcal{R} = (\Sigma, A, R)$ is deterministic and the axioms A are regular and linear (so that $t =_A t'$ implies $\mathcal{V}ar(t) = \mathcal{V}ar(t')$), the substitutions θ and θ' allowing the rewrites $t \rightarrow_{R,A} t_1$ and $t \rightarrow_{R,A} t_2$ are ground substitutions. We again reason by cases. The case of disjoint positions is again unproblematic. For case (a), we reason exactly as before to obtain an R, A-irreducible ground substitution τ' such that $\theta \rightarrow^!_{R,A} \tau'$. Then the ground confluence induction hypothesis allows us to show that $\bigwedge_{i \in [1 \ldots n]} u_i\tau' \rightarrow w'_i$, with $w'_i =_A v_i\tau'$, which then gives us $t_1 \downarrow_{R,A} t_2$, as desired. In case (b), since both θ and θ' are ground, and $\theta \cup \theta' =_A \alpha\beta$, $\alpha\beta$ is also ground, and we can use the

ground confluence assumption to get again $t_1 \downarrow_{R,A} t_2$, as desired. □

Corollary 1. *Let $\mathcal{R} = (\Sigma, A, R)$ be as in Theorem 2, and suppose that all critical pairs $C \Rightarrow s = t$ in* MCP($\mathcal{R}$) *are trivially joinable. Then $\mathcal{R}$ is confluent.*

PROOF. This follows from the fact that the rewriting relation $\rightarrow_{R,A}$ is closed under substitution, i.e., if $t \rightarrow_{R,A} t'$, and σ is a substitution, then $t\sigma \rightarrow_{R,A} t'\sigma$. Therefore, if $s \downarrow_{R,A} t$, then, a fortiori, $s\sigma \downarrow_{R,A} t\sigma$ for any solution σ of the critical pair's condition C. □

Obviously, Corollary 1 guarantees that $\text{MCP}(\mathcal{R})\downarrow = \emptyset$ is a sufficient condition for $\mathcal{R}$'s confluence. But we may have $\text{MCP}(\mathcal{R})\downarrow \neq \emptyset$ and yet all its critical pairs may be joinable, or at least ground joinable. Therefore, in the conditional case it becomes very important to use methods that can prove joinability (resp., ground joinability) of conditional critical pairs.

3.4. Context-Joinable and Unfeasible Conditional Critical Pairs

We extend to the order-sorted and modulo cases two very useful methods of proving that a conditional critical pair $C \Rightarrow s = t$ is joinable studied by Avenhaus and Loría-Sáenz in [2]. The first method consists of identifying critical pairs $C \Rightarrow s = t$ that are *context joinable*, that is, joinable if we assume the condition C as a set of additional (ground) rewrite rules to join s and t. In the second method, a conditional critical pair $C \Rightarrow s = t$ is shown joinable by showing that its condition C has no solutions whatsoever, and then $C \Rightarrow s = t$ is called *unfeasible*. The CRC tool examines all the critical pairs in $\text{MCP}(\mathcal{R})\downarrow$ trying to prove each of them either context-joinable or unfeasible. In this way, many conditional critical pairs can be discarded in practice, and in some cases no critical pairs remain.

Let a *context* $C = \{u_1 \rightarrow v_1, \ldots, u_n \rightarrow v_n\}$ 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, let the term $\overline{t}$ be the term obtained by replacing each variable $x \in \mathcal{V}ar(C)$ by the new constant $\overline{x}$.

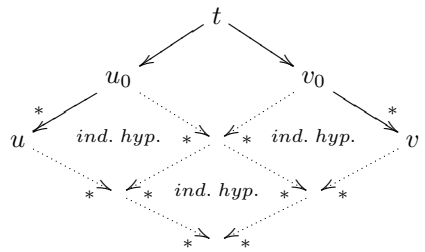
Definition 8. *Let $\mathcal{R} = (\Sigma, A, R)$ be as in Theorem 2, and let $C \Rightarrow s = t$ be a critical pair resulting from $l_i \rightarrow r_i$ if C_i for $i = 1, 2$, and $\sigma \in Unif_A(l_1|_p, l_2)$. We call the condition C of a critical pair $C \Rightarrow s = t$* unfeasible *if there is some $u \rightarrow v$ in C such that $\overline{u} \rightarrow^*_{R \cup \overline{C}, A} \overline{w}_1$, $\overline{u} \rightarrow^*_{R \cup \overline{C}, A} \overline{w}_2$, $Unif_A(w_1, w_2) = \emptyset$, and w_1 and w_2 are strongly irreducible with R modulo A; likewise, a critical pair $C \Rightarrow s = t$ is called* unfeasible *iff C is unfeasible. We call $C \Rightarrow s = t$* context-joinable *if $\overline{s} \downarrow_{R \cup \overline{C}} \overline{t}$.*

Theorem 3. *Let $\mathcal{R} = (\Sigma, A, R)$ be as in Theorem 2. If every 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 well-founded induction on $\succ$. Suppose that we have the critical peak $u \stackrel{*}{\leftarrow} t \stackrel{*}{\rightarrow} v$. All nontrivial cases to consider are of the form

$u \overset{*}{\leftarrow} u_0 \leftarrow t \rightarrow v_0 \overset{*}{\rightarrow} v$ 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 α such that αC holds. Now, if $C \Rightarrow s = t$ is context-joinable, the result follows (with minor adaptations) from [2, 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 the rules $l_1 \rightarrow r_1$ *if* C_1 and $l_2 \rightarrow r_2$ *if* C_2 which, with unifier σ, gave as the critical pair $C \Rightarrow s = t$ with condition $C = \sigma C_1 \wedge \sigma C_2$. 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 $\alpha\sigma u_i \rightarrow^*_{R,A} \alpha\sigma v_i$ and $\alpha\sigma u'_j \rightarrow^*_{R,A} \alpha\sigma v'_j$, we have that for each $u \rightarrow v$ in C we have $\alpha u \rightarrow^*_{R,A} \alpha v$, and therefore, by quasi-decreasingness of R, we have $\alpha\sigma u_i \prec \alpha\sigma l_1 \succeq \alpha\sigma l_2 \succ \alpha\sigma u'_j$.

Since $C \Rightarrow s = t$ is unfeasible, there is a condition $u \rightarrow v$ in C such that $\overline{u} \rightarrow^*_{R\cup\overline{C},A} \overline{w}_1$ and $\overline{u} \rightarrow^*_{R\cup\overline{C},A} \overline{w}_2$ with $Unif_A(w_1, w_2) = \emptyset$ and w_1 and w_2 strongly irreducible. Note that, $u \rightarrow v$ is either: (i) a condition $\sigma u_i \rightarrow \sigma v_i$ with $u_i \rightarrow v_i$ in C_1; or (ii) a condition $\sigma u'_j \rightarrow \sigma v'_j$ with $u'_j \rightarrow v'_j$ in C_2. Let us consider case (i). By [2, Lemma 4.2], we then 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 modulo A or not. But since w_1, w_2 are strongly irreducible, $\alpha\!\downarrow\!(w_1)$ and $\alpha\!\downarrow\!(w_2)$ are in canonical form (modulo A) and are *different* (by $Unif_A(w_1, w_2) = \emptyset$). But since $\alpha\sigma l_1 \succ \alpha\sigma u_i$, the confluence induction hypothesis applies to $\alpha\sigma u_i$, and therefore it is confluent, which is in contradiction with $\alpha\!\downarrow\!(w_1) \neq_A \alpha\!\downarrow\!(w_2)$.

Case (ii), where the unfeasibility problem arises in $\alpha\sigma u'_j$ is entirely similar, since $\alpha\sigma l_1 \succeq \alpha\sigma l_2 \succ \alpha\sigma u'_j$. □

Once all critical pairs in $\mathrm{MCP}(\mathcal{R})\!\downarrow$ are computed, based on this result, the CRC 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 $eq(x, x) \rightarrow tt$ are added to the rules R. Call this theory $\widehat{\mathcal{R}}_{\overline{C}}$.

(iii) In $\widehat{\mathcal{R}}_{\overline{C}}$, we search $eq(\overline{s},\overline{t}) \rightarrow^{+} tt$ up to some predetermined depth (using Maude's `search` command).

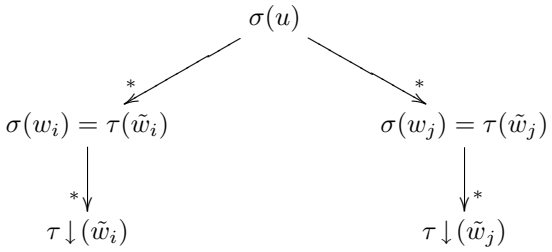
If the search is successful, then the conditional critical pair is context joinable. Otherwise, we then check whether $C \Rightarrow s = t$ is unfeasible as follows: For each condition $u_i \rightarrow v_i$, we perform in $\widehat{\mathcal{R}}_{\overline{C}}$ the search $\overline{u_i} \rightarrow^{!} x : [s]$, where $[s]$ is a top sort added to the connected component of the sort s of u_i. Let $\overline{w_1} \ldots \overline{w_m}$ be the terms thus obtained. If $m = 1$, then we can discard this term u_i and look for the next condition $u_{i+1} \rightarrow v_{i+1}$. *Otherwise*, we 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 $u_i \rightarrow v_i$ for which associated w_j, w_k satisfy (a) and (b), then the conditional critical pair $C \Rightarrow s = t$ is *unfeasible*.

This procedure can be improved as follows:

(A) Before doing this, we can first try to find *two* conditions $u_i \rightarrow v_i$, $u_j \rightarrow v_j$ in C such that $u_i =_A u_j$, and then try to get *all the canonical forms* of $\overline{u}_i$ using $\widehat{\mathcal{R}}_{\overline{C}}$ as before. This will make the process *faster* in some cases, since one focuses on *likely candidates* first.

(B) Suppose we have found canonical forms $\overline{w}_i$, $\overline{w}_j$ for $\overline{u}$ associated to a condition $u \rightarrow v$ such that w_i and w_j have no A-unifiers, but either w_i or w_j *fail to be* strongly $\mathcal{R}$*-irreducible*. Let $p_1 \ldots p_n$ (resp., $q_1 \ldots q_m$) be the *highest* nonvariable positions in w_i (resp., w_j) such that there is an *overlap* with a rule in R (and *none* of the p_l, q_r are *root positions*). Then *abstract* w_i and w_j to $\tilde{w}_i = w_i[x_1]_{p_1} \ldots [x_n]_{p_n}$ (with x_l a *fresh new variable* of the sort of $w_i|_{p_l}$) and $\tilde{w}_j = w_j[y_1]_{q_1} \ldots [y_m]_{q_m}$ (with y_r a *fresh new variable* of the sort of $w_j|_{q_r}$), respectively. Note that $\tilde{w}_i$ and $\tilde{w}_j$ are *strongly* $\mathcal{R}$*-irreducible by construction*. Then if $Unif(\tilde{w}_1, \tilde{w}_2) = \emptyset$, we can *still conclude that the critical pair is unfeasible*.

PROOF. (of (B)) Let σ be a *substitution* satisfying the condition C of the critical pair, and let X be the set of variables in C. Then we have



where $\tau\downarrow$ is the normalized substitution for the substitution τ defined below. Therefore, $\tau\downarrow(\tilde{w}_i)$ and $\tau\downarrow(\tilde{w}_j)$ are in *canonical form*, and since $\tilde{w}_i$ and $\tilde{w}_j$ *have no A-unifiers* we have $\tau\downarrow(\tilde{w}_i) \neq_A \tau\downarrow(\tilde{w}_j)$.

In more detail, the substitution τ is defined as follows:

$$\tau : X \cup \{x_1, \ldots, x_n, y_1, \ldots, y_m\} \longrightarrow T_\Sigma(X)$$

with $x_1, \ldots, x_n, y_1, \ldots, y_m$ fresh new variables, and where

(a) $\forall x \in X, \tau(x) = \sigma(x)$,

(b) $\tau(x_l) = \sigma(w_i)_{p_l}$, and

(c) $\tau(y_r) = \sigma(w_j)_{q_r}$.

□

These two optimizations are not currently available in the CRC tool.

3.5. The Proof Obligations Returned by the Church-Rosser Check

Given an order-sorted equational specification $\mathcal{R}$, the CRC tool returns a pair $\langle\ \mathrm{MCP}(\mathcal{R})^\bullet,\ \mathrm{MMA}(\mathcal{R})\ \rangle$, were $\mathrm{MCP}(\mathcal{R})^\bullet$ denotes the subset of critical pairs in $\mathrm{MCP}(\mathcal{R})\downarrow$ that could not be proved either context-joinable or unfeasible. As discussed above, 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 a quasi-decreasing specification $\mathcal{R}$ to be *Church-Rosser*. 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 still have unsatisfiable conditions in the critical pairs or in the membership assertions; that is, we can have $\langle\ \mathrm{MCP}(\mathcal{R})^\bullet,\ \mathrm{MMA}(\mathcal{R})\ \rangle \neq \langle\ \emptyset, \emptyset\ \rangle$ 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 CRC tool as a guide. In fact, as we explain in Section 5, several possibilities exist.

4. Coherence of Conditional Rewrite Theories

We assume an order-sorted rewrite theory of the form $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$, where:

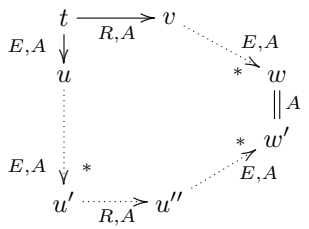
(1) ϕ is a *frozenness map* (see [9]) of the form $\phi : \Sigma \longrightarrow \mathcal{P}(\mathbb{N})$, which assigns to each operator $f\ :\ k_1 \ldots k_n \to k$ in Σ the subset $\phi(f) \subseteq \{1, \ldots, n\}$ of its *frozen arguments*, that is, those argument positions under which rewriting with R is forbidden in the rewrite theory $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$.

(2) $(\Sigma, E \cup A)$ is an order-sorted conditional equational theory, which can be converted into a strongly deterministic rewrite theory (Σ, A, E) which is Church-Rosser (resp., ground Church-Rosser). Furthermore, the regular, linear, and sort-preserving axioms A are unconditional equations at the *kind level*, i.e., each connected component in the poset $(S, \leq)$ of sorts has a top sort, and the variables in the axioms A all have such top sorts.

(3) R is a collection of A-coherent rewrite rules $l \to r$ *if* C, where C is an *equational condition*, which again can be turned into a deterministic rewrite rule of the form[4] $l \to r$ *if* $u_1 \to_E v_1 \wedge \ldots \wedge u_n \to_E v_n$ with the $v_1, \ldots, v_n$ strongly E, A-irreducible.

The following definition of coherence, due to Viry [59], intuitively states that a rewrite step with R can always be *postponed* until after performing more equational reduction with E, without compromising $E \cup A$-equality of states. Note that the condition is stronger than the so-called *weak coherence* property [59, 47], where after reduction with E we would perform $u' \to^*_{R,A} u''$. Weak coherence is less satisfactory in some respects. For example, we could not rely anymore on representing states of $\mathcal{R}$ as E, A-canonical forms to model check an LTL formula $\bigcirc \varphi$ using such states.

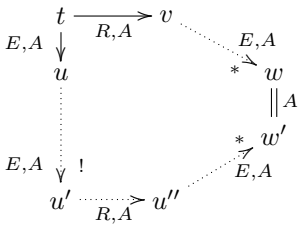
Definition 9. *A rewrite theory $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$ satisfying (1)–(3) above is called* coherent *(resp.,* ground coherent*) iff for each Σ-term t (resp., ground Σ-term t) such that $t \to_{E,A} u$ and $t \to_{R,A} v$ we have*



(C)

$\mathcal{R} = (\Sigma, E \cup A, R, \phi)$ satisfying (1)–(3) above and with (Σ, A, E) quasi-decreasing is called locally coherent *(resp.,* locally ground coherent*) iff for each Σ-term t (resp., ground Σ-term t) such that $t \to_{E,A} u$, and $t \to_{R,A} v$ we have*

[4] Note that this rule involves *two different rewrite relations*: R defines a relation $\to_{R,A}$, and E defines a relation $\to_{E,A}$. But in rewriting logic (see [9, 20]), the definition of $\to_{R,A}$ uses the auxiliary relation $\to_{E,A}$ to evaluate conditions of rules in R (see [20]). To mark this difference, the rewrites in the equational condition of a rule in R are denoted $u_i \to_E v_i$.



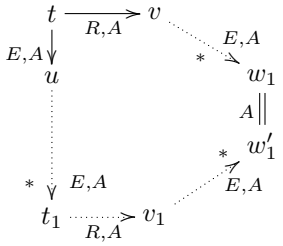
(LC)

where $s \to^{!}_{E,A} t$ *iff* $s \to^{*}_{E,A} t$ *and* t *is* E, A*-irreducible.*

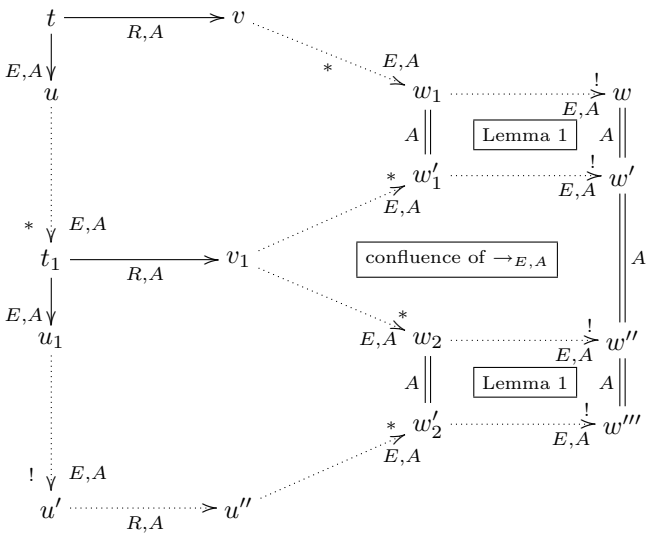
Theorem 4. *Let* $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$ *satisfy (1)–(3), with* (Σ, A, E) *quasi-decreasing. Then,* $\mathcal{R}$ *is coherent (resp., ground coherent) iff* $\mathcal{R}$ *is locally coherent (resp., locally ground coherent).*

Proof. Obviously $(LC) \Rightarrow (C)$. Let us prove that $(C) \Rightarrow (LC)$ by well-founded induction on the terminating relation $\to_{E,A}$.

Let t be any term. If $t = t{\downarrow_{E,A}}$ or $t = t{\downarrow_{R,A}}$ both (C) and (LC) hold trivially. Therefore, we may assume that $t \to_{E,A} u$ and $t \to_{R,A} v$. By coherence we then have:



If $t_1 = t_1{\downarrow_{E,A}}$ we are done, so we may assume that we have $t_1 \to_{E,A} u_1 \to^{*}_{E,A} u_1'{\downarrow_{E,A}}$. By noetherian induction on $\to_{E,A}$, t_1 is (LC) and therefore we have:



□

Since for $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$ satisfying (1)–(3), with (Σ, A, E) quasi-decreasing, for all terms t, t is coherent iff t is locally coherent, we can approach the verification of coherence for such a theory $\mathcal{R}$ as follows: We can reason by cases on the situations $u \overset{E,A}{\leftarrow} t \overset{R,A}{\rightarrow} v$ depending on whether they are or not *overlap* situations. For this we need the notion of a conditional critical pair, and the notion of conditional critical pair joinability.

Definition 10. *Given conditional rewrite rules with disjoint variables $l \to r$ if C in R and $l' \to r'$ if C' in E, their set of conditional critical pairs modulo A is defined as usual: either we find a non-variable position p in l such that $\alpha \in Unif_A(l|_p, l')$ and then we form the conditional critical pair*

$$\alpha(C) \wedge \alpha(C') \quad \Rightarrow \quad \begin{array}{c} \alpha(l[l']_p) \\ \downarrow{\scriptstyle E} \\ \alpha(l[r']_p) \end{array} \overset{A}{=\!=} \alpha(l) \xrightarrow[R]{} \alpha(r) \tag{I}$$

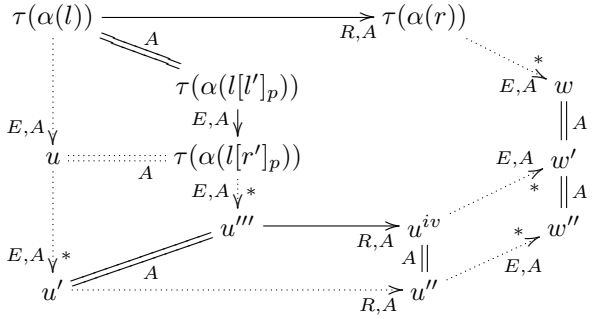
or we have a non-variable and non-frozen *position p' in l' such that $\alpha \in Unif_A(l'|_{p'}, l)$ and we form the conditional critical pair:*

$$\alpha(C) \wedge \alpha(C') \quad \Rightarrow \quad \begin{array}{c} \alpha(l') \\ \downarrow{\scriptstyle E} \\ \alpha(r') \end{array} \overset{A}{=\!=} \alpha(l'[l]_{p'}) \xrightarrow[R]{} \alpha(l'[r]_{p'}) \tag{II}$$

We typically write these critical pairs as $\alpha(C) \wedge \alpha(C') \Rightarrow \alpha(l[r']_p) \rightarrow \alpha(r)$ *and* $\alpha(C) \wedge \alpha(C') \Rightarrow \alpha(r') \rightarrow \alpha(l'[r]_{p'})$.

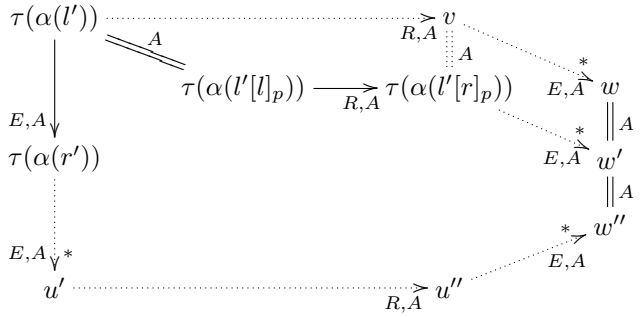
Note the use of $\rightarrow$ instead of $=$ to distinguish these critical pairs from those introduced in Section 3, were only one rewrite relation was used.

We say that a critical pair of type (I) is *joinable* iff for any substitution τ such that $E \cup A \vdash \tau\alpha(C) \wedge \tau\alpha(C')$ we then have[5]



Of course, by $(C) \Leftrightarrow (LC)$ *it is enough to make this check* with $u''' = u'''{\downarrow_{E,A}}$.

Similarly, we say that a critical pair of type (II) is *joinable* iff for any substitution τ such that $E \cup A \vdash \tau\alpha(C) \wedge \tau\alpha(C')$ we then have



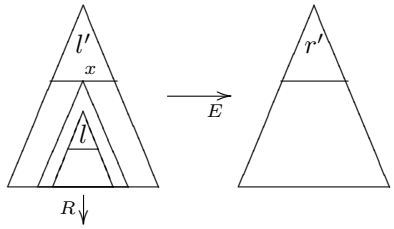
where, again, by $(C) \Leftrightarrow (LC)$ it is enough to perform the check with $u' = u'{\downarrow_{E,A}}$.

Of course, joinability of all conditional critical pairs of R and E is a *necessary* condition for coherence. The challenge now is to find a set of *sufficient conditions* for coherence that includes the joinability of conditional critical pairs.

Specifically, non-overlapping situations between equations and rules require additional conditions. In the case of *coherence* checking, we need to worry, not

[5]Note that this diagram, and others to come, would be much simplified using the relations $\rightarrow_{E/A}$ and $\rightarrow_{R/A}$. However, actual computation uses the relations $\rightarrow_{E,A}$ and $\rightarrow_{R,A}$; but thanks to the A-coherence of E and R we can use Lemma 1 to fill in the appropriate quadrilaterals.

only about overlapping situations as for the case of confluence, but also about non-overlapping of R under E, that is, for $l' \to_E r'$ *if* C' in E and $l \to_R r$ *if* C in R we need to worry about non-overlap situations of the form:



This situation can be problematic in two related ways: (1) when $l' \to_E r'$ is unconditional but not linear, or (2) when $l' \to_E r'$ *if* C' is conditional. The problem with case (1) is well-understood since [59]. The problem with case (2) was also mentioned by Viry in [59]; it has to do with the fact that the satisfiability of the condition C' in an equation $l' \to_E r'$ *if* C' *depends* on the substitution θ (it may hold or not depending on the given θ). But since R *rewrites* the substitution θ, we do *not know if* C' *will hold anymore after a one-step rewrite with the rule* $l \to_R r$ *if* C. Note that we can view cases of unconditional $l \to r$ with l *non-linear* as special cases of (2), since we can *linearize* l, and give an explicit equality condition instead. E.g., `x + x = x` becomes `x + y = x if x = y`.

Theorem 5. *Let* $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$ *satisfy (1)–(3), with* (Σ, A, E) *quasi-decreasing. Then if:*

(i) all conditional critical pairs are joinable and

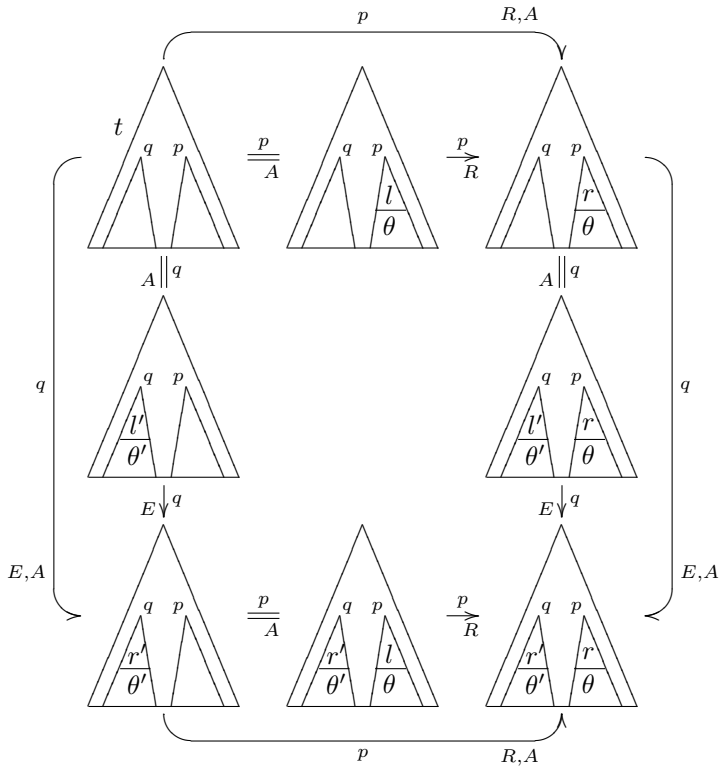
(ii) for any equation $l' \to r'$ *if* C' *in* E*, for each* $x \in \mathcal{V}ar(l')$ *such that* x *is non-frozen in* l'*, then* either

  (a) x *is such that* $x \notin \mathcal{V}ar(C')$*,* x *is also non-frozen in* r'*, and* x *is linear in both* l' *and* r'*, or*

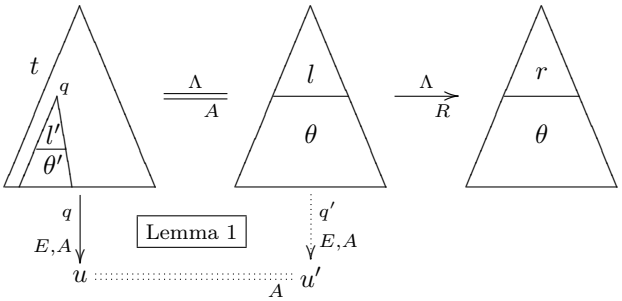
  (b) the sort s *of* x *is such that* no rewriting with $\to_{R,A}$ is possible for terms of such sort s*,*

then $\mathcal{R}$ *is coherent.*

PROOF. Consider $\overset{E,A}{u} \overset{q}{\swarrow} t \overset{p}{\searrow} \overset{R,A}{v}$. Then if neither $p \leq q$, nor $q \leq p$ (disjoint positions) the coherence property holds for t, since we have:



Therefore, the heart of the matter lies in the cases $p \leq q$ and $q \leq p$. Let us first consider the case $p \leq q$. Without loss of generality we may assume $p = \Lambda$ (top position). Therefore, for $l \to_R r$ *if* C and $l' \to_E r'$ *if* C' we have:



with $E \cup A \vdash \theta(C)$ and $E \cup A \vdash \theta'(C')$.

There are now two possibilities:

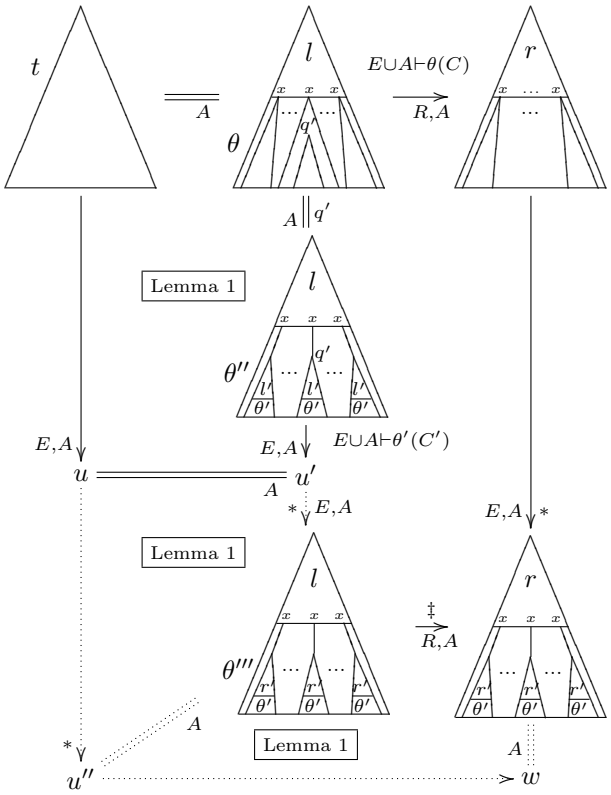


Figure 4: Non-overlap case in Proof of Theorem 5.

(a) (*overlap* case) q' is a non-variable position of l.

(b) (*non-overlap* case) q' is *not* a non-variable position of l.

Let us first show in Figure 4 that the non-overlap case is unproblematic. The rule application ‡ is possible because, by the assumption of E being *confluent* modulo A, and C being equational, since $\theta =_A \theta''$ so $\theta\downarrow_{E,A} =_A \theta''\downarrow_{E,A}$, and therefore, since $\theta'' =_{E\cup A} \theta'''$, $E \cup A \vdash \theta(C)$ implies $E \cup A \vdash \theta'''(C)$.

Therefore, we are only left with the overlap case, in which q is a non-variable position in l. Therefore, we have the situation in Figure 5.

Let us now look more carefully at θ and θ'. Let $X = \mathcal{V}ar(l \to_R r \text{ if } C)$, $X' = \mathcal{V}ar(l' \to_E r' \text{ if } C')$, $X_0 = \mathcal{V}ar(l|_p) \subseteq X$, and $X'_0 = \mathcal{V}ar(l') \subseteq X'$; and let $\theta_0 = \theta|_{X_0}$ and $\theta'_0 = \theta'|_{X'_0}$. We therefore have a unifier $\theta_0 \uplus \theta'_0$ (by X and X' disjoint) such that

$$(\theta_0 \uplus \theta'_0)(l|_p) =_A (\theta_0 \uplus \theta'_0)(l'),$$

and therefore we have $\alpha \in \mathit{Unif}_A(l|_p, l')$ and τ_0 such that $\tau_0 \circ \alpha =_A \theta_0 \uplus \theta'_0$.

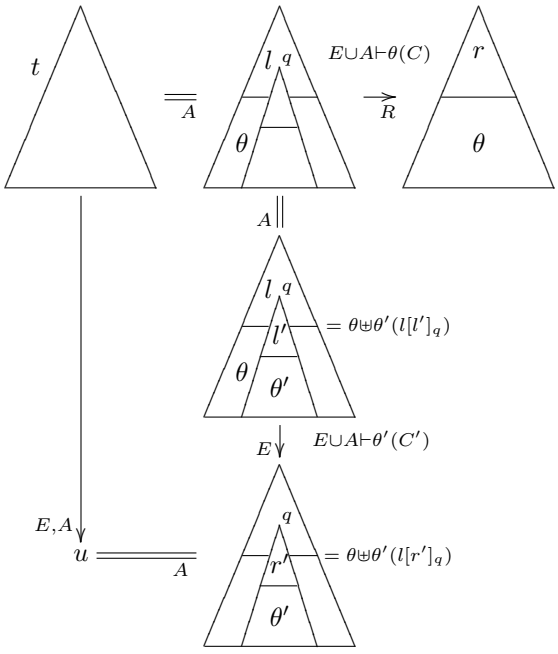


Figure 5: Overlap case, with q a non-variable position in l, in Proof of Theorem 5.

Let us define $\hat{\tau} : (X \cup X') - (X_0 \cup X'_0) \to T_\Sigma(\mathcal{X})$ (with $\mathcal{X}$ an infinite set of variables) by

$$\hat{\tau}(x) = \begin{cases} \theta(x) & \text{if } x \in X - X_0 \\ \theta'(x) & \text{if } x \in X' - X'_0 \end{cases}$$

Then, since $\alpha = \alpha|_{X_0 \cup X'_0}$, we obviously have that for $\tau = \tau_0 \uplus \hat{\tau}$ the equality

$$\tau \circ \alpha =_A \theta \uplus \theta'$$

holds. Furthermore, since $E \cup A \vdash \theta(C) \wedge \theta'(C')$ and therefore $E \cup A \vdash \theta \uplus \theta'(C) \wedge \theta \uplus \theta'(C')$, we also have $E \cup A \vdash \tau(\alpha(C)) \wedge \tau(\alpha(C'))$. And by the joinability assumption we then have the diagram in Figure 6 as desired.

Therefore, *the only remaining case* is that of $p \leq q$. Again, we can consider two subcases, an *overlap* subcase, and a *non-overlap* subcase. The proof of the overlap subcase is given by the diagram of Figure 7.

The only case left is the non-overlap case with $p \leq q$, where we have the situation depicted in the diagram of Figure 8.

Note that for this to happen, x must be a *non-frozen* variable in l'. If x disappears from r', or appears more than once in r', the situation is hopeless (no single rewrite with R possible). Similarly, if x appears more than once in l', the situation is likewise hopeless, since the patterns l' *will not match* the term $\theta'(l')[\theta(r)]_p$ (the other subterms under x will be different!).

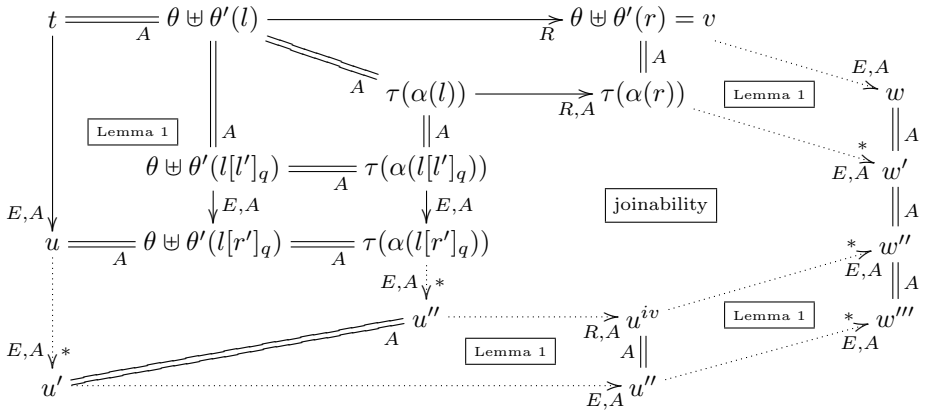


Figure 6: Overlap case, with q a non-variable position in l, in Proof of Theorem 5.

Let us prove that condition (ii) is enough. Case $(ii).(b)$ makes the very possibility of a non-overlap case with R below E impossible, and the diagram in Figure 9 proves case $(ii).(a)$. Notice that $E \cup A \vdash \theta(C')$ implies $E \cup A \vdash \theta''(C')$ because $\theta =_A \theta'$ and $\theta''(C') = \theta'(C')$ since $x \notin \mathcal{V}ar(C')$.

□

Condition $(ii).(b)$ of Theorem 5 requires a fixpoint calculation. An algorithm that checks that situations where a non-frozen variable x in a lefthand side of an equation fails to satisfy $(ii).(a)$ or $(ii).(b)$ is provided in [24].

4.1. Context-Joinability and Unfeasibility of Conditional Critical Pairs

As for the conditional critical pairs of the confluence check (see Section 3.4), from those conditional critical pairs for E and R which cannot be trivially joined, the ChC tool can currently automatically discard those that are either *context-joinable* or *unfeasible*.

Definition 11. *Let $\mathcal{R} = (\Sigma, E \cup A, R)$ be an order-sorted conditional rewrite theory satisfying conditions (1)–(3), with E quasi-decreasing modulo A w.r.t. an A-compatible order $\succ$. We call a conditional critical pair $C \Rightarrow s \to t$* unfeasible *iff its condition is unfeasible with respect to (Σ, A, E) in the sense of Definition 8.*

As pointed out in Section 3.4, a Maude order-sorted conditional specification can be converted into an order-sorted deterministic rewrite theory with a simple procedure (see, e.g., [24]). Maude checks that the conditional equational specifications entered are deterministic (cf. [10]), and we assume it is operationally terminating, and therefore there exists a well-founded A-compatible order $\succ$ such that we can use (a simple adaptation of) the results in [2] and their extension to the Maude case [26], to discard those conditional critical pairs generated that are unfeasible.

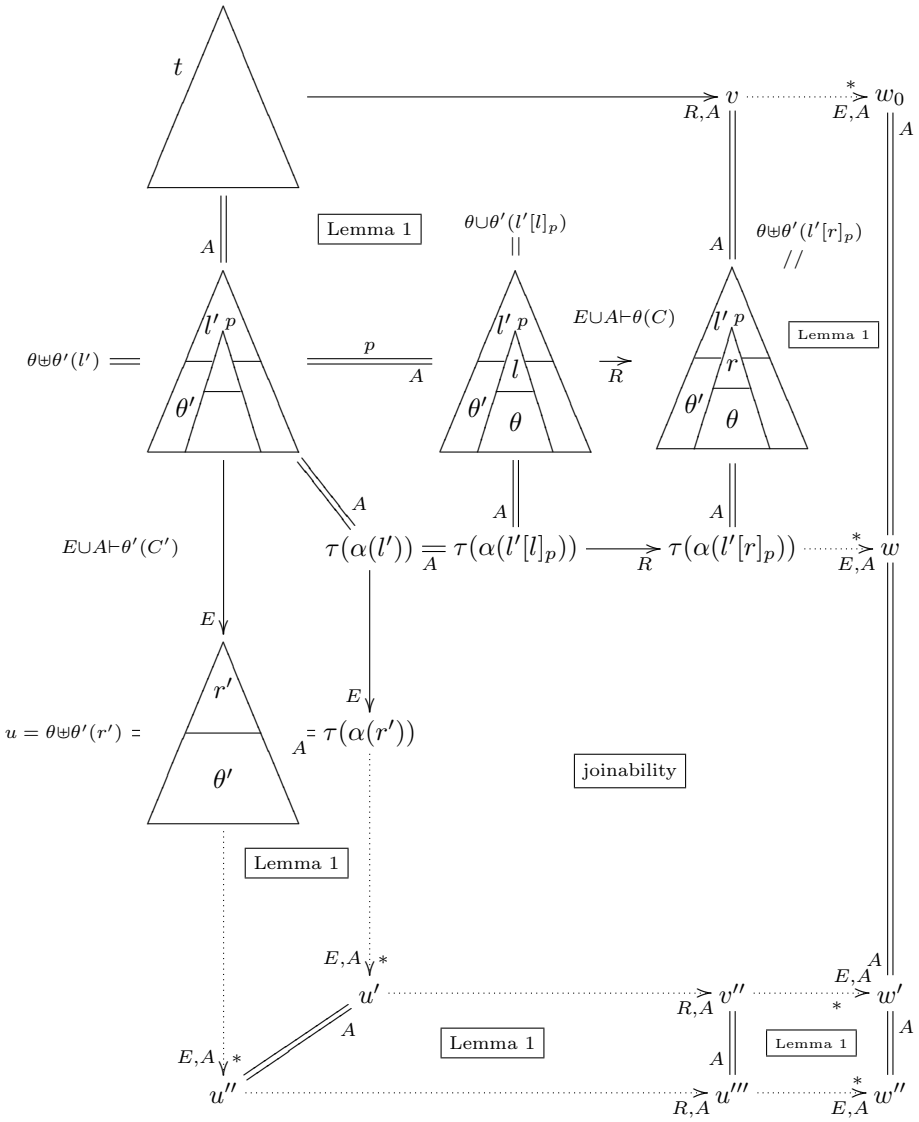


Figure 7: $p \leq q$, overlap case, in Proof of Theorem 5.

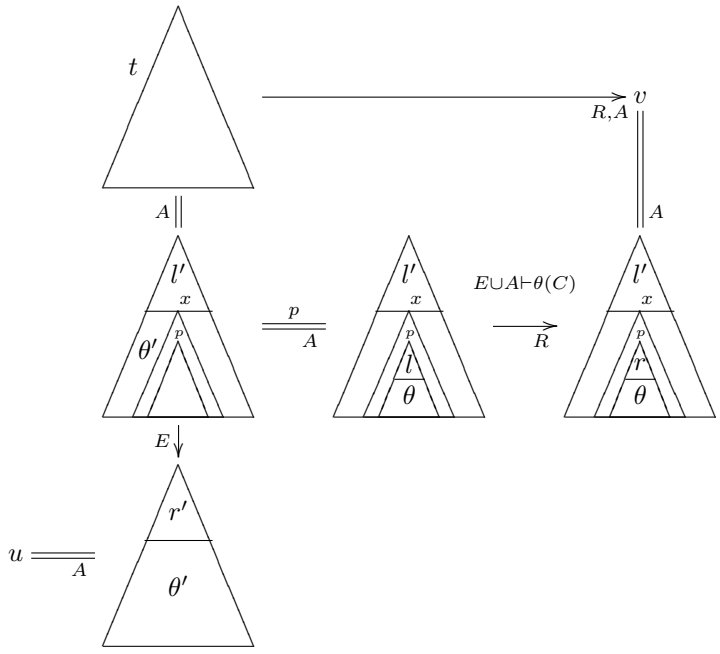
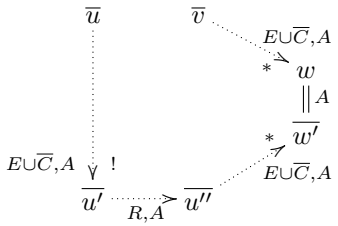


Figure 8: $p \leq q$, non-overlap case, in Proof of Theorem 5.

Definition 12. *Given a rewrite theory $\mathcal{R} = (\Sigma, E \cup A, R)$ satisfying conditions (1)–(3) above, a non-joinable conditional critical pair $C \Rightarrow u \rightarrow v$ (coming from a conditional critical pair $C \Rightarrow {}_{E,A}\!\leftarrow u \leftarrow t \rightarrow v \rightarrow_{R,A}$) is called* context-joinable *if and only if in the extended rewrite theory $\mathcal{R}_C = (\Sigma \cup \overline{X}, E \cup \overline{C} \cup A, R)$ we have:*



Lemma 2. *If the conditional critical pair $C \Rightarrow u \rightarrow v$ is context joinable, then*

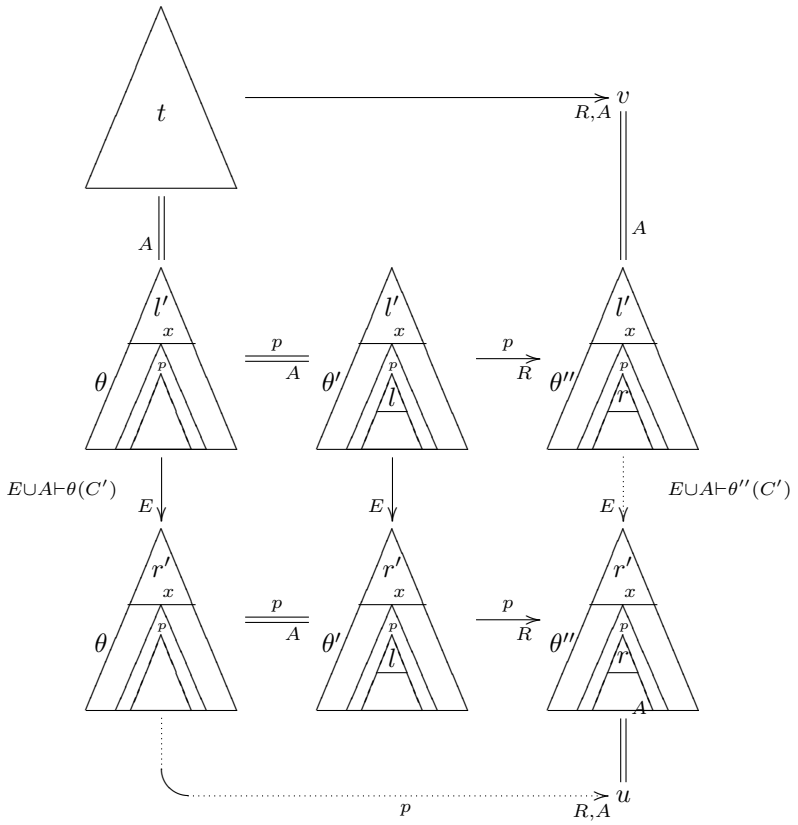
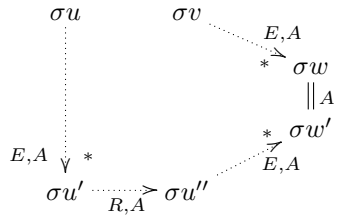


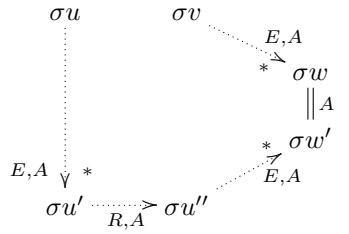
Figure 9: Case $(ii).(a)$ in the Proof of Theorem 5.

for all substitutions σ such that σC holds we have



and therefore, the coherence property holds for the conditional critical pair $C \Rightarrow {}_{E,A}\!\swarrow^{u}\ {}^{t}\ \searrow^{v}\!{}_{R,A}$.

Proof. By a simple adaptation of [2, Lemma 4.2], since σC holds, we have $\sigma u \to^*_{E,A} \sigma u'$, $\sigma v \to^*_{E,A} \sigma w$, and $\sigma u'' \to^*_{E,A} \sigma w$. But we also have $\overline{u'} \to^1_{R,A} \overline{u''}$, where $\overline{u'}$ is in $E \cup \overline{C}$-canonical form. This means that if we applied $l \to r$ *if* D in R to $\overline{u'}$ with substitution α and $D = u_1 \to_E v_1 \wedge \ldots \wedge u_n \to_E v_n$ then $\alpha u_1 \to^*_{E \cup \overline{C},A} \alpha v_1 \wedge \ldots \wedge \alpha u_n \to^*_{E \cup \overline{C},A} \alpha v_n$ holds, which means that (by [2, Lemma 4.2]), since σC holds we have $\sigma\alpha u_1 \to^*_{E,A} \sigma\alpha v_1 \wedge \ldots \wedge \sigma\alpha u_n \to^*_{E,A} \sigma\alpha v_n$. Therefore, we have



as desired. □

In the implementation of the ChC tool, the lefthand sides of the rules in C are simplified to their normal forms before turning their variables into constants.

4.2. The Ground Coherence Case

Assume that Σ has a sub-signature of constructors Ω that has been verified to be *sufficiently complete* with respect to the equations E modulo A, that is, that for each ground Σ-term t there is a ground Ω-term t' such that $t \to^*_{E,A} t'$. Then, we can view each $f \in \Sigma$ with a different syntactic form from Ω as a *frozen* operator, since any ground term in E,A-canonical form will *not* contain the symbol f. This automatically excludes all problematic non-overlaps with R below E except for:

(a) constructor equations, and

(b) equations $f(t_1, \ldots, t_n) \rightarrow r$ *if* C in E with $f \in \Sigma - \Omega$, and (for axioms A which are combinations of associativity and/or commutativity and/or identity axioms) with f having the identity, left identity, or right identity attributes, and such that the lefthand side of the equation resulting from the variant-based transformation to remove the identity attributes has a *non-frozen variable* (see [21] for details on the variant-based transformation).

Therefore, assuming again that A is a combination of associativity and/or commutativity and/or identity, for ground coherence under the assumption of frozenness of defined symbols, we only have to check condition (ii) in Theorem 5 on equations of types (a) and (b) above.

Furthermore, for those conditional critical pairs $C \Rightarrow u \rightarrow v$ for which we have not been able to check unfeasibility nor context joinability, we can guarantee their *inductive ground joinability* if for $w = u{\downarrow_{E,A}}$ and for the rewrite theory

$$\tilde{\mathcal{R}}_{\overline{C},Y} = (\Sigma \cup \overline{X} \cup \bigcup_{\lambda} \overline{Y_\lambda}, E \cup \overline{C} \cup A, \{\lambda' : l \rightarrow \overline{r}^{Y_\lambda} \mid \lambda : l \rightarrow r \ \textit{if}\ D \in R\})$$

where $X = \mathcal{V}ar(C \Rightarrow u \rightarrow v)$, $Y_\lambda = \mathcal{V}ar(r) - \mathcal{V}ar(l)$ for a rule $\lambda : l \rightarrow r$ *if* D in R, and $\overline{r}^{Y_\lambda}$ denotes the term r with all variables in Y_λ made constants, we can prove $\overline{w} \rightarrow^1_{R,A} \overline{v}_i$ for some substitution θ_i for the variables of $l \rightarrow \overline{r}$ for some such rule. Then inductive ground joinability amounts to proving the inductive theorem:

$$E \cup A \vdash_{ind} \ C \Rightarrow (\theta_1 D_1 \wedge v_1 = v) \vee \ldots \vee (\theta_n D_n \wedge v_n = v)$$

where θ_i and D_i are the matching substitution and the condition of the rule used to reach each $\overline{v}_i$ from $\overline{w}$.

The intuition behind this procedure is as follows. When we have a critical pair $C \Rightarrow u \rightarrow v$ that we cannot automatically join, in some cases it is just because the conditions of the appropriate rules cannot be satisfied, or because the term resulting from the application of the rule or the term v we want to reach cannot be further simplified. In some other cases it is just because of the way in which the equations were written, because they are too general or simply because they cannot be applied to terms with variables as such (more on this in Section 5). However, in the ground case, we can reduce the ground joinability problem to an inductive *equational* proof based on the application of the rules whose lefthand sides match the lefthand side of a particular conditional critical pair. For a critical pair $C \Rightarrow u \rightarrow v$ to be rewritten we just need to find a match θ of $w = u{\downarrow_{E,A}}$ with the lefthand side of a rule $\lambda' : l \rightarrow \overline{r}^{Y_\lambda}$ (coming from $\lambda : l \rightarrow r$ *if* D) such that its condition is satisfied ($D\theta$) and the term reached is provably equal to v (i.e., v must be proved equal to $u[\overline{r}^{Y_\lambda}\theta]_p$ for some position p after restoring the variables in Y_λ). Notice that since there might be more than one match with each equation, the conjuncts in the proof obligation above are indexed by $1..n$ rather than by the labels of the rules.

5. How to Use the Tools

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

We assume a context of use in which the user has already developed an *executable specification* of his/her intended system, and that this specification has already been *tested* with examples, so that the user is in fact reasonably confident that the specification is, respectively, *ground Church-Rosser* or *ground coherent*, and wants only to check the corresponding property with the tools. In the case of the CRC it is assumed that the specification has previously been checked to be operationally terminating, and in the case of the ChC that its equational sub-specification is Church-Rosser (or at least ground Church-Rosser) and operationally terminating.

The tools can only *guarantee* success automatically when the user's specification is unconditional, has sort-decreasing equations, and is confluent or coherent and, furthermore, any associativity axiom in A for an operator has a corresponding commutativity axiom. In all other cases, the fact that the tools do not generate any proof obligations is only a *sufficient* condition, so that even when they return a collection of proof obligations, the specification may still be *ground* Church-Rosser (resp., *ground* coherent), or for a conditional specification it may even be Church-Rosser (resp., coherent).

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 rules to the user's specification in a mindless way. In many cases it will certainly lead to a nonterminating process. For the CRC in some cases this is even impossible in the standard sense, because some critical pair cannot be oriented. 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 tools 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 often 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. In the case of the ChC, since the user's specification has an initial model semantics, and the equational sub-specification is assumed to be ground Church-Rosser and operationally terminating, the proof of the inductive rewritability of the critical pairs can be attempted, and conditional critical pairs can be discharged if their conditions are proven unsatisfiable.

In the case of the Church-Rosser checker, 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/she can prove operational 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 whether 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. Of course, if the new equations added are those returned by the CRC as proof obligations, the initial algebra semantics is automatically preserved and does not need to be checked, since the added equations are by construction *theorems* derivable from the original equations $E \cup A$.

- The other alternative is to reason inductively about the *ground joinability* of the critical pairs, and also about the *inductive satisfaction* of the membership assertions, 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 orienting the equations of T as rewrite rules. An approach to inductive proofs for membership assertions with $\overrightarrow{T}$ has already been sketched in Section 3.2. For proving ground joinability, several proof methods, e.g., [53, 5, 39, 46, 6, 1, 7], can be used. In particular, for order-sorted specifications, constructor-based methods such as those described in [55, 54] can be used.

An unresolved methodological issue in the case of the CRC is what to do with *conditional* critical pairs, or conditional membership assertions, whose conditions are *unsatisfiable*. As we explain in Section 3.4, we currently discard critical pairs which the tool can show are *unfeasible* or *context-joinable*, but all remaining not trivially joinable critical pairs are returned to the user. Since we do not *know* whether the specification is Church-Rosser, we cannot use any methods that rely in the Church-Rosser assumption to discard them. Perhaps a

modular/hierarchical approach could be used, in conjunction with the inductive proof methods described above, to establish the unsatisfiability of such conditions to discard the corresponding proof obligations.

The CRC and ChC tools are both implemented in Maude using reflection as extensions of the Full Maude language [23, 17]. They accept as inputs any Maude (or Full Maude) conditional order-sorted equational theories (resp., conditional order-sorted rewrite theories) satisfying the requirements already mentioned in Sections 3–4. However, no use of built-in operators that rely on the underlaying C++ implementation of Maude is allowed: such operators should be fully specified by equations. Also, the `owise` feature[6] is not allowed (see [10, Section 4.5.4]).

We give in the following sections examples illustrating the use of the tools. The examples have been chosen trying to highlight those features not simultaneously supported by other tools, namely, order-sortedness, conditional equations and rules, and rewriting modulo axioms. All the examples and details of their verification can be found at `http://maude.lcc.uma.es/CRChC`.

5.1. Hereditarily Finite Sets

The following functional 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 R. Sasse and J. Meseguer and is inspired by the generalized sets module in Maude's prelude [10, 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 `_~_` is the set equality predicate. The membership relation 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 .
```

[6] 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.

```
  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 the labels 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 1027 critical pairs. Most of them are trivially joinable, and therefore discarded. From the remaining 26 critical pairs, all of which are conditional, 20 are discarded because they can be proven to be 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 extend the module with 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 (#S', #N)` and `true`, with `#N` a new constant of sort `Magma`, can be joined in the extended module.

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 `09` 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 .
```

This is not enough, however. With these new two equations, the tool gives us now four conditional critical pairs.

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

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` the check 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 proven operationally terminating,[7] module `HF-SETS-3` is confluent.

5.2. 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.

[7] The termination of the `HF-SETS-`i modules, as the rest of the termination proofs in this article, has been carried out using the MTT tool (see [19, 22, 18, 21] for details on the MTT tool and on the different techniques and transformations available for checking the termination of membership equational logic and rewriting logic specifications).

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 the CRC 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 [21]), we can conclude that it is Church-Rosser.

As explained in Section 1, to handle this specification, the CRC applies several semantics-preserving transformations on the original module to remove identity attributes and associativity attributes that do not come with commutativity ones and turning them into explicit equations. We refer the interested reader to [25] for details on the use of this transformation in the CRC, and to [21] for a detailed description of the variant transformation used.

5.3. The Bakery Protocol

The bakery protocol is a classical solution by Lamport [42] to the problem of achieving mutual exclusion between processes, as originally stated by Dijkstra [16], and then extended by Knuth in [41]. The algorithm is based on the procedure commonly used in bakeries and deli shops, in which every customer gets a number when entering the store. Each client takes as its number the successor of the maximum of the numbers of the clients in the store. The next client to be served is the one with the smallest number.

In our specification, processes are represented as terms of sort `BProcess`. The elements of sort `BProcess` are constructed by an operator `<_,_,_>`, which takes the identifier of the process (a natural number of sort `MNat`), the mode it is currently in (a constant of sort `Mode`), and the number it has been assigned

(of sort `MNat`). The state of the bakery is represented as a term of sort `GBState`, constructed by an operator `[[_]]` whose argument is a term of sort `BState`, which represents a multiset of processes.

A process can be in modes `sleep`, `wait` or `crit`. The rules describe how each process goes from being sleeping to waiting, from waiting to its critical section, and then back to sleeping. When a process is in the `sleep` mode it has a 0 number; a process in `wait` or `crit` mode has a number greater than zero. Auxiliary functions `maxNumber` and `minNzNumber` return, respectively, the maximum and minimum, without considering zeros, of the numbers of the processes in a `BState`. If the set of processes passed to the `minNzNumber` function is either empty or all of them have zero as their number, i.e., are in the `sleep` mode, then `minNzNumber` returns `0`.

The following `MNAT` module defines the sort `MNat` with constructors `0` and `s_`, a less than predicate `_<_`, and associative and commutative operators `min` and `max` that return, respectively, the smallest and the greatest of two natural numbers. Constants representing numbers $1 \ldots 5$ are also defined. The predefined module `TRUTH-VALUE` defines a `Bool` sort with constants `true` and `false`.

```
(fmod MNAT is
   protecting TRUTH-VALUE .
   sort MNat .
   op 0 : -> MNat [ctor] .
   op s_ : MNat -> MNat [ctor] .
   op _<_ : MNat MNat -> Bool .
   ops min max : MNat MNat -> MNat [assoc comm] .
   ops 1 2 3 4 5 : -> MNat .
   vars N N' : MNat .
   eq 1 = s 0 . eq 2 = s 1 . eq 3 = s 2 . eq 4 = s 3 . eq 5 = s 4 .
   eq [MNat-1] : N < 0 = false .
   eq [MNat-2] : 0 < s N = true .
   eq [MNat-3] : s N < s N' = N < N' .
   eq [MNat-4] : min(0, N) = 0 .
   eq [MNat-5] : min(s N, s N') = s min(N, N') .
   eq [MNat-6] : max(0, N) = N .
   eq [MNat-7] : max(s N, s N') = s max(N, N') .
 endfm)
```

Given the `MNAT` module, the following `BAKERY` module specifies the bakery protocol as explained above.

```
(mod BAKERY is
   protecting MNAT .
   sorts Id Mode BProcess BState GBState .
   ops sleep wait crit : -> Mode [ctor] .
   op <_`,_`,_> : MNat Mode MNat -> BProcess [ctor] .
   subsort BProcess < BState .
   op __ : BState BState -> BState [ctor assoc comm id: none] .
   op none : -> BState [ctor] .
   op `[`[_`]`] : BState -> GBState [ctor] .
   var P : Mode . vars I N M : MNat . var BSt : BState .

   ---- initial state
   op initial : MNat -> BState .
   eq initial(s N) = < s N, sleep, 0 > initial(N) .
   eq initial(0) = none .

   ---- max of the numbers assigned to processes (0 if none)
   op maxNumber : BState -> MNat .
   op maxNumber : BState MNat -> MNat .
   eq [max-1] : maxNumber(< I, P, N > BSt)
```

```
    = max(N, maxNumber(BSt)) .
  eq [max-2] : maxNumber(none) = 0 .

  ---- min. of the nonzero numbers assigned to processes (0 if none)
  op minNzNumber : BState -> MNat .
  op minNzNumber : BState MNat -> MNat .
  eq [min-1] : minNzNumber(< I, P, 0 > BSt) = minNzNumber(BSt) .
  eq [min-2] : minNzNumber(< I, P, s N > BSt)
    = minNzNumber(BSt, s N) .
  eq [min-3] : minNzNumber(none) = 0 .
  eq [min-4] : minNzNumber(< I, P, 0 > BSt, M) = minNzNumber(BSt, M) .
  eq [min-5] : minNzNumber(< I, P, s N > BSt, M)
    = minNzNumber(BSt, min(M, s N)) .
  eq [min-6] : minNzNumber(none, M) = M .

  rl [sleep2wait] : [[ < I, sleep, 0 > BSt ]]
    => [[ < I, wait, s maxNumber(BSt) > BSt ]] .
 crl [wait2crit] : [[ < I, wait, N > BSt ]]
    => [[ < I, crit, N > BSt ]]
    if N < minNzNumber(BSt) .
  rl [crit2sleep] : [[ < I, crit, N > BSt ]]
    => [[ < I, sleep, 0 > BSt ]] .
endm)
```

This specification makes use of some of the advanced features supported by the CRC and ChC tools: it is an order-sorted specification, with a conditional rule, two associative-commutative operators (`min` and `max`), and an associative-commutative operator with identity (`__`).

Before reducing or rewriting any term, we should make sure that it satisfies the expected executability requirements: the equational part must be checked terminating and Church-Rosser, and the rules must be coherent with the equations.

The termination of the equational part can be checked using the MTT tool [19].

The CRC gives the following result:

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

Since the equational part of the specification is terminating, and the CRC tool certifies that it is locally confluent and sort decreasing, we can conclude that it is Church-Rosser.

The `BAKERY` module is also ground coherent, as shown by the result returned by the ChC tool:

```
Maude> (check ground coherence BAKERY .)
Coherence checking of BAKERY
Coherence checking solution:
All critical pairs have been rewritten and all equations are non-
    ↪constructor.
The specification is ground coherent.
```

Let us now verify some properties about this protocol. For instance, let us try to verify *mutual exclusion*, that is, that two processes are never simultaneously in their critical sections, and *liveness*, that is, that whenever a process enters

the waiting mode, it will eventually enter its critical section. To do that we could use the Maude LTL model checker.

However, notice that the range of numbers that can be assigned to customers is unbounded, which creates an infinite number of reachable states from an initial configuration of processes generated by the `initial` operator for any value of its argument greater than 1. Therefore, we should model check these properties using an abstraction. We can for instance define an equational abstraction [49] by adding to the `BAKERY` module equations defining a quotient of the set of states.

To define an abstraction we can take into account the fact that the process with the smallest number is the one getting into the critical section, and that we should not change the order in which the assigned numbers are given. We can therefore safely decrease the numbers of all processes if the smallest of the numbers given is greater than 1. We can do so in the following module extending the `BAKERY` module by adding a few equations and leaving the rules unchanged:

```
(mod ABSTRACT-BAKERY is
   including BAKERY .
   var  P : Mode .
   var  BSt : BState .
   vars I N : MNat .

  ceq [AB] :
     [[ BSt ]]
    = [[ dec(BSt) ]]
     if 1 < minNzNumber(BSt) .

   op dec : BState -> BState .
   op dec : MNat -> MNat .
   eq dec(< I, P, N > BSt) = < I, P, dec(N) > dec(BSt) .
   eq dec(none) = none .
   eq dec(0) = 0 .
   eq dec(s N) = N .
endm)
```

The intuition behind this abstraction is basically that, if there is no customer with number 1 and some of them have numbers different from 0, then all the numbers of non-sleeping customers, i.e., with a nonzero number, can be decreased by 1. The auxiliary `dec` function decreases (by 1) the number of each of the non-sleeping processes in a given `BState`. This abstraction makes the set of reachable states finite, since the numbers assigned to N processes will never grow beyond $N + 1$. As soon as the customer with the smallest number is served, the numbers of all customers are decreased.

We can check the satisfaction of the Church-Rosser property of the equational part of the specification using the CRC tool. The result given by the tool is the following:

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

Since the specification is terminating, we can conclude that it is also confluent. The last requirement is the coherence of equations and rules. The result of

the ChC tool is as follows:

```
Maude> (check ground coherence ABSTRACT-BAKERY .)
Coherence checking of ABSTRACT-BAKERY
Coherence checking solution:
The following critical pairs cannot be rewritten:

ccp for AB and wait2crit
  [[ dec(BSt:BState) < I:MNat, wait, dec(N:MNat) > ]]
  => [[ BSt:BState < I:MNat, crit, N:MNat > ]]
  if N:MNat < minNzNumber(BSt:BState) = true
  /\ s 0 < minNzNumber(BSt:BState < I:MNat, wait, N:MNat >) = true .

Inductive ground joinability amounts to proving the following
    ↪inductive theorem:
E U A |-ind A(BSt:BState ; I:MNat ; N:MNat)
      N:MNat < minNzNumber(BSt:BState) = true
      /\ s 0 < minNzNumber(BSt:BState < I:MNat, wait, N:MNat >) = true
      => dec(N:MNat) < minNzNumber(dec(BSt:BState)) = true
          /\ [[ dec(BSt:BState) < I:MNat, crit, dec(N:MNat) > ]]
             = [[ dec(BSt:BState) < I:MNat, crit, dec(N:MNat) > ]]
```

One single critical pair is given by the tool. And, as we asked for a ground coherence check, the associated inductive equational proof obligation to be discharged is given as part of the output of the tool.

The first key observation to interpret these critical pairs is that `TRUTH-VALUE` and `MNAT` are *protected* in `ABSTRACT-BAKERY`.[8] This follows from the confluence, termination, and the sufficient completeness,[9] of the equational part of the `ABSTRACT-BAKERY` module, plus the observation that no equations involving either `0` or the `s_` function, or `true` or `false` have been added in `ABSTRACT-BAKERY`. An inductive proof discharging this proof obligation is relatively easy to do.

In order to specify the desired mutual exclusion and liveness properties, we may specify the state predicates `wait(`N`)`, `crit(`N`)`, and `2crit`, which are satisfied, respectively, when process N is in `wait` mode, when process N is in `crit` mode, and when there are two processes simultaneously in the critical section:

```
(mod BAKERY-PREDS is
   protecting BAKERY-2 .
   including SATISFACTION .

   subsort GBState < State .
   ops wait crit : MNat -> Prop [ctor] .
   op 2crit : -> Prop [ctor] .

   vars X Y N M : MNat .  var  BSt : BState .

   eq [[ < N, wait, X > BSt ]] |= wait(N) = true .
   eq [[ < N, crit, X > BSt ]] |= crit(N) = true .
   eq [[ < N, crit, X > < M, crit, Y > BSt ]] |= 2crit = true .
endm)
```

[8] A sort `S` is protected in an importation of a module `M'` into another module `M` if no new data items of sort `S` are added, and no data items of sort `S` are identified in `M` (no junk and no confusion).

[9] Note that there is only one conditional equation, the `AB` equation, which is not required to be considered in the sufficient completeness check because it operates on constructors. Thus, although the SCC [35] does not support conditional axioms, it can be used, and has in fact been used, to prove the sufficient completeness of the specification.

The SATISFACTION module is a predefined module declaring sorts State and Prop and an operator

```
op _|=_ : State Prop -> Bool [frozen] .
```

that represents the satisfaction of a given proposition in a given state.

The preservation of these state predicates can be guaranteed if we show that the BAKERY-PREDS module protects TRUTH-VALUE. This follows from the sufficient completeness, termination, confluence and sort-decreasingness of the BAKERY-PREDS module, plus the observation of the absence of any equations having true or false in their lefthand sides.

For the checking of the Church-Rosser property the CRC tool can be used.

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

The correctness of the abstraction requires deadlock freedom. To ensure deadlock freedom, we can use the automatic module transformation described in [10, Section 15.3], which preserves all the desired executability properties. With this transformation, we obtain a semantically equivalent, deadlock-free version of our specification.

We can finally verify our desired properties on the specification resulting from the transformation.

We can check mutual exclusion for, e.g., five processes as follows:

```
Maude> (red modelCheck([[ initial(5) ]], [] ~ (2crit)) .)
result Bool :
  true
```

And liveness also for five processes with:

```
Maude> (red modelCheck([[ initial(5) ]],
              (wait(1) |-> crit(1)) /\
              (wait(2) |-> crit(2)) /\
              (wait(3) |-> crit(3)) /\
              (wait(4) |-> crit(4)) /\
              (wait(5) |-> crit(5))) .)
result Bool :
  true
```

5.4. An Unordered Communication Channel

Consider a communication channel in which messages can get out of order. There is a sender and a receiver. The sender is sending a sequence of data items, for example numbers. The receiver is supposed to get the sequence in the exact same order in which they were in the sender's sequence. To achieve this in-order communication in spite of the unordered nature of the channel, the sender sends each data item in a message together with a sequence number, and the receiver sends back an ack message indicating that has received the item. The Full Maude specification of the protocol is as follows:

```
(mod UNORDERED-CHANNEL is
   sorts Nat NatList Msg Conf State .
   subsort Msg < Conf .
   op 0 : -> Nat [ctor] .
   op s : Nat -> Nat [ctor] .
   op nil : -> NatList [ctor] .
   op _;_ : Nat NatList -> NatList [ctor] .    ---- list constructor
   op _@_ : NatList NatList -> NatList .       ---- list append
   op `[_`,_`] : Nat Nat -> Msg [ctor] .
   op ack : Nat -> Msg [ctor] .
   op null : -> Conf [ctor] .
   op __ : Conf Conf -> Conf [ctor assoc comm id: null] .
   op `{_`,_|_|_`,_`} : NatList Nat Conf NatList Nat -> State [ctor] .

   vars N M J K : Nat .          var  C : Conf .
   vars L P Q : NatList .

   eq nil @ L = L .              eq (N ; L) @ P = N ; (L @ P) .

   rl [snd]: {N ; L, M | C | P, K} => {N ; L, M | [N, M] C | P, K} .
   rl [rec]: {L, M | [N, J] C | P, J}
     => {L, M | ack(J) C | P @ (N ; nil), s(J)}  .
   rl [rec-ack]: {N ; L, J | ack(J) C | P, M} => {L, s(J) | C | P, M} .
endm)
```

The contents of the unordered channel is modeled as a *multiset* of messages of sort `Conf`. The entire system state, involving the sender, the channel, and the receiver is a 5-tuple of sort `State`, where the components are:

- a buffer for the sender containing the current list of items to be sent,
- a counter for the sender keeping track of the sequence number for items to be sent,
- the contents of the unordered channel,
- a buffer for the receiver storing the sequence of items already received, and
- a counter for the receiver keeping track of the sequence number for items received.

One essential property of this protocol is of course that it achieves *in-order communication* in spite of the unordered communication medium. We can specify this in-order communication property as an *invariant* in Maude. We will assume that all initial states are of the form:

```
{n1 ; ... ; nk ; nil, 0 | null | nil, 0}
```

That is, the sender's buffer contains a list of numbers `n1 ; ... ; nk ; nil` and has the counter set to `0`, the channel is empty, and the receiver's buffer is also empty. Also, the receiver's counter is initially set to `0`.

In specifying the invariant, the auxiliary notion of a list prefix is useful. Given lists L and L' we say that L is a *prefix* of L' iff either: (1) $L = L'$, or (2) there is a nonempty list L'' such that L `@` $L'' = L'$.

```
(mod UNORDERED-CHANNEL-INVARIANT is
   including UNORDERED-CHANNEL .
   protecting BOOL-OPS .
```

```
   op _~_ : Nat Nat -> Bool [comm] . *** equality predicate

   vars M N K : Nat .
   var  C : Conf .
   vars L L' L'' : NatList .

   eq 0 ~ 0 = tt .
   eq 0 ~ s(N) = ff .
   eq s(N) ~ s(M) = N ~ N .

   op prefix : NatList State -> Truth .
   eq [I1]: prefix(M ; L, {L', N | C | K ; L'', K})
     = (M ~ K) and  prefix(L, {L', N | C | L'', K}) .
   eq [I3]: prefix(L, {L, N | C | nil, K}) = tt .
   eq [I4]: prefix(nil, {L', N | C | M ; L'', K}) = ff .
endm)
```

Notice that the `_~_` predicate is declared commutative, and the `_and_` operator is declared commutative and associative with identity element `tt`.

The equational part of the specification can be checked terminating and Church-Rosser using the MTT [19] and the CRC. And the rules can be shown to be ground coherent with the equations by using the ChC tool.

```
Maude> (check ground coherence .)

Coherence checking of UNORDERED-CHANNEL
Coherence checking solution:
All critical pairs have been rewritten and all equations are non-
    ↪constructor.
The specification is ground coherent.
```

The problem with this simple example is that one cannot verify the invariant using the `search` command in Maude, because, due to the `snd` rule, the number of messages that can be present in the channel is unbounded, so that there is an infinite number of reachable states. One should therefore use an *equational abstraction* [49].

```
(mod UNORDERED-CHANNEL-ABSTRACTION is
   including UNORDERED-CHANNEL-INVARIANT .
   vars M N P K : Nat .
   vars L L' L'' : NatList .
   var  C : Conf .

   eq [A1]: {L, M | [N, P] [N, P] C | L', K}
     = {L, M | [N, P] C | L', K} .
endm)
```

As in the bakery example in Section 5.3, there are of course several key properties that such an abstraction should satisfy:

(1) the set of states reachable from any initial state should be finite,

(2) the equational theory should be ground confluent and terminating,

(3) the rules should be ground coherent with the equations, and

(4) the abstraction should preserve the invariant.

Properties (1), (2) and (4) can easily be checked. For (3) we can use the ChC.

```
Maude> (check ground coherence .)
Coherence checking of UNORDERED-CHANNEL-ABSTRACTION
Coherence checking solution:
The following critical pairs cannot be rewritten:

cp for A1 and rec
  { L:NatList, M:Nat | #3:Conf [N:Nat, J:Nat ]
                     | P:NatList, J:Nat}
  => { L:NatList, M:Nat | #3:Conf ack(J:Nat) [ N:Nat, J:Nat ]
                        | P:NatList @ N:Nat ; nil, s(J:Nat) } .

Inductive ground joinability amounts to proving the following
    ↪inductive theorem:
E U A |-ind A(#3:Conf ; J:Nat ; L:NatList ; M:Nat ; N:Nat ; P:NatList)
            { L:NatList, M:Nat | #3:Conf ack(J:Nat)
                               | P:NatList @ N:Nat ; nil, s(J:Nat) }
            = { L:NatList, M:Nat | #3:Conf ack(J:Nat) [ N:Nat, J:Nat ]
                               | P:NatList @ N:Nat ; nil, s(J:Nat) }

cp for A1 and rec
  { L:NatList, M:Nat | [ N:Nat, J:Nat] | P:NatList, J:Nat }
  => { L:NatList, M:Nat | ack(J:Nat) [ N:Nat, J:Nat]
                        | P:NatList @ N:Nat ; nil, s(J:Nat) } .
Inductive ground joinability amounts to proving the following
    ↪inductive theorem:
E U A |-ind A(J:Nat ; L:NatList ; M:Nat ; N:Nat ; P:NatList)
            { L:NatList, M:Nat | ack(J:Nat)
                               | P:NatList @ N:Nat ; nil, s(J:Nat) }
            = { L:NatList, M:Nat | ack(J:Nat) [ N:Nat, J:Nat ]
                               | P:NatList @ N:Nat ; nil, s(J:Nat) }
```

These critical pairs are in fact not rewritable, and indicate that a rule is missing. We can add the rule:

```
(mod UNORDERED-CHANNEL-ABSTRACTION-2 is
   inc UNORDERED-CHANNEL-ABSTRACTION .
   vars M N K : Nat . vars L L' : NatList . var  C : Conf .

   rl [rec2]: {L, M | [N, K] C | L', K}
     => {L, M | [N, K] ack(K) C | L' @ (N ; nil), s(K)} .
endm)
```

After checking again properties (1), (2) and (4) above, we can check also the ground coherence of the specification.

```
Maude> (check ground coherence .)
Coherence checking of UNORDERED-CHANNEL-ABSTRACTION-2
Coherence checking solution:
All critical pairs have been rewritten, and no rule can be applied
below non-frozen and non-linear variables of equations.
```

6. Related Work and Conclusions

The results we present on methods for proving confluence of conditional order-sorted equational specifications are part of a substantial body of work on confluence and/or completion methods for such specifications. Among other references, methods for unconditional order-sorted specifications were studied in [56], and for the modulo case in [32, 60]. Completion methods for conditional order-sorted specifications were treated in [30] using a reduction to many-sorted specifications proposed in [33]. Our work extends that previous work and also the work of Avenhaus and Loría-Sáenz [2] on confluence of conditional unsorted

specifications. To the best of our knowledge, Theorem 2 is the most general characterization to date for the confluence (resp., ground confluence) of conditional order-sorted specifications modulo axioms which are terminating in a meaningful "operational" way, that is, such that a reduction interpreter will always terminate with a term in normal form. Therefore, the CRC tool covers a very general class of order-sorted equational specifications. Also related to our work on confluence is previous work on confluence and/or completion of specifications in membership equational logic [8], as well as the work of Comon in [11, 12], which in some sense addresses a middle ground between order-sorted and membership equational specifications using tree automata techniques.

Since our interest is not only on confluence but also on ground confluence, our work is related to methods for proving ground confluence and ground joinability, e.g., [53, 39, 46, 6, 38, 7]. The CRC tool generates proof obligations that can then be subjected to formal analysis for proving ground joinability of critical pairs. Therefore, the above work can be seen as complementary to ours in helping to discharge such inductive proof obligations. In particular, the methods of Bouhoula [7] can treat order-sorted specifications, and the recent constructor-based inductive methods for ground joinability in [54, 55] can treat order-sorted specifications modulo axioms.

Regarding work on coherence, there is of course a connection with coherence work for equational specifications [52, 37, 4], but the most closely related work studies coherence between equations (and possibly axioms) and non-equational rules describing transitions in a rewrite theory, including [58, 47, 59]. The work of Marché [45], which studies coherence between two sets of equational rules, covers a middle ground between the work on making equational specifications coherent modulo axioms, and the work on making the rules of a non-equational rewrite theory coherent with its equations and axioms. Our computation of critical pairs between equations and rules under frozenness constraints has some similarities with the computation of critical pairs for context-sensitive equational specifications in [43], but the purposes are quite different, since in [43] the goal is to prove the confluence of context-sensitive equational specifications. All the above-mentioned work addresses only unconditional specifications and, except for [47], only in the unsorted case. Furthermore, the most complete previous work on coherence of rewrite theories, namely [59], covers the modulo case only for AC axioms. To the best of our knowledge, our work is the first to cover the coherence of conditional order-sorted rewrite theories modulo regular and linear axioms A, possibly with frozenness constraints. And, as far as we know, the first to study and give proof methods for the case of *ground* coherence of such specifications. Ground coherence is in fact the property most needed in practice, for example when model checking temporal logic properties of a finite-state rewrite theory, or when proving that a finite-state abstraction of a rewrite theory [49, 29, 50] is correct for purposes of verifying temporal logic properties of an infinite-state concurrent system. Fortunately, unlike the case of ground confluence, which is harder to prove than confluence, ground coherence is in some ways easier to prove than coherence.

Regarding tools for checking the Church-Rosser property of equational spec-

ifications, in the unsorted and unconditional case the most general tool available is probably CiMe [13], which supports completion of equational theories modulo a rich family of equational axioms. To the best of our knowledge, the CRC tool is the first to support the checking of confluence for operationally terminating specifications in the general case of conditional order-sorted specifications modulo associativity and/or commutativity and/or identity. Of course, this includes as special cases the checking of many-sorted or unsorted specifications under such general assumptions. For checking coherence of rewrite theories, the ChC tool seems to be the only tool currently available.

Future work should proceed in several complementary directions. First of all, since all critical pair computations are instances of narrowing modulo axioms, new versions of Maude supporting narrowing modulo axioms at the C++ level will lead to more efficient versions of the CRC and ChC tools. The following cases are currently not supported and should be investigated: (i) possibly nonterminating conditional order-sorted equational specifications; (ii) rewrite theories whose equational part is Church-Rosser but may be nonterminating; and (iii) rewrite theories whose rules may contain rewrites in their conditions. Also, a tighter integration between the CRC, ChC, ITP, SCC, and MTT Maude tools is highly desirable: this will be supported by the upcoming Maude Formal Environment (MFE) [28]. In particular, support for proofs of ground joinability in the Maude ITP should be provided, so that proof obligations generated by the CRC tool can be discharged (using MFE to interoperate both tools); likewise, proof obligations generated by the ChC should be discharged using other tools in the MFE. Finally, modularity techniques, which can facilitate proofs of confluence or coherence for large specifications, should also be investigated and supported in future versions of the CRC and ChC tools.

Acknowledgements. We thank the referees for their careful reading of a previous version and their very helpful suggestions to improve the exposition. We also thank Salvador Lucas and Camilo Rocha for their help discharging some of the proof obligations encountered when preparing the examples in Section 5, and Santiago Escobar for very fruitful discussions on the use of the narrowing library in Maude as a component of the CRC and ChC tools. 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] J. Avenhaus, T. Hillenbrand, and B. Löchner. On using ground joinable equations in equational theorem proving. *Journal of Symbolic Computation*, 36(1-2):217–233, 2003.

[2] J. Avenhaus and C. Loría-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 1994, Proceedings*, volume 822 of *Lecture Notes in Computer Science*, pages 215–229. Springer, 1994.

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

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

[5] L. Bachmair, N. Dershowitz, and D. A. Plaisted. Completion without failure. In A. H. Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume 2 (Rewriting Techniques), pages 1–30. Academic Press, 1989.

[6] K. Becker. Proving ground confluence and inductive validity in constructor based equational specifications. In M.-C. Gaudel and J.-P. Jouannaud, editors, *Theory and Practice of Software Development, International Joint Conference CAAP/FASE, Proceedings*, volume 668 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 1993.

[7] A. Bouhoula. Simultaneous checking of completeness and ground confluence for algebraic specifications. *ACM Transactions on Computational Logic*, 10(3), 2009.

[8] A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoretical Computer Science*, 236(1):35–132, 2000.

[9] R. Bruni and J. Meseguer. Semantic foundations for generalized rewrite theories. *Theoretical Computer Science*, 351(1):286–414, 2006.

[10] 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.

[11] H. Comon. Completion of rewrite systems with membership constraints. Part I: Deduction rules. *Journal of Symbolic Computation*, 25(4):397–419, 1998.

[12] H. Comon. Completion of rewrite systems with membership constraints. Part II: Constraint solving. *Journal of Symbolic Computation*, 25(4):421–453, 1998.

[13] E. Contejean and C. Marché. CiME: Completion modulo E. In H. Ganzinger, editor, *7th International Conference on Rewriting Techniques and Applications*, volume 1103 of *Lecture Notes in Computer Science*, pages 416–419. Springer, 1996.

[14] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume Formal Models and Sematics (B), pages 244–320. Elsevier, 1990.

[15] N. Dershowitz and D. Plaisted. Rewriting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 9, pages 535–610. Elsevier, 2001.

[16] E. W. Dijkstra. Solution of a problem in concurrent programming control. *Communications of the ACM*, 8(9):569, 1965.

[17] F. Durán. *A Reflective Module Algebra with Applications to the Maude Language*. PhD thesis, Universidad de Málaga, Spain, June 1999. Available at `http://maude.csl.sri.com/papers`.

[18] F. Durán, S. Lucas, C. Marché, J. Meseguer, and X. Urbain. Proving operational termination of membership equational programs. *Higher-Order and Symbolic Computation*, 21(1-2):59–88, 2008.

[19] 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, Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 313–319. Springer, 2008.

[20] F. Durán, S. Lucas, and J. Meseguer. Methods for proving termination of rewriting-based programming languages by transformation. *Electronic Notes in Theoretical Computer Science*, 248:93–113, 2009.

[21] 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, Proceedings*, volume 5749 of *Lecture Notes in Computer Science*, pages 246–262. Springer, 2009.

[22] F. Durán, S. Lucas, J. Meseguer, C. Marché, and X. Urbain. Termination of membership equational programs. In N. Heintze and P. Sestoft, editors, *Proceedings of the 2004 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation, PEPM 2004*, pages 147–158. ACM Press, 2004.

[23] F. Durán and J. Meseguer. Maude's module algebra. *Science of Computer Programming*, 66(2):125–153, April 2007.

[24] F. Durán and J. Meseguer. ChC 3: A coherence checker tool for conditional order-sorted rewrite Maude specifications. Available at `http://maude.lcc.uma.es/CRChC`, 2009.

[25] 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/CRChC`, 2009.

[26] F. Durán and J. Meseguer. A Church-Rosser checker tool for conditional order-sorted equational Maude specifications. In P. C. Ölveczky, editor, *Rewriting Logic and Its Applications - 8th International Workshop, WRLA 2010, Revised Selected Papers*, volume 6381 of *Lecture Notes in Computer Science*, pages 69–85. Springer, 2010.

[27] F. Durán and J. Meseguer. A Maude coherence checker tool for conditional order-sorted rewrite theories. In P. C. Ölveczky, editor, *Rewriting Logic and Its Applications - 8th International Workshop, WRLA 2010, Revised Selected Papers*, volume 6381 of *Lecture Notes in Computer Science*, pages 86–103. Springer, 2010.

[28] F. Durán, C. Rocha, and J. M. Álvarez. Tool interoperability in the maude formal environment. In A. Corradini, B. Klin, and C. Cîrstea, editors, *Algebra and Coalgebra in Computer Science, CALCO 2011, Proceedings*, volume 6859 of *Lecture Notes in Computer Science*, pages 400–406. Springer, 2011.

[29] A. Farzan and J. Meseguer. State space reduction of rewrite theories using invisible transitions. In M. Johnson and V. Vene, editors, *Algebraic Methodology and Software Technology, 11th International Conference, AMAST 2006, Proceedings*, volume 4019 of *Lecture Notes in Computer Science*, pages 142–157. Springer, 2006.

[30] H. Ganzinger. Order-sorted completion: the many-sorted way. *Theoretical Computer Science*, 89:3–32, 1991.

[31] J. Giesl and D. Kapur. Dependency pairs for equational rewriting. In A. Middeldorp, editor, *Rewriting Techniques and Applications, 12th International Conference, RTA 2001, Proceedings*, volume 2051 of *Lecture Notes in Computer Science*, pages 93–108. Springer, 2001.

[32] I. Gnaedig, C. Kirchner, and H. Kirchner. Equational completion in order-sorted algebras. *Theoretical Computer Science*, 72:169–202, 1990.

[33] J. Goguen, J.-P. Jouannaud, and J. Meseguer. Operational semantics of order-sorted algebra. In W. Brauer, editor, *Proceedings of the 1985 International Conference on Automata, Languages and Programming*, volume 194 of *Lecture Notes in Computer Science*, pages 221–231. Springer, 1985.

[34] 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.

[35] J. Hendrix, M. Clavel, and J. Meseguer. A sufficient completeness reasoning tool for partial specifications. In J. Giesl, editor, *Term Rewriting and Applications, 16th International Conference, RTA 2005, Proceedings*, volume 3467 of *Lecture Notes in Computer Science*, pages 165–174. Springer, 2005.

[36] J. Hendrix and J. Meseguer. Order-sorted equational unification revisited. In G. Kniesel and J. S. Pinto, editors, *Proceedings of The Ninth International Workshop on Rule-Based Programming (RULE 2008)*, 2008.

[37] 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.

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

[39] D. Kapur, P. Narendran, and F. Otto. On ground-confluence of term rewriting systems. *Information and Computation*, 86(1):14–31, 1990.

[40] 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.

[41] D. E. Knuth. Additional comments on a problem in concurrent programming control. *Communications of the ACM*, 9(5):321–322, 1966.

[42] L. Lamport. A new solution of Dijkstra's concurrent programming problem. *Communications of the ACM*, 17(8):453–455, 1978.

[43] S. Lucas. Context-sensitive computations in functional and functional logic programs. *Journal on Functional and Logic Programming*, 1(4):446–453, 1998.

[44] S. Lucas, C. Marché, and J. Meseguer. Operational termination of conditional term rewriting systems. *Information Processing Letters*, 95(4):446–453, 2005.

[45] C. Marché. Normalised rewriting and normalised completion. In *Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 394–403. IEEE Computer Society, 1994.

[46] U. Martin and T. Nipkow. Ordered rewriting and confluence. In M. E. Stickel, editor, *10th International Conference on Automated Deduction, CADE 1990, Proceedings*, volume 449 of *Lecture Notes in Computer Science*, pages 366–380. Springer, 1990.

[47] J. Meseguer. A logical theory of concurrent objects and its realization in the Maude language. In G. Agha, P. Wegner, and A. Yonezawa, editors, *Research Directions in Concurrent Object-Oriented Programming*, pages 314–390. The MIT Press, 1993.

[48] J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT 1997, Selected*

Papers, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61. Springer, 1998.

[49] J. Meseguer, M. Palomino, and N. Martí-Oliet. Equational abstractions. *Theoretical Computer Science*, 403(2-3):239–264, 2008.

[50] J. Meseguer, M. Palomino, and N. Martí-Oliet. Algebraic simulations. *Journal of Logic and Algebraic Programming*, 79(2):103–143, 2010.

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

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

[53] D. Plaisted. Semantic confluence tests and completion methods. *Information and Control*, 65:182–215, 1985.

[54] C. Rocha and J. Meseguer. Constructors, sufficient completeness and deadlock freedom of generalized rewrite theories. Technical report, Department of Computer Science in the University of Illinois at Urbana-Champaign, Urbana, May 2010.

[55] C. Rocha and J. Meseguer. Constructors, sufficient completeness and deadlock freedom of rewrite theories. In C. G. Fermüller and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17, Proceedings.*, volume 6397 of *Lecture Notes in Computer Science*, pages 594–609. Springer, 2010.

[56] G. Smolka, W. Nutt, J. Goguen, and J. Meseguer. Order-sorted equational computation. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures.*, volume 2 (Rewriting Techniques), chapter 10, pages 297–367. Academic Press, Inc., 1989.

[57] TeReSe, editor. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

[58] P. Viry. Rewriting: An effective model of concurrency. In C. Halatsis, D. Maritsas, G. Philokyprou, and S. Theodoridis, editors, *PARLE'94 Parallel Architectures and Languages Europe, 6th International PARLE Conference, Proceedings*, volume 817 of *Lecture Notes in Computer Science*, pages 648–660. Springer, 1994.

[59] P. Viry. Equational rules for rewriting logic. *Theoretical Computer Science*, 285(2):487–517, 2002.

[60] U. Waldmann. Semantics of order-sorted specifications. *Theoretical Computer Science*, 94(1):1–35, 1992.