# ChC 3:
# A Coherence Checker Tool for
# Conditional Order-Sorted Equational Maude Specifications

Francisco Durán
E.T.S.I. Informática
Universidad de Málaga, Spain
duran@lcc.uma.es

José Meseguer
Computer Science Dept.
University of Illinois at
Urbana-Champaign, IL, USA
meseguer@uiuc.edu

December 11, 2009

**Abstract**

This document explains the design and use of the coherence checker tool ChC 3, which checks whether a (possibly conditional) equational specification satisfies the coherence property modulo any combination of associativity, and/or commutativity, and/or identity axioms (combinations of associativity without commutativity are handled only under certain conditions). This tool can be used to prove the coherence property of order-sorted rewriting logic specifications in Maude [9, 6, 10]. The tool has been written entirely in Maude and is in fact an executable specification in rewriting logic [35] of the formal inference system that it implements. The fact that rewriting logic is reflective [3, 16], and that Maude efficiently supports reflective rewriting logic computations [5, 7] is systematically exploited in the design of the tool.

# Contents

# 1 Introduction

This document explains the design and use of a coherence checker, ChC, a proving tool to check whether a (possibly conditional) order-sorted rewriting specification modulo equational axioms satisfies the coherence property. It can be used to prove properties of rewriting specifications in Maude [9, 6, 10].

*Coherence* is a key executability requirement for rewrite theories, and therefore for Maude system modules. It allows reducing the, in general undecidable, problem of computing rewrites of the form $[t]_{E \cup A} \longrightarrow [t']_{E \cup A}$, with $A$ a set of equational attributes (associativity, commutativity, identity) for which matching algorithms exist, to the much simpler and decidable problem of computing rewrites of the form $[t]_A \longrightarrow [t']_A$. Coherence was studied in detail by P. Viry in [40].

The *Maude Coherence Checker* (ChC) provides a *decision procedure* for order-sorted system modules. Although the current version of the tool significantly improves the previous version [20], it still has some restrictions on the kind of modules that can handle. In particular, the Maude modules entered to the tool must satisfy the following requirements:

1. Only order-sorted specifications are allowed; modules cannot contain membership axioms.
2. Built-in modules are not supported, and built-in operations such as, e.g., `_==_` and `_=/=_` in the `BOOL` predefined module are not allowed. supported
3. The `owise` attribute cannot be used.

The tool checks that none of these features is used in the specifications entered, and gives an error message in case any of them is detected. Moreover, the equational part of the specification is assumed Church-Rosser and terminating, properties for which the Maude Church-Rosser Checker (CRC) [27] and the Maude Termination Tool (MTT) [23] could be used. If a module with membership axioms is entered, the ChC just discards them. The tool generates a set of *critical pairs*, whose coherence guarantees that of the entire system module [40]. It then checks whether each of these pairs is coherent.

The main improvements of the tool with respect to its previous version [20] are the following:

- Thanks to the availability in Maude 2.4 of unification modulo commutativity (C) and associativity and commutativity (AC) [10], and the new techniques for dealing with frequently used combinations of equational theories in [24], the ChC 3 can now handle modules with operators declared C, CU, AC, and ACU, that is, any combination of A, C, and U (including A without C in many frequent cases).
- The tool now discards those non-rewritable conditional critical pairs that are context-rewritable or unfeasible (see Section **??**).
- The tool is more robust, checks the required conditions on the input module and provides more detailed error messages.

For Maude system modules, which always have an initial model semantics, the weaker requirement of *ground coherence*, that is, coherence for ground terms, is enough. When the ChC tool cannot prove coherence—either because this fails, or because the input specification falls outside the class of decidable theories—it outputs a set of *proof obligations* associated with the critical pairs that it could not prove coherent. The user can then interact with the ChC tool to try to prove the ground coherence of the input system module by a constructor-based process of reasoning by cases. In the end, either:

1. all proof obligations are discharged and the module is shown to be ground coherent; or

2. proving ground coherence can be reduced to proving that the inductive validity of a set of equations follows from the equational part of the input system module, for which the ITP [3, 17] can be used; or

3. it is not possible to reduce some of the proof obligations to inductively proving some equations.

Case (3) may be a clear indication that the specification is not ground coherent, so that a new specification should be developed.

The purpose of the coherence checker tool discussed in this document is to check whether a rewrite theory is coherent or not. A coherence completion tool would attempt to complete a noncoherent specification so that it becomes coherent.

An important feature of this tool is that it is written entirely in Maude, and is in fact an *executable specification* in rewriting logic of the formal inference system that it implements. The tool treats equational specifications as *data* that is manipulated. The coherence checker computes critical pairs by inspecting the equations and rules in the original specification. This makes a *reflective* design—in which theories become data at the metalevel—ideally suited for the task. Indeed, the fact that rewriting logic is a reflective logic [3, 16], and that Maude efficiently supports reflective rewriting logic computations is systematically exploited in the tool. The same reflective design has been followed for other Maude tools, as, e.g., the inductive theorem prover [3, 11, 12, 13, 15], the Knuth-Bendix completion tool [22, 11, 12, 13], the Church-Rosser checker tool [27, 26, **?**, 12, 13, 15]. Real-Time Maude [37, 15], the Maude Termination Tool (MTT) [23, 15], and the Sufficient-Completeness Checker (SCC) [31, 15].

### 1.1 Reflective Design of the Tool

The very high level of abstraction at which the tool has been developed has made it relatively easy for us to build it, makes understanding its implementation much easier, and will make its maintenance and future extensions much easier than if a conventional implementation, say in C++ or Java, had instead been chosen. Thanks to the high performance of the Maude engine [7, 8, 4], these important benefits in ease of development, understandability, maintainability, and extensibility are achieved with a reasonable performance, in spite of using reflection, sophisticated rewriting modulo associativity and associativity-commutativity, and a combination of built-in (AC and C) unification algorithms and theory transformations to deal with identities and the A without C case. We expect that a future version of Maude will also support U-unification in a built-in way, which will further increase ChC's performance.

The design of the present tool exploits and illustrates the logical framework capabilities of rewriting logic by formally specifying the *inference system* of the tool as a rewrite theory [34, 33]. The fact that the ChC reasons about *rewriting theories* is not a restriction of the general framework capabilities: inference systems for reasoning about specifications in any other logic could be represented just as easily.

The coherence checker tool has a simple design. Let $T$ be a system module, which has an initial algebra semantics. Such module, that we want to check whether it is coherent, is at the object level. An inference system $\mathcal{C}$ for checking the coherence property uses $T$ as a data structure—that is, it actually uses its metarepresentation $\overline{T}$—and therefore is a rewrite theory at the metalevel. The checking process can be described in a purely functional way, that is, the

metalevel theory specifying it is in fact an equational theory in Maude sublanguage of functional modules.

As other tools in the Maude formal environment [15], the ChC has been implemented as an extension of Full Maude [25, 19]. Details on how to extend Full Maude in different forms can be found in, e.g., [19, 21, 28]. Following these techniques, the ChC has been integrated within the Full Maude environment, to allow checking of modules defined in Full Maude and to get a much more convenient user interface. Of course, it would have been possible to define an interface for the tool without integrating it with Full Maude. Since all the infrastructure built for Full Maude can be used by itself, just by selecting functions from that infrastructure in the needed modules, any of the two possibilities can give rise to an interface in a very short time. However, integrating the specifications of Full Maude and of the ChC we not only have such a needed infrastructure, but in addition we can, for example, check the coherence property of any module in Full Maude's database. We can therefore use the tool on any module accepted by Full Maude, including structured modules, parameterized modules, etc. We still have, of course, the restrictions mentioned in the previous sections, that is, the module has to be order-sorted, the specification does not contain any built-in function, and has already been proved Church-Rosser and terminating using another tool.

### 1.2 The Coherence Check

Traditionally, a rewrite system is a set of directed equations used to compute by repeatedly replacing subterms of a given formula with equal terms until the simplest possible form is obtained. This interpretation of a rewrite system gives an equational semantics to a rewrite system, and also a way of executing equational specifications by rewriting. But rewriting is also useful for specifying nonequational relations, such as transitions between states or reduction steps. Rewriting logic [35] suggests keeping all rules with an equational interpretation as a set $E$ of equations, and considering the remaining rules as defining rewrite steps over classes modulo $E$.

A rewrite logic signature is an equational specification. But, rewriting logic is parameterized by the choice of its underlying equational logic. For Maude [6], the underlying equational logic is membership equational logic, so that signatures are of the form $(\Omega, E)$, where $\Omega = (K, \Sigma, S)$ is a membership equational logic signature and $E$ is a set of (conditional) membership axioms and equations. Such a signature $(\Omega, E)$ makes explicit the set of equations in order to emphasize that rewriting will operate on congruence classes of terms *modulo* $E$. This is precisely what Maude does, using the equational attributes given in operation declarations (associativity, commutativity, and identity) to rewrite modulo such axioms (see [6]).

A rewrite theory has both rules and equations, so that rewriting is performed modulo such equations. However, this does not mean that the Maude implementation must have a matching algorithm for each equational theory that a user might specify, which is impossible, since matching modulo an arbitrary theory is undecidable. What Maude instead requires for rewrite theories in system modules is that:

- The equations are divided into a set $A$ of structural axioms, for which matching algorithms exist in the Maude implementation, and a set $E$ of equations that are Church-Rosser and terminating modulo $A$; that is, the equational part must be equivalent to a functional module. For some axioms $A$, this can be checked using a Church-Rosser checker as the one presented in [27, 26, 19, 14].

- The rules $R$ in the module are *coherent* [36, 40] with the equations $E$ modulo $A$. This means

that appropriate critical pairs exist between rules and equations, allowing us to intermix rewriting with rules and rewriting with equations without losing rewrite computations by failing to perform a rewrite that would have been possible before an equational deduction step was taken. In this way, we get the effect of rewriting modulo $E \cup A$ with just a matching algorithm for $A$. In particular, a simple strategy available in these circumstances is to always reduce to canonical form using $E$ before applying any rule in $R$. This is precisely the strategy adopted by the Maude interpreter.

Therefore, for computational purposes it becomes very important to know whether a given Church-Rosser and terminating specification is indeed ground-coherent. A nontrivial question is how to best support this with adequate tools. One can prove the Church-Rosser property of his/her (possibly conditional) Maude equational specification by using the Maude Church-Rosser checker (CRC) [27] and its operational termination by using the MTT tool [23]. A thornier issue is what to do for establishing the ground-coherence property for a Church-Rosser and terminating specification. The problem is that a specification with an initial algebra semantics can often be ground-coherent even though some of its critical pairs may not be rewritable. That is, the specification can often be ground-coherent without being coherent for arbitrary terms with variables. In such a situation, blindly applying a completion procedure that is trying to establish the coherence property for arbitrary terms may be both quite hopeless—the procedure diverges or gets stuck, and even with success may return a specification that is quite different from the original one—and even unnecessary, if the specification was already ground-coherent.

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

The present ChC tool only accepts order-sorted conditional specifications, where each of the operation symbols has either no equational attributes, or any combination of associativity/commutativity/identity. Furthermore, it is assumed that such specifications do not contain any built-in function, do not use the `owise` attribute, and that they have already been proved Church-Rosser and terminating. The tool attempts to establish the ground-coherence property *modulo* the equational axioms specified for each of the operators by checking a sufficient condition. Therefore, the tool's output consists of a set of critical pairs that must be shown ground-rewritable.

### 1.3 Use of the Tool and Commands Available

The current version of the tool is ChC 3.[1] The tool is entirely written in Maude as an extension of Full Maude. ChC 3 works on Maude 2.4 (alpha 92a)[2] and Full Maude 2.4o.[3] To start the tool one just needs to load the Maude code of the ChC after starting Full Maude. If `maude` is your Maude executable, the Full Maude file is `full-maude.maude`, and the ChC program is in `crchc3.maude`, you can start the ChC as follows:

```
$ maude full-maude crchc3

           \||||||||||||||||||/
         --- Welcome to Maude ---
           /||||||||||||||||||\
      Maude alpha92a built: Nov 12 2009 18:47:47
      Copyright 1997-2009 SRI International
         Tue Dec  8 03:45:12 2009

      Full Maude 2.4o December 5th 2009

      Church-Rosser Checker 3 - December 7th 2009
      Coherence Checker 3 - December 7th 2009

set include BOOL off

set include TRUTH-VALUE on
```

Notice that since the modules entered to the tool cannot use built-in modules, at start up, the ChC drops the default inclusion of the `BOOL` module. The `TRUTH-VALUE` module only contains sort and constant declarations (cf. [10]) and therefore is fine.

The commands available in the CRC tool are the following:

**(help .)** shows the list of commands available in the tool.

**(check coherence .)** checks the coherence property of the default module.

**(check coherence <module-name> .)** checks the coherence property of the specified module. This feature can be used to check the coherence property of any (Core) Maude or Full Maude module specified in the same Maude session (see below).

**(check ground coherence .)** checks the ground coherence property of the default module.

**(check ground coherence <module-name> .)** checks the ground coherence property of the specified module. This feature can be used to check the coherence property of any (Core) Maude or Full Maude module specified in the same Maude session (see below).

**(show ChC critical pairs .)** shows the reduced form of the critical pairs that could not be rewritten in the last `check coherence` command. Those rewritten are not shown.

**(show all ChC critical pairs .)** shows the unreduced form of all critical pairs computed in the last `check coherence` command.

[1] CRC 3 and ChC 3 are distributed together. They are a available at `http://maude.lcc.uma.es/CRChC`.
[2] Maude 2.4 is available at `http://maude.cs.uiuc.edu`.
[3] Full Maude 2.4o is available at `http://maude.lcc.uma.es/FullMaude`.

Their use and the syntax of the results is described in the following sections.

Any module available in Full Maude that satisfies the above-mentioned requirements (1)-(4) can be checked by the coherence checker. This includes any (Core) Maude module satisfying requirements (1)-(4) that has been entered *before* loading (or re-initializing) Full Maude and ChC 3, which can also be checked using the `check coherence` <`module-name`> or `check ground coherence` <`module-name`> commands.

Once ChC 3 has been started, we can enter a module and check whether it satisfies the coherence property as follows.

```
Maude> (mod IDENTITY is
          sort S .
          op 0 : -> S .
          op _+_ : S  S -> S [comm] .
          op f : S -> S .
          vars X Y : S .
          eq X + 0 = X .
          rl f(X + Y) => f(X) + f(Y) .
        endm)

Introduced module IDENTITY

Maude> (check coherence .)

Coherence checking of IDENTITY
Coherence checking solution:
The following critical pairs cannot be rewritten:
  cp f(X:S)
    => f(0)+ f(X:S).
```

We will discuss the output above in the following sections. For now, let us say that the tool gives as result a set of critical pairs, some of which may be conditional. The critical pairs given come from the overlapping of the equations and rules in the specification being checked. If labels are used, each critical pair contains the labels of the equations that generated it.

```
Maude> (mod IDENTITY is
          sort S .
          op 0 : -> S .
          op _+_ : S  S -> S [comm] .
          op f : S -> S .
          vars X Y : S .
          eq [e1] : X + 0 = X .
          rl [r1] : f(X + Y) => f(X) + f(Y) .
        endm)

Introduced module IDENTITY

Maude> (check coherence .)

Coherence checking of IDENTITY
Coherence checking solution:
The following critical pairs cannot be rewritten:
  cp for e1 and r1
    f(X:S)
    => f(0)+ f(X:S).
```

As in Full Maude, if we want to check the Church-Rosser property of a (Core) Maude module, we must select the `CRChC` module and restart the loop [10, Section 15.2].

```
Maude> set include BOOL off .

Maude> set include TRUTH-VALUE on .

Maude> mod IDENTITY is
         sort S .
         op 0 : -> S .
         op _+_ : S  S -> S [comm] .
         op f : S -> S .
         vars X Y : S .
         eq [e1] : X + 0 = X .
         rl [r1] : f(X + Y) => f(X) + f(Y) .
       endm

Maude> select CRChC .

Maude> loop init .

  Church-Rosser Checker 3 - December 7th 2009
  Coherence Checker 3 - December 7th 2009

Maude> (check Church-Rosser IDENTITY .)

Coherence checking of IDENTITY
Coherence checking solution:
The following critical pairs cannot be rewritten:
  cp for e1 and r1
    f(X:S)
    => f(0)+ f(X:S).
```

After introducing some basic formal concepts and results underlying the tool's design, we give recommendations for its use and some examples.

## Acknowledgements

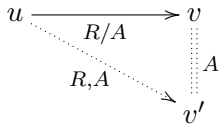
We are very thankful for the many fruitful discussions with the other members of the Maude team, Manuel Clavel, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, and Carolyn Talcott, with whom we have designed and built the Maude language and system. We are very thankful to Santiago Escobar, for the narrowing infrastructure he developed for the Maude-NPA tool, part of which has been used in the CRC and ChC tools. The first version of this tool, together with an inductive theorem prover—to prove theorems about such specifications—was developed as part of the Cafe project [1]. It was integrated within the Full Maude environment and also, thanks to the efforts of our colleagues in Japan, within the overall Cafe environment [29]. Through that second integration, both tools could be used to prove formal properties of order-sorted equational specifications in CafeOBJ [18]. We are particularly grateful to Santiago Escobar for his development of the narrowing modulo axioms feature in Maude, which is used in an essential way as a component of the CRC 3 reflective implementation.

# 2 Coherent Order-Sorted Specifications Modulo Axioms

## 2.1 Conditonal Rewriting Modulo Linear and Regular Axioms $A$

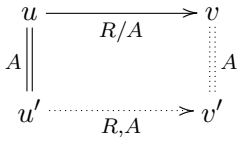
Given an order-sorted rewrite theory $\mathcal{R} = (\Sigma, A, R)$, where $A$ is a collection of unconditional equational axioms of the form $u = v$ that are *linear* (no repeated variables in either $u$ or $v$), and *regular* ($vars(u) = vars(v)$), we define the relation $\rightarrow_{R/A}$, either by the inference system of rewriting logic (see [2]), 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 classes $[u]_A$ and $[v]_A$ 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 [38]) $\rightarrow_{R,A} = \bigcup_n \rightarrow_{R,A,n}$ 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 $\sigma$ such that $u|_p =_E l\sigma$, $v = u[r\sigma]_p$, and $\forall i, u_i\sigma \rightarrow^*_{R,A,n} w_i$ with $w_i =_E v_i\sigma$. Of course, $\rightarrow_{R,A} \subseteq \rightarrow_{R/A}$. The important question is the *completeness* question: can any $\rightarrow_{R/A}$-step be simulated by a $\rightarrow_{R,A}$-step? We say that $\mathcal{R}$ satisfies the *$A$-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[4] (or just *coherence* when $A$ is understood) property:

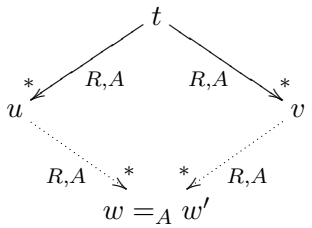



If a theory $\mathcal{R}$ is not coherent, we can try to make it so by completing the set of rules $R$ to a set of rules $\widetilde{R}$ by a Knuth-Bendix-like completion procedure that computes critical pairs between equations in $A$ and rules in $R$ (see, e.g., [32], [40] for the *strong* coherence completion that we use here, and [30] for the equivalent notion of extension completion). As we will further discuss in Section **??**, for theories $A$ that are combinations of $A$, $C$, $LU$, and $RU$ axioms, the coherence completion procedure always terminates and has a very simple description.

We say that $\mathcal{R} = (\Sigma, A, R)$ is *$A$-confluent*, resp. *$A$-terminating*, if the relation $\rightarrow_{R/A}$ is confluent, resp. terminating. If $\mathcal{R}$ is $A$-coherent, then $A$-confluence is equivalent to asserting

[4]Note that the assumption of $A$ being regular and linear is essential for one $\rightarrow_{R/A}$-step to exactly correspond to one $\rightarrow_{R,A}$-step. For this reason, some authors (e.g., [32, 40]) call conditions as the one above *strong coherence*, and consider also weaker notions of coherence.

that, for any $t \rightarrow^*_{R,A} u$, $t \rightarrow^*_{R,A} v$, we have:




and $A$-termination is equivalent to the termination of the $\rightarrow_{R,A}$ relation.

The fact that we are performing *order-sorted* rewriting makes one more requirement necessary. When $A$-matching a subterm $t|_p$ against a rule's left-hand side to obtain a matching substitution $\sigma$, we need to check that $\sigma$ is well-sorted, that is, that if a variable $x$ has sort $s$, then the term $x\sigma$ has also sort $s$. This may however fail to be the case even though there is a term $w \in [x\sigma]_A$ which does have sort $s$. We call an order-sorted signature $A$-*preregular* if the set of sorts $\{s \in S \mid \exists w' \in [w]_A \text{ s.t. } w' \in \mathcal{T}_\Sigma(\mathcal{X})_s\}$ has a least upper bound, denoted $ls[w]_A$ which can be effectively computed.[5] Then we can check the well-sortedness of the substitution $\sigma$ not based on $x\sigma$ above, but, implicitly, on all the terms in $[w]_A$.

Yet another property required for the good behavior of confluent and terminating rewrite theories modulo $A$ is their being $A$-*sort-decreasing*. This means that $\mathcal{R}$ is $A$-preregular, and for each rewrite rule $l \rightarrow r$ if $\bigwedge_i u_i \rightarrow v_i$, and for each specialization substitution $\nu$ such that $\bigwedge u_i\sigma \rightarrow^*_{R,A} w =_A v_i$ we have $ls[r\nu]_A \leq ls[l\nu]_A$.

As we said above, for $\rightarrow_{R,A}$ to be decidable we need an $A$-matching algorithm. Therefore, we will consider the set of equations as a union $E \cup A$ with $A$ a set of axioms for which there exists a matching algorithm (as associativity (A), commutativity (C), and identity (U)), and $E$ the rest.

## 2.2 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) $\phi$ is the frozenness information.

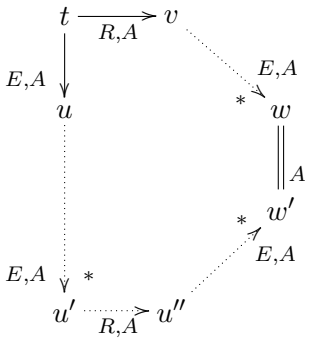
(2) $(\Sigma, E \cup A)$ is an order-sorted equational theory with possibly conditional equations, which can be converted into 3-CTRS rules that are *strongly deterministic* and *operationally terminating* modulo $A$. Furthermore, the equations $E$ are *confluent* modulo $A$. Also, the axioms in $A$ are a collection of regular and linear unconditional equational axioms and all are at the *kind level*, i.e., each connected component in the poset $(S, \leq)$ of sorts has a top element, and the variables in the axioms $A$ all have such sorts.

(3) $R$ is a collection of rewrite rules $l \rightarrow r \; if \; C$ where $C$ is an *equational condition*, which again can be turned into a 3-CTRS condition of the form $u_1 \rightarrow_E v_1 \wedge \ldots \wedge u_n \rightarrow_E v_n$ with the $v_1, \ldots, v_n$ strongly $E, A$-irreducible.

(4) Both the equations $E$ and the rules $R$ are $A$-*coherent*. Therefore, the relations $\rightarrow_{R/A}$ (resp. $\rightarrow_{E/A}$) and $\rightarrow_{R,A}$ (resp. $\rightarrow_{E,A}$) essentially coincide.

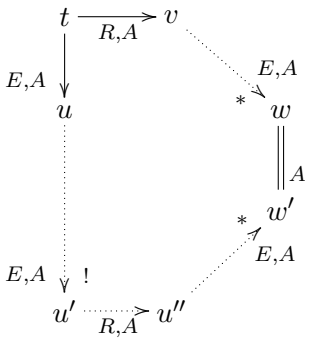
[5] The Maude system automatically checks the $A$-preregularity of a signature $\Sigma$ for $A$ any combination of $A$, $C$, $LU$, and $RU$ axioms (see [9, Chapter 22.2.5]).

**Definition 1** *A rewrite theory $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$ satisfying (1)-(4) above is called* coherent *(resp.* ground coherent*) iff for each $\Sigma$-term $t$ (resp. ground $\Sigma$-term $t$) such that $t \rightarrow_{E,A} u$, and $t \rightarrow_{R,A} v$ we have*




*(C)*

*Likewise, $\mathcal{R}$ is called* locally coherent *(resp.* ground locally coherent*) iff for each $\Sigma$-term $t$ (resp. ground $\Sigma$-term $t$) such that $t \rightarrow_{E,A} u$, and $t \rightarrow_{R,A} v$ we have*




*(LC)*

**Theorem 2** *$\mathcal{R}$ is coherent (resp. ground coherent) iff $\mathcal{R}$ is locally coherent (resp. locally ground coherent).*

Since for all terms $t$, $t$ is coherent iff $t$ is locally coherent, we can approach the verification of coherence for $\mathcal{R}$ as follows:

We can reason by cases on the situations $\overset{t}{\underset{E,A\ u}{\swarrow}}\ \ \underset{v\ R,A}{\searrow}$ 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.

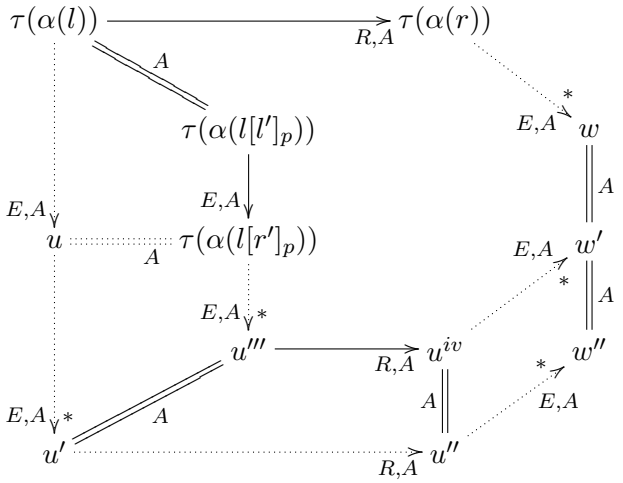
**Definition 3** *Given rewrite rules with disjoint variables $l \rightarrow r\ if\ C$ in $R$ and $l' \rightarrow r'\ if\ C'$ in $E$, their set of conditional critical pairs modulo $A$ is defined as usual: either we find a non-variable position $p$ in $l$ such that $\alpha \in Unif_A(l|_p, l')$ and then we form the conditional critical pair*

$$\begin{array}{lll} \alpha(C) \wedge \alpha(C') & \Rightarrow & \alpha(l[l']_p) \underset{A}{=\!=} \alpha(l) \underset{R}{\longrightarrow} \alpha(r) \\ & & \Big\downarrow{\scriptstyle E} \\ & & \alpha(l[r']_p) \end{array} \qquad (I)$$

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

$$\begin{array}{lll} \alpha(C) \wedge \alpha(C') & \Rightarrow & \alpha(l') \underset{A}{=\!=} \alpha(l'[l]_{p'}) \underset{R}{\longrightarrow} \alpha(l'[r]_{p'}) \\ & & \Big\downarrow{\scriptstyle E} \\ & & \alpha(r') \end{array} \qquad (II)$$

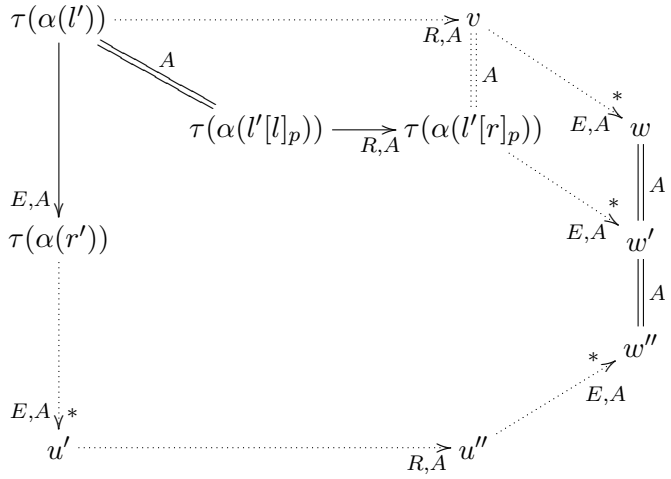
We say that a critical pair of type $(I)$ is *joinable* iff for any substitution $\tau$ such that $E \cup A \vdash \tau\alpha(C) \wedge \tau\alpha(C')$ we then have




(I)

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 $\tau$ such that $E \cup A \models \tau\alpha(C) \wedge \tau\alpha(C')$ we then have

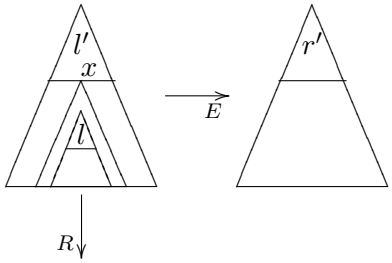



(I)

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

Of course, joinability of all CCPs is a *necessary* condition for coherence. The challenge now is to find a set of *sufficient conditions* for coherence that includes the joinability of CCPs.

Specifically, non-overlapping situations between equations and rules require additional conditions. Let $\mathcal{R} = (\Sigma, E \cup A, R, \emptyset)$ be our given rewrite theory. In the case of *coherence* checking, we need to worry about non-overlapping situations of the form $l' \rightarrow_E r' \; if \; C'$ in $E$ and $l \rightarrow_R r \; if \; C$ in $R$ where we have




This case can be problematic in two related ways: (1) $l' \rightarrow_E r' \; if \; C'$ unconditional but not linear, or (2) $l' \rightarrow_E r' \; if \; C'$ conditional. The problem with (1) is well-understood since [40]. The problem with (2) was also mentioned by Viry in [40]; it has to do with the fact that the satisfiability of the condition $C'$ in an equation $l' \rightarrow_E r' \; if \; C'$ *depends* on the substitution $\Theta$ (may hold or not depending on the given $\Theta$). But since $R$ *rewrites* the substitution $\Theta$, we do *not know if* $C'$ *will hold anymore after a one-step rewrite with the rule* $l \rightarrow_R r \; if \; C$.[6]

**Theorem 4** *Given* $\mathcal{R}$ *as above, then if:*

*(i) all CCPs are joinable and*

*(ii) for any equation* $l' \rightarrow r'$ *if* $C'$ *in* $E$*, for each* $x \in vars(l')$ *such that* $x$ *is non-frozen in* $l'$*, then* either

  *(a)* $x$ *is such that* $x \notin vars(C')$*,* $x$ *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 $\rightarrow_{R,A}$ is possible for terms of such sort $s$.

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

Condition $(ii).(b)$ of Theorem 4 requires a fixpoint calculation. The following algorithm can check that situations where a non-frozen variable $x \in vars(l')$ in a left-hand side of an equation fails to satisfy $(ii) - (a)$ or $(ii) - (b)$ do *not* arise as follows: Given an order-sorted signature $(\Sigma, S, \leq)$ and a sort $s \in S$, call the *closure* $\bar{s} \subseteq S$ of $s$ to be the set defined inductively by:

1. $s \in \bar{s}$
2. $s' \in \bar{s} \wedge s'' \leq s' \Rightarrow s'' \in \bar{s}$
3. $s' \in \bar{s} \wedge f : s_1 \ldots s_n \rightarrow s' \in \Sigma \Rightarrow s_1 \ldots s_n \in \bar{s}$

Then the algorithm does the following. For each $l' \rightarrow_E r' \; if \; C'$ in $E$, and for each non-frozen variable $x \in vars(l')$:

1. If $x \notin vars(C')$ and $x$ is also non-frozen in $r'$, and $x$ is linear in $l'$ and $r'$, this variable is ok.

[6] Note that we can view cases of unconditional $l \rightarrow r$ 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`.

2. Let $x_1 : s_1 \ldots x_n : s_n$ be the remaining non-frozen variables in $vars(l')$. Then, if for each $l \to t$ *if* $C$ in $R$ and for each $s_i$, $1 \leq i \leq n$, we have $\bar{s}_i \cap \{LS(\rho(l)) \mid \rho$ *variable specialization*$\} = \emptyset$, then the disjunction of conditions $(ii) - (a)$ and $(ii) - (b)$ in the Theorem holds.

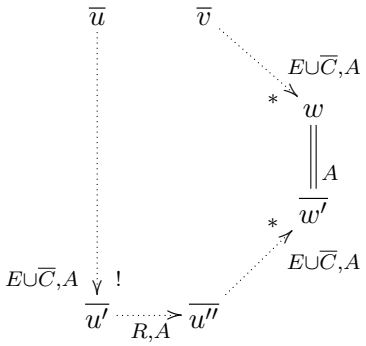
3. Otherwise, there is a substitution $\theta$ such that we can rewrite with $R$ inside the equation's substitution $\theta$, where either the rewrite happens in $\theta(x)$, with $x \in vars(C')$, so that the equation's condition $C'$ may *fail* after this $R$-rewrite, blocking coherence, or either frozenness in $r'$, or nonlinearities in $l'$ or $r'$, may again block coherence.

### 2.3 Context-joinability and unfeasibility of CCPs

**Definition 5** *Given a rewrite theory $\mathcal{R} = (\Sigma, E \cup A, R)$, a non-joinable conditional critical pair*

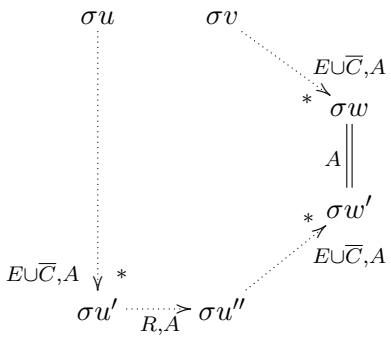
$C \Rightarrow u \to^1 v$ *(coming from a critical pair* $C \Longrightarrow$ $\begin{array}{ccc} t & \xrightarrow[R,A]{} & v \\ {\scriptstyle E,A}\downarrow & & \\ u & & \end{array}$ *) is* context-joinable *if and only if*

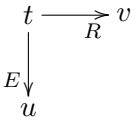
*in the extended rewrite theory $\mathcal{R}_C = (\Sigma \cup \overline{X}, E \cup \overline{C} \cup A, R)$ we have:*




*where $\overline{C}$ is the result of replacing in the condition $C$ each variable $x$ by a new constant $\overline{x}$, and $\overline{X}$ is the set of such new constants. Thus, given a term $t$, $\overline{t}$ results from replacing each variable $x \in vars(C)$ by the corresponding constant $\overline{x}$.*

**Lemma 1** *If the conditional critical pair $C \Rightarrow u \to^1 v$ is context joinable, then for all substitution $\sigma$ such that $\sigma C$ holds we have*



*and therefore, the coherence property holds for the critical pair* $C \Longrightarrow$ 

.

(1) (1)

unfeasability

### 2.4 The ground coherence case

Assume that $\Sigma$ has a sub-signature of constructors $\Omega$ that has been verified to be *sufficiently complete* modulo $A$. Then we can view each $f \in \Sigma$ with a different syntactic form from $\Omega$ 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:

(i) constructor equations, and

(ii) equations $f(t_1, \ldots, t_n) \to r$ *if* $C$ in $E$ with $f \in \Sigma - \Omega$, and with $f$ having the $U$, $LU$, or $RU$ attributes and such that, assuming $A = \vec{Ids} \cup B$, there is a $\vec{Ids}, B$-variant $\hat{l} \to \alpha(r)$ *if* $\hat{C}$ of the above rule such that $\hat{l}$ has a *non-frozen variable*.

Therefore, for ground coherence under the assumption of frozenness of defined symbols, we only have to check condition $(ii)$ in Theorem 4 on equations of types (i) and (ii) above.

Furthermore, for those CCPs for which we have not been able to check context joinability, we can guarantee their *inductive ground joinability* if for $w = u \downarrow_{E,A}$ and for each rule $(\forall Y)\lambda : l \to t$ *if* $C$ in $R$ such that in the theory

$$\tilde{\mathcal{R}}_{\overline{\alpha(C)},\overline{\alpha(C')}} = (\Sigma \cup \overline{X} \cup \overline{Y_{0,\lambda}}, A, E \cup \overline{\alpha(C)} \cup \overline{\alpha(C')}, \{l \to \overline{r}^{Y_{0,\lambda}} \mid \lambda : l \to r \text{ if } C \text{ in } R\})$$

where $Y_{0,\lambda} = vars(r) - vars(l)$ for a rule $\lambda : l \to t$ *if* $C$ in $R$, and $\overline{r}^{Y_{0,\lambda}}$ denotes the term $r$ with all variables in $Y_{0,\lambda}$ are made constants, we can prove $\overline{w} \to^1_{R,A} v'_i$ for some substitution $\overline{\theta}_i$ for the variables of $l \to \overline{r}$ for some such rule. Then inductive ground joinability amounts to proving the inductive theorem:

$$E \cup A \vdash_{ind} (\alpha(C) \wedge \alpha(C')) \Rightarrow (\theta_1 C_1 \wedge v_1 = v) \vee \ldots \vee (\theta_n C_n \wedge v_n = v).$$

## 3 How to Use the Coherence Checker

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

We assume a context of use quite different from the usual context for *completing* a rewriting theory. The starting point for completing a theory is a rewriting theory that is *not* coherent. A Knuth-Bendix-like completion process then attempts to make it so by *automatically adding* new rules.

In our case, however, we assume that the user has developed an *executable specification* of his/her intended system with an initial model semantics, and that this specification has already

been *tested* with examples, so that the user is in fact confident that the specification is *ground-coherent*, and wants only to check this property with the tool.

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

An important methodological question is what to do, or not do, with these proof obligations. As the examples that we discuss illustrate, 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. In any case, it will modify the user's specification in ways that can make it difficult for the user to recognize the final result, if any, as intuitively equivalent to the original specification.

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

### 3.1 Rewriting Modulo Commutativity with an Identity Equation

Our first example is based on one given by Viry in [39], in which we consider a specification with a constant `0`, a unary operator `f`, and a commutative binary operator `_+_`. We define the operator `_+_` with `0` as its identity element by including the equation `X + 0 = X`, which must be oriented from left to right in order to avoid nontermination. Let us consider then the following specification.

```
(set include BOOL off .)

(mod IDENTITY is
   sort S .
   op 0 : -> S .
   op _+_ : S  S -> S [comm] .
   op f : S -> S .
   vars X Y : S .
   eq [e1] : X + 0 = X .
   rl [r1] : f(X + Y) => f(X) + f(Y) .
 endm)
```

The equational part of the specification is terminating and Church-Rosser, as can be proved with MTT and CRC. However, $R/A$ is not coherent with $E/A$, as the tool shows.

```
Maude> (check coherence IDENTITY .)
```

```
Coherence checking of IDENTITY
Coherence checking solution:
The following critical pairs cannot be rewritten:
  cp for e1 and r1
    f(X:S)
    => f(0)+ f(X:S).
```

The result of the tool suggests the addition of the critical pair as a rule.

```
(mod IDENTITY-2 is
   sort S .
   op 0 : -> S .
   op _+_ : S  S -> S [comm] .
   op f : S -> S .
   vars X Y : S .
   eq [e1] : X + 0 = X .
   rl [r1] : f(X + Y) => f(X) + f(Y) .
   rl [r2] : f(X) => f(0) + f(X) .
 endm)
```

The tool now confirms the coherence of the specification.

```
Maude> (check coherence IDENTITY-2 .)

Coherence checking of IDENTITY-2
Coherence checking solution:
All critical pairs have been rewritten and all equations are non-constructor.
The specification is ground coherent.
```

### 3.2 Readers and Writers

Consider the following specification of a readers-writers system borrowed from [9].

```
(set include BOOL off .)

(mod READERS-WRITERS is
   sort Nat .
   op 0 : -> Nat [ctor] .
   op s : Nat -> Nat [ctor] .

   sort Config .
   op <_,_> : Nat Nat -> Config [ctor] .  --- readers/writers

   vars R W : Nat .

   rl < 0, 0 > => < 0, s(0) > .
   rl < R, s(W) > => < R, W > .
   rl < R, 0 > => < s(R), 0 > .
   rl < s(R), W > => < R, W > .
 endm)
```

A state is represented by a tuple `< R, W >` indicating the number `R` of readers and the number `W` of writers accessing a critical resource. Readers and writers can leave the resource at any time,

but writers can only gain access to it if nobody else is using it, and readers only if there are no writers.

Taking `< 0, 0 >` as the initial state, this simple system satisfies two important invariants, namely:

- *mutual exclusion*: readers and writers never access the resource simultaneously: only readers or only writers can do so at any given time.
- *one writer*: at most one writer will be able to access the resource at any given time.

We first introduce an *explicit* representation of those invariants and then an equational abstraction preserving such invariants. Since at present the ChC tool does not handle predefined modules like `BOOL`, we use instead a sort `NewBool`.

```
(mod READERS-WRITERS-PREDS is
   protecting READERS-WRITERS .
   sort NewBool .
   ops tt ff : -> NewBool [ctor] .
   ops mutex one-writer : Config -> NewBool [frozen] .
   eq mutex(< s(N:Nat), s(M:Nat) >) = ff .
   eq mutex(< 0, N:Nat >) = tt .
   eq mutex(< N:Nat, 0 >) = tt .
   eq one-writer(< N:Nat, s(s(M:Nat)) >) = ff .
   eq one-writer(< N:Nat, 0 >) = tt .
   eq one-writer(< N:Nat, s(0) >) = tt .
 endm)
```

We can now define our abstraction, in which all the states having more than one reader and no writers are identified with the state having exactly one reader and no writer.

```
(mod READERS-WRITERS-ABS is
   including READERS-WRITERS-PREDS .
   including READERS-WRITERS .
   eq < s(s(N:Nat)), 0 > = < s(0), 0 > .
 endm)
```

In order to check both the executability and the invariant-preservation properties of this abstraction, since we have no equations with either `tt` or `ff` in their lefthand side, we now just need to check:

(i) that the equations in both `READERS-WRITERS-PREDS` and `READERS-WRITERS-ABS` are ground confluent, sort-decreasing, and terminating;

(ii) that the equations in both `READERS-WRITERS-PREDS` and `READERS-WRITERS-ABS` are sufficiently complete; and

(iii) that the rules in both `READERS-WRITERS-PREDS` and `READERS-WRITERS-ABS` are ground coherent with respect to their equations.

Regarding termination, since the equations of `READERS-WRITERS-ABS` contain those of the module `READERS-WRITERS-PREDS`, it is enough to check the termination of the equations in the former. The MTT tool, using the AProVE termination tool, checks this automatically. We

can also check confluence of the equations by invoking the CRC. Sufficient completeness can be proved using the SCC.

This leaves us with task (3), where the Coherence Checker tool responds as follows:

```
Maude> (check ground coherence READERS-WRITERS-PREDS .)

Coherence checking of READERS-WRITERS-PREDS
Coherence checking solution:
All critical pairs have been rewritten and all equations are non-constructor.
The specification is coherent.


Maude> (check ground coherence READERS-WRITERS-ABS .)

Coherence checking of READERS-WRITERS-ABS
Coherence checking solution:
The following critical pairs cannot be rewritten:
  cp < s(0), 0 >
    => < s(#1:Nat), 0 > .
```

Ground coherence means that all ground instances of this pair can be rewritten by a one-step rewrite up to canonical form by the equations. We can reason by cases and decompose this critical pair into two:

```
cp < s(0), 0 > => < s(0), 0 > .
cp < s(0), 0 > => < s(s(N:Nat)), 0 > .
```

Using the `reduce` command we can check that the canonical form of the term `< s(s(N:Nat)), 0 >` is `< s(0), 0 >`. Therefore, all we need to do is to check that `< s(0), 0 >` rewrites to *itself* (up to canonical form) in one step. We can do this check by giving the command:

```
Maude> (search in READERS-WRITERS-ABS : < s(0), 0 > =>1 X:Config .)

Solution 1
X:Config --> < s(0),0 >

Solution 2
X:Config --> < 0,0 >

No more solutions.
```

### 3.3 Bakery

Our following example is a simplified version of Lamport's bakery protocol, also borrowed from [9]. This is an infinite-state protocol that achieves mutual exclusion between processes by the usual method common in bakeries and deli shops: there is a number dispenser, and customers are served in sequential order according to the number that they hold. A simple Maude specification for the case of two processes is as follows:

```
(fmod MBOOL is
   op _and_ : Bool Bool -> Bool [assoc comm prec 55] .
   op _or_ : Bool Bool -> Bool [assoc comm prec 59] .
```

```
   op not_ : Bool -> Bool [prec 53] .
   vars A B C : Bool .
   eq true and A = A .
   eq false and A = false .
   eq not true = false .
   eq not false = true .
   eq not not A = A .
   eq A or B = not (not A and not B) .
 endfm)

(fmod NATURAL is
   pr MBOOL .
   sort Nat .
   op 0 : -> Nat [ctor] .
   op s_ : Nat -> Nat [ctor] .
   op _<_ : Nat Nat -> Bool .
   vars N M : Nat .
   eq [NATURAL-1] : N < 0 = false .
   eq [NATURAL-2] : 0 < s(M) = true .
   eq [NATURAL-3] : s N < s M = N < M .
 endfm)

(mod BAKERY is
   protecting NATURAL .
   sorts Mode BState .
   ops sleep wait crit : -> Mode [ctor] .
   op <_‘,_‘,_‘,_> : Mode Nat Mode Nat -> BState [ctor] .

   op initial : -> BState .
   eq initial = < sleep, 0, sleep, 0 > .

   vars P Q : Mode .
   vars X Y : Nat .

   rl [p1_sleep] : < sleep, X, Q, Y > => < wait, s Y, Q, Y > .
   rl [p1_wait] : < wait, X, Q, 0 > => < crit, X, Q, 0 > .
   crl [p1_wait] : < wait, X, Q, Y > => < crit, X, Q, Y >
     if (Y < X) = false .
   rl [p1_crit] : < crit, X, Q, Y > => < sleep, 0, Q, Y > .

   rl [p2_sleep] : < P, X, sleep, Y > => < P, X, wait, s X > .
   rl [p2_wait] : < P, 0, wait, Y > => < P, 0, crit, Y > .
   crl [p2_wait] : < P, X, wait, Y > => < P, X, crit, Y >
     if Y < X  = true .
   rl [p2_crit] : < P, X, crit, Y > => < P, X, sleep, 0 > .
 endm)
```

In this module, states are represented by terms of sort `BState`, which are constructed by a 4-tuple operator `<_,_,_,_>`; the first two components describe the status of the first process (the mode it is currently in, and its priority as given by the number according to which it will be served), and the last two components the status of the second process. The rules describe how each process passes from being sleeping to waiting, from waiting to its critical section, and then back to sleeping.

We may wish to verify two basic properties about this protocol, namely:

- *mutual exclusion*, that is, the two processes are never simultaneously in their critical section, and
- *liveness*, that is, whenever a process enters the waiting mode, it will eventually enter its critical section.

Since the set of states reachable from `initial` is *infinite*, we should model check these properties using an abstraction. We can define the abstraction by adding to the equations of `BAKERY` a set $G$ of additional equations defining a quotient of the set of states. We can do so in the following module extending `BAKERY` by equations and leaving the rules unchanged:

```
(mod ABSTRACT-BAKERY is
   including BAKERY  .
   vars P Q : Mode .
   vars X Y : Nat .
   eq [AB-1] : < P, 0, Q, s s Y > = < P, 0, Q, s 0 > .
   eq [AB-2] : < P, s s X, Q, 0 > = < P, s 0, Q, 0 > .
   eq [AB-3] : < P, s s s X, Q, s s s Y > = < P, s s X, Q, s s Y > .
   eq [AB-4] : < P, s s s X, Q, s s 0 > = < P, s s 0, Q, s 0 > .
   eq [AB-5] : < P, s s s X, Q, s 0 > = < P, s s 0, Q, s 0 > .
   eq [AB-6] : < P, s s 0, Q, s s Y > = < P, s 0, Q, s 0 > .
   eq [AB-7] : < P, s 0, Q, s s Y > = < P, s 0, Q, s 0 > .
 endm)
```

Three key questions are:

- Is the set of states now finite?
- Does this abstraction correspond to a rewrite theory whose equations are ground confluent, sort-decreasing, and terminating?
- Are the rules still ground coherent?

We discuss here the ground coherence property. For proofs of the other requirements see [9]. Checking with Maude's Coherence Checker (ChC) gives us:

```
Maude> (check ground coherence ABSTRACT-BAKERY .)

Coherence checking of ABSTRACT-BAKERY
Coherence checking solution:
The following critical pairs cannot be rewritten:
  cp for AB-1 and p1_sleep
    < sleep,0,Q:Mode,s 0 >
    => < wait,s s s #3:Nat,Q:Mode,s s #3:Nat > .
  cp for AB-2 and p2_sleep
    < P:Mode,s 0,sleep,0 >
    => < P:Mode,s s #2:Nat,wait,s s s #2:Nat > .
  cp for AB-4 and p2_sleep
    < P:Mode,s s 0,sleep,s 0 >
    => < P:Mode,s s #2:Nat,wait,s s s #2:Nat > .
  cp for AB-6 and p1_sleep
    < sleep,s 0,Q:Mode,s 0 >
    => < wait,s s s #3:Nat,Q:Mode,s s #3:Nat > .
  ccp for AB-3 and p1_wait
```

```
    < wait,s s #2:Nat,Q:Mode,s s #4:Nat >
    => < crit,s s #2:Nat,Q:Mode,s s #4:Nat >
    if s s s #4:Nat < s s s #2:Nat = false .
  ccp for AB-3 and p2_wait
    < P:Mode,s s #2:Nat,wait,s s #4:Nat >
    => < P:Mode,s s #2:Nat,crit,s s #4:Nat >
    if s s s #4:Nat < s s s #2:Nat = true .
```

All of these critical pairs can be inductively rewritten.

The unconditional critical pairs have a similar pattern, they must be rewritten to a term of either the form `< P:Mode,s s V:Nat,wait,s s s V:Nat >` or `< P:Mode,s s s V:Nat,Q:Mode,s s V:Nat >`. We want to extend our specification with the equations

```
eq < Q:Mode, s s s Y:Nat, Q':Mode, s s Y:Nat >
  = < Q:Mode, s s 0, Q':Mode, s 0 > .
eq < Q:Mode, s s Y:Nat, Q':Mode, s s s Y:Nat >
  = < Q:Mode, s 0, Q':Mode, s s 0 > .
```

Before introducing them we prove that they are inductive consequences of our specification.

Let us begin with the first one. We can easily prove it using the ITP or directly in Maude. Directly in Maude, we have the following two goals

```
eq < Q:Mode, s s s 0, Q':Mode, s s 0 >
  = < Q:Mode, s s 0, Q':Mode, s 0 > .
eq < Q:Mode, s s s s Y:Nat, Q':Mode, s s s Y:Nat >
  = < Q:Mode, s s 0, Q':Mode, s 0 >
  if < Q:Mode, s s s Y:Nat, Q':Mode, s s Y:Nat >
     = < Q:Mode, s s 0, Q':Mode, s 0 > .
```

These two goals can be proved by simplifying the first goal to a syntactic identity.

```
Maude> reduce in ABSTRACT-BAKERY : < Q, s s s 0, Q':Mode, s s 0 > .
result BState: < Q, s s 0, Q':Mode, s 0 >
Maude> reduce in ABSTRACT-BAKERY : < Q, s s 0,Q':Mode, s 0 > .
result BState: < Q, s s 0, Q':Mode, s 0 >
```

And by applying the Theorem of Constants to the second goal and adding the premise (instantiated with a constant) as an extra lemma to simplify the conclusion (also instantiated with a constant) to a syntactic identity.

```
mod ABSTRACT-BAKERY-2 is
  including ABSTRACT-BAKERY .

  op a : -> Nat .

  eq < Q:Mode, s s s a, Q':Mode, s s a >
    = < Q:Mode, s s 0, Q':Mode, s 0 > .
endm

Maude> reduce in ABSTRACT-BAKERY-2 : < Q:Mode, s s s s a, Q':Mode, s s s a > .
result BState: < Q:Mode, s s 0, Q':Mode, s 0 >
Maude> reduce in ABSTRACT-BAKERY-2 : < Q:Mode, s s 0, Q':Mode, s 0 > .
result BState: < Q:Mode, s s 0, Q':Mode, s 0 >
```

We now want to prove the symmetric equation in a similar way.

```
eq < Q:Mode, s s Y:Nat, Q':Mode, s s s Y:Nat >
  = < Q:Mode, s 0, Q':Mode, s s 0 > .
```

We have the following two goals

```
eq < Q:Mode, s s 0, Q':Mode, s s s 0 >
  = < Q:Mode, s 0, Q':Mode, s s 0 > .
eq < Q:Mode, s s s Y:Nat, Q':Mode, s s s s Y:Nat >
  = < Q:Mode, s 0, Q':Mode, s s 0 >
  if < Q:Mode, s s Y:Nat, Q':Mode, s s s Y:Nat >
     = < Q:Mode, s 0, Q':Mode, s s 0 > .
```

The first goal is simplified to a syntactic identity.

```
Maude> reduce in ABSTRACT-BAKERY : < Q, s s 0, Q':Mode, s s s 0 > .
result BState: < Q, s 0, Q':Mode, s s 0 >
Maude> reduce in ABSTRACT-BAKERY : < Q, s 0,Q':Mode, s s 0 > .
result BState: < Q, s 0, Q':Mode, s s 0 >
```

And by proceeding similarly for the second goal, we also simplify the conclusion to a syntactic identity.

```
mod ABSTRACT-BAKERY-3 is
  including ABSTRACT-BAKERY .

  op a : -> Nat .

  eq < Q:Mode, s s a, Q':Mode, s s s a >
    = < Q:Mode, s 0, Q':Mode, s s 0 > .
endm
```

```
Maude> reduce in ABSTRACT-BAKERY-3 : < Q:Mode, s s s a, Q':Mode, s s s s a > .
result BState: < Q:Mode, s 0, Q':Mode, s 0 >
Maude> reduce in ABSTRACT-BAKERY-3 : < Q:Mode, s 0, Q':Mode, s s 0 > .
result BState: < Q:Mode, s 0, Q':Mode, s 0 >
```

We can now check the ground coherence of the module resulting from including these new two equations.

```
(mod ABSTRACT-BAKERY-4 is
   including ABSTRACT-BAKERY .

   eq [AB-4-1] : < Q:Mode, s s s Y:Nat, Q':Mode, s s Y:Nat >
     = < Q:Mode, s s 0, Q':Mode, s 0 > .
   eq [AB-4-2] : < Q:Mode, s s Y:Nat, Q':Mode, s s s Y:Nat >
     = < Q:Mode, s 0, Q':Mode, s s 0 > .
endm)
```

```
Maude> (check Church-Rosser ABSTRACT-BAKERY-4 .)

Church-Rosser checking of ABSTRACT-BAKERY-4
Checking solution:
All critical pairs have been joined.
The specification is locally-confluent.
The specification is sort-decreasing.


Maude> (check ground coherence ABSTRACT-BAKERY-4 .)

Coherence checking of ABSTRACT-BAKERY-4
Coherence checking solution:
The following critical pairs cannot be rewritten:
  ccp for AB-3 and p1_wait
    < wait,s s #2:Nat,Q:Mode,s s #4:Nat >
    => < crit,s s #2:Nat,Q:Mode,s s #4:Nat >
    if s s s #4:Nat < s s s #2:Nat = false .
  ccp for AB-3 and p2_wait
    < P:Mode,s s #2:Nat,wait,s s #4:Nat >
    => < P:Mode,s s #2:Nat,crit,s s #4:Nat >
    if s s s #4:Nat < s s s #2:Nat = true .
  ccp for AB-4-1 and p1_wait
    < wait,s s 0,Q:Mode,s 0 >
    => < crit,s s 0,Q:Mode,s 0 >
    if s s #2:Nat < s s s #2:Nat = false .
  ccp for AB-4-2 and p2_wait
    < P:Mode,s 0,wait,s 0 >
    => < P:Mode,s 0,crit,s 0 >
    if s s s #2:Nat < s s #2:Nat = true .
```

The conditions of the last two conditional critical pairs are unfeasible. It can easily be proven by induction, or let the tool do it by introducing equations

```
eq X:Nat < s X:Nat = true .
eq s X:Nat < X:Nat = false .
```

The first two can be proven rewritable. Let us go for the first one.

```
crl [p1_wait] : < wait, X, Q, Y > => < crit, X, Q, Y >
  if (Y < X) = ff .
```

We can simplify its condition as follows:

```
ccp for AB-3 and p1_wait
  < wait,s s #2:Nat,Q:Mode,s s #4:Nat >
  => < crit,s s #2:Nat,Q:Mode,s s #4:Nat >
  if #4:Nat < #2:Nat = ff .
```

Using the Theorem of Constants, we can convert the variables `#2:Nat` and `#4:Nat` into constants `a` and `b` and add an equation assuming the condition `b < a = ff`.

```
mod ABSTRACT-BAKERY-5 is
```

```
    including ABSTRACT-BAKERY .
    ops a b : -> Nat .
    eq b < a = ff .
  endm
```

Then, we can fill in this conditional critical pair by giving the search command:

```
search in ABSTRACT-BAKERY-5 : < wait,s s a,Q:Mode,s s b > =>1 X:BState .

Solution 1 (state 1)
states: 2  rewrites: 4 in 0ms cpu (0ms real) (2000000 rewrites/second)
X:BState --> < crit,s s a,Q:Mode,s s b >

No more solutions.
```

## References


[1] *Proceedings of the CafeOBJ Symposium'98*, April 1998.

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

[3] Manuel Clavel. *Reflection in Rewriting Logic: Metalogical Foundations and Metaprogramming Applications*. CSLI Lecture Notes. CSLI Publications, 2000.

[4] Manuel Clavel, Francisco Durán, Steven Eker, Santiago Escobar, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott. Unification and narrowing in Maude 2.4. In Ralf Treinen, editor, *Rewriting Techniques and Applications, 20th International Conference, RTA 2009, Brasília, Brazil, June 29 - July 1, 2009, Proceedings*, volume 5595 of *Lecture Notes in Computer Science*, pages 380–390. Springer, 2009.

[5] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, and José Meseguer. Metalevel computation in Maude. In Claude Kirchner and Hélène Kirchner, editors, *Proceedings Second International Workshop on Rewriting Logic and its Applications, WRLA'98, Pont-à-Mousson, France, September 1–4, 1998*, volume 15 of *Electronic Notes in Theoretical Computer Science*, pages 3–24. Elsevier, 1998. `http://www.elsevier.nl/locate/entcs/volume15.html`.

[6] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José Quesada. Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, 2002.

[7] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José F. Quesada. The Maude system. In P. Narendran and M. Rusinowitch, editors, *Rewriting Techniques and Applications, 10th International Conference, RTA'99, Trento, Italy, July 2–4, 1999, Proceedings*, volume 1631 of *Lecture Notes in Computer Science*, pages 240–243. Springer, 1999.

[8] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. The Maude 2.0 system. In Robert Nieuwenhuis, editor, *Proceedings of the 14th International Conference on Rewriting Techniques and Applications (RTA 2003)*, volume 2706 of *Lecture Notes in Computer Science*, pages 76–87, Berlin, June 2003. Springer.

[9] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *All About Maude - A High-Performance Logical Framework: How to Specify, Program, and Verify Systems in Rewriting Logic*. Lecture Notes in Computer Science. Springer, 2007.

[10] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. Maude 2.4 manual. Available in `http://maude.cs.uiuc.edu`, November 2008.

[11] Manuel Clavel, Francisco Durán, Steven Eker, and José Meseguer. Building equational logic tools by reflection in rewriting logic. In *Proceedings of the CafeOBJ Symposium*. CafeOBJ Project, April 1998.

[12] Manuel Clavel, Francisco Durán, Steven Eker, and José Meseguer. Design and implementation of the Cafe prover and the Church-Rosser checker tools. Technical report, Computer Science Laboratory, SRI International, March 1998. `http://maude.csl.sri.com/papers`.

[13] Manuel Clavel, Francisco Durán, Steven Eker, and José Meseguer. Building equational proving tools by reflection in rewriting logic. In Kokichi Futatsugi, Ataru T. Nakagawa, and Tetsuo Tamai, editors, *Cafe: An Industrial-Strength Algebraic Formal Method*, pages 1–31. Elsevier, 2000. `http://maude.csl.sri.com/papers`.

[14] Manuel Clavel, Francisco Durán, Steven Eker, José Meseguer, and Mark-Oliver Stehr. Maude as a formal meta-tool. In Kokichi Futatsugi, Joseph Goguen, and José Meseguer, editors, *Proceedings of the OBJ/CafeOBJ/Maude Wokshop*, pages 1–16. Theta, 1999.

[15] Manuel Clavel, Francisco Durán, Joe Hendrix, Salvador Lucas, José Meseguer, and Peter Ölveczky. The Maude formal tool environment. In Till Mossakowski, Ugo Montanari, and Magne Haveraaen, editors, *Algebra and Coalgebra in Computer Science, Second International Conference, CALCO 2007, Bergen, Norway, August 20-24, 2007, Proceedings*, volume 4624 of *Lecture Notes in Computer Science*, pages 173–178. Springer, 2007.

[16] Manuel Clavel and José Meseguer. Reflection in conditional rewriting logic. *Theoretical Computer Science*, 285(2):245 – 288, 2002.

[17] Manuel Clavel, Miguel Palomino, and Adrián Riesco. Introducing the ITP tool: a tutorial. *Journal of Universal Computer Science*, 12(11):1618–1650, 2006.

[18] Răzvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report. The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*, volume 6 of *AMAST Series in Computing*. World Scientific, 1998.

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

[20] Francisco Durán. Coherence checker and completion tools for Maude specifications. Technical Report ITI-2000-7, Dpto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga, October 2000. Available at `http://maude.cs.uiuc.edu`.

[21] Francisco Durán. The extensibility of Maude's module algebra. In Teodor Rus, editor, *Algebraic Methodology and Software Technology, 8th International Conference, AMAST 2000, Iowa City, Iowa, USA, May 20–27, 2000, Proceedings*, volume 1816 of *Lecture Notes in Computer Science*, pages 422–437. Springer, 2000.

[22] Francisco Durán. Termination checker and Knuth-Bendix completion tools for Maude equational specifications. Technical Report ITI-2000-6, Dpto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga, October 2000. Available at `http://maude.cs.uiuc.edu`.

[23] Francisco Durán, Salvador Lucas, and José Meseguer. MTT: The Maude termination tool (system description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning 4th International Joint Conference, IJCAR 2008 Sydney, Australia, August 12-15, 2008 Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 313–319. Springer, 2008.

[24] Francisco Durán, Salvador Lucas, and José Meseguer. Termination modulo combinations of equational theories. In Silvio Ghilardi and Roberto Sebastiani, editors, *Frontiers of Combining Systems, 7th International Symposium, FroCoS 2009, Trento, Italy, September 16-18, 2009. Proceedings*, volume 5749 of *Lecture Notes in Computer Science*, pages 246–262. Springer, 2009.

[25] Francisco Durán and José Meseguer. The Maude specification of Full Maude. Manuscript, SRI International. Available at `http://maude.cs.uiuc.edu`, February 1999.

[26] Francisco Durán and José Meseguer. A Church-Rosser checker tool for Maude equational specifications. Technical Report ITI-2000-5, Dpto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga, October 2000. Available at `http://maude.cs.uiuc.edu`.

[27] Francisco Durán and José Meseguer. Crc 3: A church-rosser checker tool for conditional order-sorted equational maude specifications. Available at `http://maude.lcc.uma.es/CRChC`, 2009.

[28] Francisco Durán and Peter Csaba Ölveczky. A guide to extending Full Maude illustrated with the implementation of Real-Time Maude. In Grigore Roşu, editor, *Proceedings 7th International Workshop on Rewriting Logic and its Applications (WRLA'08)*, Electronic Notes in Theoretical Computer Science. Elsevier, 2008.

[29] Kokichi Futatsugi and Toshimi Sawada. Cafe as an extensible specification environment. In *Proceedings of the Kunming International CASE Symposium, Kunming, China*, November 1994.

[30] Jürgen Giesl and Deepak Kapur. Dependency pairs for equational rewriting. In *Proceedings of the 12th International Conference on Rewriting Techniques and Applications (RTA'01)*, volume 2051 of *Lecture Notes in Computer Science*, pages 93–108. Springer, 2001.

[31] Joe Hendrix, José Meseguer, and Hitoshi Ohsaki. A sufficient completeness checker for linear order-sorted specifications modulo axioms. In Ulrich Furbach and Natarajan Shankar, editors, *A Sufficient Completeness Checker for Linear Order-Sorted Specifications Modulo Axioms*, volume 4130 of *Lecture Notes in Computer Science*, pages 151–155. Springer, 2006.

[32] Jean-Pierre Jouannaud and Hélène Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15(4):1155–1194, 1986.

[33] Narciso Martí-Oliet and José Meseguer. General logics and logical frameworks. In D. M. Gabbay, editor, *What is a Logical System?*, volume 4 of *Studies in Logic and Computation*, pages 355–392. Oxford University Press, 1994.

[34] Narciso Martí-Oliet and José Meseguer. Rewriting logic as a logical and semantic framework. In D.M. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 9, pages 1–87. Kluwer Academic Publishers, second edition, 2002.

[35] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[36] José 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.

[37] Peter Csaba Ölveczky and José Meseguer. Semantics and pragmatics of Real-Time Maude. *Higher-Order and Symbolic Computation*, 20(1-2):161–196, 2007.

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

[39] Patrick Viry. Rewriting modulo a rewrite system. Technical Report TR-95-20, Dipartimento di Informatica, Università di Pisa, December 1995. `ftp://ftp.di.unipi.it/pub/techreports/TR-95-20.ps.Z`.

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