ELSEVIER

Theoretical Computer Science 285 (2002) 487–517

Theoretical Computer Science

www.elsevier.com/locate/tcs

# Equational rules for rewriting logic

Patrick Viry*

*ASTEM RI, 17 Chudoji Minami-machi, Shimogyo-ku, Kyoto 600-8813, Japan*

**Abstract**

In addition to equations and rules, we introduce equational rules that are oriented while having an equational interpretation. Correspondence between operational behavior and intended semantics is guaranteed by a property of coherence, which can be checked by examination of critical pairs and linearity conditions. We present applications of this theory to three examples where the rewrite relation is interpreted, respectively, as equality, transition and deduction. © 2002 Elsevier Science B.V. All rights reserved.

*Keywords:* Rewriting; Equational programming; Coherence; Building-in equality

## 1. Introduction

Rewriting logic [31] can be thought of as one of the most general logical frameworks, in the sense that all logical systems found in practice can be mapped in a simple way into a rewriting logic theory [29]. A major reason of this success is that rewriting logic does not impose any implicit structure on the objects to be represented: all structure must be explicitly defined using equations.

A rewrite theory mainly consists of a set of oriented rules $R$ and a set of non-oriented equations $E$. Operationally, the sequent $s \rightarrow t$ is valid in the rewrite theory $(E,R)$ iff there is a derivation from $s$ to $t$ by rewriting with the rules of $R$ modulo $E$. Equations in $E$ can be thought of as specifying the structure over which rules of $R$ operate. The semantics of rules in $R$ is not fixed; depending on the application, a rewrite step can be interpreted for instance as a transition between states (e.g. automata, processes), as a logical deduction (e.g. sequent calculus) or as equality (e.g. $\beta$-reduction in the $\lambda$-calculus).

* Corresponding author. Present address: ILOG S.A., B.P. 85, grue de Verdun, 94253 Gentilly Cedex, France.

*E-mail address:* pviry@ilog.fr (P. Viry).

0304-3975/02/$ - see front matter 
PII: S0304-3975(01)00366-8

Let us take a closer look at this dichotomy between rules and equations:

- from a semantic point of view, the equations in $E$ are always given an equational semantics, but it is not necessarily so for the rules in $R$.
- from an operational point of view, the rules in $R$ can be used only according to their orientation, while the equations in $E$ are not oriented.

These two aspects are often conflicting, and choosing between specifying something as a rule or an equation can be a dilemma. In particular, when $E$ is not tractable directly, classical rewriting techniques can replace all or part of $E$ by oriented rewrite rules, mapping equivalence classes to their unique canonical representative. But doing so implies adding these rules to $R$, where they lose their equational semantics.

A solution to this dilemma is to introduce a second set of rules $ER$ (for "Equational Rules", denoted with $\overset{=}{\rightarrow}$), which are oriented but keep an equational semantics. Consider the following example, which may be thought of as specifying a double function ($f$) and a non-deterministic choice operator (?) over the natural numbers:

| $E$ | $R$ |
|---|---|
| (1) $x+y=y+x$ | (4) $f(x)\rightarrow x+x$ |
| (2) $0+x=x$ | (5) $?\rightarrow 0$ |
| (3) $s(x)+y=s(x+y)$ | (6) $?\rightarrow 1$ |

$R$ is not interpreted equationally, hence the rules for non-deterministic choice do not imply $0=1$. In order to handle $E$ more efficiently, one may wish to orient Eqs. (2) and (3) into equational rules, changing their operational meaning (they ought to be used only according to their orientation), while preserving their equational semantics ($0+x$ is *equal* to $x$).

For semantic considerations, one may wish also to transform rule (4) into an equational rule, keeping its operational meaning while giving it an equational semantics.

The resulting *oriented rewrite theory* $(E,ER,R)$ is

| $E$ | $ER$ | $R$ |
|---|---|---|
| (1) $x+y=y+x$ | (2) $0+x\overset{=}{\rightarrow}x$<br>(3) $s(x)+y\overset{=}{\rightarrow}s(x+y)$<br>(4) $f(x)\overset{=}{\rightarrow}x+x$ | (5) $?\rightarrow 0$<br>(6) $?\rightarrow 1$ |

## 2. Structure of the paper

After introducing some basic concepts (Sections 3 and 4), we formally define the notion of an oriented rewrite theory in Section 5.

The central notion appearing in this framework is the property of *coherence* (Section 5.1), which ensures a correspondence between operational behavior and intended semantics. We show how to check coherence (Sections 5.2 and 5.3), and possibly obtain it through coherence completion (Section 5.5). Section 6 extends the previous results to the case of conditional rules.

In Section 7, we show how the techniques of [23] for implementing rewriting modulo $E$ using matching modulo $E$ can be extended to the case of an oriented rewrite theory.

In Section 8, we develop an example showing how the coherence techniques may help replacing a non-terminating rewrite relation by a terminating one.

The rest of the paper is devoted to examples. We present three examples, corresponding to the most usual interpretations of the rewriting relation:

(1) *Equational semantics* (Section 9): Example of calculi with bound variables, whose main representative is the $\lambda$-calculus. Substitution is usually considered an implicit meta-operation, but a full first-order specification requires to make it explicit using a substitution calculus. The substitution rules are oriented rules, with an equational semantics (i.e. part of the internal structure).

(2) *Transition semantics* (Section 10): A rewrite step is interpreted as a transition between states, the structure of a state being specified by $E$ and $ER$. A typical example is a process algebra, in the spirit of CCS [33]. Building on the treatment of bound variables developed above, we present the example of the $\pi$-calculus [34].

This example can be extended by specifying the actual data exchanged by processes as abstract data types, in a manner similar to LOTOS [5], thus having the transition semantics and data types specified in a single framework.

(3) *Deduction semantics* (Section 11): A rewrite step is interpreted as a logical deduction between propositions. By varying the structure of propositions, we can specify for instance classical logic or linear logic (including or not the weakening axioms in $E$). Here moving a rule from $R$ to $ER$ can be seen as the building-in of equality.

Most of the material presented here originates from [38,39,40,41,42]. Some changes and additions have been introduced, in particular a new definition of local coherence (Section 5.2) that allows to get rid of a troublesome termination condition, and a section on conditional rules.

The coherence checking and coherence completion techniques presented here have been implemented in Maude [30] using the reflection features of this system, and are now part of the standard distribution [14].

## 3. Basic concepts

### 3.1. Relations

Given a binary relation $\rightarrow$, we denote $\leftarrow$ its inverse, $\twoheadrightarrow$ its transitive and reflexive closure, $\leftrightarrow$ its symmetric closure and $\twoheadleftarrow\!\!\twoheadrightarrow$ its transitive symmetric and reflexive closure. The notation $\twoheadrightarrow$ is equivalent to the usual $\rightarrow^*$, but makes diagrams much more readable. An element $u$ is a $\rightarrow$-normal form if there is no $v$ such that $u \rightarrow v$. The normalizing relation $\overset{!}{\twoheadrightarrow}$ is defined as $u \overset{!}{\twoheadrightarrow} v$ if $u \twoheadrightarrow v$ and $v$ is a $\rightarrow$-normal form.

Given a rewrite system $R = \{l_i \rightarrow r_i\}$, the rewrite relation $\overset{R}{\rightarrow}$ is the smallest relation stable by context and instantiation containing $l_i \overset{R}{\rightarrow} r_i$ for each $l_i \rightarrow r_i \in R$. For a set of equations $E = \{u_i = v_i\}$, the replacement relation $\overset{E}{\leftrightarrow}$ is defined similarly, and closed

by symmetry. $[u]_E$ denotes the equivalence class of $u$ modulo $E$, i.e. $[u]_E$ is the set $\{v | v \overset{E}{\leftarrow\!\!\!\twoheadrightarrow} u\}$. Similarly, a set of rules $R$, when considered as equations, also defines equivalence classes $[u]_R$. For any relations $\overset{R}{\rightarrow}$ and $\overset{S}{\rightarrow}$, define

$$\overset{R/S}{\rightarrow} = \overset{S}{\twoheadrightarrow}\overset{R}{\rightarrow}\overset{S}{\twoheadrightarrow}$$

In particular, when $\overset{S}{\rightarrow}$ is the replacement relation $\overset{E}{\leftrightarrow}$ generated by a set $E$ of equations, $\overset{R/E}{\longrightarrow}$ is the basis for defining rewriting over classes modulo $E$:

$$[u]_E \overset{[R]_E}{\rightarrow} [v]_E \quad \text{iff} \quad u \overset{R/E}{\rightarrow} v.$$

We use the abbreviations $\overset{[R/S]_E}{\longrightarrow}$ for $\overset{[R]_E/[S]_E}{\longrightarrow} = \overset{[S]_E}{\twoheadrightarrow}\overset{[R]_E}{\longrightarrow}\overset{[S]_E}{\twoheadrightarrow}$, and $\overset{[R\cup S]_E}{\longrightarrow}$ for $\overset{[R]_E}{\longrightarrow} \cup \overset{[S]_E}{\longrightarrow}$.

Finally, because it makes definitions and diagrams simpler, instead of $\overset{[R/S]_E}{\longrightarrow}$ we will often use the relation $\overset{[R(S)]_E}{\longrightarrow}$ defined as

$$\overset{[R(S)]_E}{\rightarrow} = \overset{[S]_E}{\twoheadrightarrow}\overset{[R]_E}{\rightarrow}$$

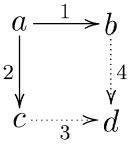
$\overset{[S]_E}{\twoheadrightarrow}$ steps are allowed only before the $\overset{[R]_E}{\longrightarrow}$ steps, but we have the following correspondence:

$$\overset{[R/S]_E}{\rightarrow} = \overset{[R(S)]_E}{\rightarrow}\overset{[S]_E}{\twoheadrightarrow}$$

and for derivations $\overset{[R/S]_E}{\twoheadrightarrow} = \overset{[R(S)]_E}{\twoheadrightarrow}\overset{[S]_E}{\twoheadrightarrow}$

### 3.2. Diagrams

A diagram is a way to express complex formulas about relations. Plain arrows are quantified universally and dotted arrows existentially. In a complex diagram, the numbers inside the smaller diagrams state in what order we prove them. For instance:




reads: for all $a$, $b$ and $c$ such that $a \overset{1}{\rightarrow} b$ and $a \overset{2}{\rightarrow} c$, there exists $d$ such that $c \overset{3}{\rightarrow} d$ and $b \overset{4}{\rightarrow} d$.

## 4. The permutation lemma

The following permutation lemma will be used in Section 5.3 (checking local coherence). It is generic in the sense that it is valid for $E$ containing axioms for A (associative) and AC (associative and commutative) symbols. We do not know of

any other case where this lemma is valid (it does not hold even for arbitrary shallow theories [8]). Intuitively, the lemma relies on being able to identify positions in a term. For A and AC symbols, positions in flattened terms are considered.

We do not consider this restriction to A and AC as a practical limitation, since available rewrite interpreters such as OBJ [17], Maude [32,30] or ELAN [39,24] do provide implementations only for rewriting modulo theories including the axioms of associativity, commutativity, identity and idempotency, and the last two axioms will normally be considered as oriented equational rules.

**Definition 1** (Flattened terms [11]). A flattened term is a term $[t]_{\mathsf{A}}$ considered modulo all associativity axioms in $E$ (in the following we consider only flattened terms, including in rules, and abbreviate $[t]_{\mathsf{A}}$ to $t$). It is possible to define a notion of positions $Pos(t)$ in flattened terms, depending on the properties of the topmost symbol $f$:

- If $f$ is a free symbol, then

$$Pos(f(t_1,\dots,t_n) = \{\lambda\} \cup \bigcup_{i=1}^{n} \{ip \mid p \in Pos(t_i)\}.$$

- If $f$ is an A symbol, then

$$Pos(f(t_1,\dots,t_n)) = \{\lambda\} \cup \{[i,j] \mid 1 \leqslant i < j \leqslant n\} \cup \bigcup_{i=1}^{n} \{ip \mid p \in Pos(t_i)\}.$$

- If $f$ is an AC symbol, then

$$Pos(f(t_1,\dots,t_n)) = \{\lambda\} \cup \mathscr{P}(\{1,\dots,n\}) \cup \bigcup_{i=1}^{n} \{ip \mid p \in Pos(t_i)\}.$$

The usual notions of substitution, stability by context and instantiation, rewriting, are easily extended to flattened terms.

For instance, considering an AC symbol $+$, the rule $a + c \to d$ applies to the term $a + b + c$ at position $\{1,3\}$, giving $b + d$ as a reduct.

**Definition 2** (Extended critical pairs [27]). Extended critical pairs extend the usual notion of critical pairs to flattened terms. Consider two flattened rules $\rho_1 : l_1 \to r_1$ and $\rho_2 : l_2 \to r_2$; there is a head superposition between $l_1$ and $l_2$ if $Head(l_1) = Head(l_2) = f$ and there exists a substitution $\sigma$ such that one of the three possibilities below holds:

- $f$ is a free symbol and $l_1\sigma =_E l_2\sigma$
- $f$ is an A symbol and $L_1 =_E L_2$, where $L_1$ is either $l_1\sigma$, $f(x,l_1)\sigma$, $f(x,l_1,y)\sigma$ or $f(l_1,y)\sigma$, and $L_2$ is either $l_2\sigma$, $f(u,l_2)\sigma$, $f(u,l_2,v)\sigma$ or $f(l_2,v)\sigma$, with $x,y,u,v$ being new variables.
- $f$ is an AC symbol and $L_1 =_E L_2$, where $L_1$ is either $l_1\sigma$ or $f(x,l_1)\sigma$, $L_2$ is either $l_2\sigma$ or $f(u,l_2)\sigma$, with $x$, $u$ being new variables.

Superpositions of $l_2$ within $l_1$ at position $p \neq \lambda$ are defined similarly, considering for $L_1$ only the case $L_1 = l_1\sigma$.

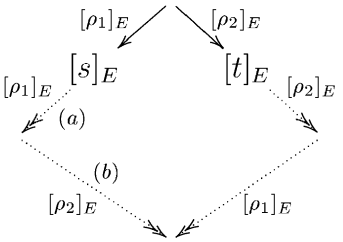
The set $CP_E(\rho_1,\rho_2)$ of extended critical pairs is defined as usual considering all superpositions between $l_1$ and $l_2$. Note that in the presence of A symbols, the set of critical pairs may possibly be infinite since A-unification is not finitary [37].

For instance, considering an AC symbol $+$, the rules $a+b\to c$ and $a+d\to e$ induce the critical pair $(c+d,b+e)$. Taking $\sigma=\{x\mapsto d, y\mapsto b\}$, we have $(a+b+x)\sigma=_{\mathsf{AC}}(a+d+y)\sigma=_{\mathsf{AC}}a+b+d$.

This definition of critical pairs is different from the usual notion of critical pairs modulo [23]: it takes into account the fact that rewriting is performed directly on flattened terms and in some sense internalizes the extension rules [35] that would be needed if rewriting modulo were done on syntactic (non-flattened) terms.

**Lemma 3** (Permutation lemma). *For an equational theory $E$ containing only axioms for* A *and* AC *symbols, we have: for any two rules $\rho_1: l_1\to r_1$ and $\rho_2: l_2\to r_2$, and any critical peak $[s]_E \xleftarrow{[\rho_1]_E} \xrightarrow{[\rho_2]_E}[t]_E$,*

- *(Superposition case) either there is a critical pair $(s,t)\in CP_E(\rho_1,\rho_2)$,*
- *(Non-superposition case) or the critical peak can be closed as follows*




*Moreover, if $l_2$ is linear or if the $l_1\to r_1$ step is not* below *the $l_2\to r_2$ step, then the derivation $(a)$ is empty. If $r_2$ is linear or if the $l_1\to r_1$ step is not* below *the $l_2\to r_2$ step, then the derivation $(b)$ is a single step. Note that the notion of applying a step below another has to be understood in the context of flattened terms.*

A proof for the case AC is given in [28]. With the definition of superposition for A symbols and the associated definition of extended critical pairs that we introduced here, the same proof works by replacing $=_{\mathsf{AC}}$ with $=_{\mathsf{A}}$ when considering A symbols. Note that the notion of critical pairs modulo $E$ proposed in [23] is not general enough for the permutation lemma to hold.

## 5. Oriented rewrite theory

Extending the notion of a rewrite theory of rewriting logic [31], we define an *oriented rewrite theory* as a tuple $(\Sigma,E,ER,R)$, where

- $\Sigma$ is the signature upon which terms are built
- $E$ is a set of *equations* $(u=v)\in T^2_{\Sigma,X}$

- $ER$ is a set of *equational rules* $(u \xrightarrow{=} v) \in T^2_{\Sigma,X}$
- $R$ is a set of *reduction rules*[1] $(u \to v) \in T^2_{\Sigma,X}$.

The case of conditional rules will be addressed later in Section 6. Compared to a "classical" rewrite theory, this definition makes room for two kinds of rules:

- Rules of $ER$ are always given an equational interpretation: the oriented rewrite theory defines equivalence classes modulo $E \cup ER$. We will always suppose that the rules of $ER$ form a rewrite system convergent modulo $E$ (i.e. $\xrightarrow{[ER]_E}$ is confluent and terminating).
- Rules of $R$ are not necessarily interpreted equationally. They may be interpreted for instance as transitions between states or deduction steps. However, nothing prevents giving an equational interpretation to the reduction rules as well.

In "classical" rewriting logic, the same *semantics* would be obtained by considering $ER$ rules as non-oriented equations. The idea behind the above definition is that even though rules of $ER$ have an equational interpretation, they must be used only according to their orientation. Each of these considerations leads to a different view:

*Operational view*: "The rules of $ER$ must be used according to their orientation". This leads to consider sequents for the rewrite theory $(E, ER \cup R)$:

$$[s]_E \xrightarrow{[R(ER)]_E} [t]_E$$

*Semantic view*: "The rules of $ER$ have an equational semantics". This leads to consider sequents for the rewrite theory $(E \cup ER, R)$:

$$[s]_{E\cup ER} \xrightarrow{[R]_{E\cup ER}} [t]_{E\cup ER}$$

These two relations are not equivalent. We have

$$[s]_E \xrightarrow{[R(ER)]_E} [t]_E \;\Rightarrow\; [s]_{E\cup ER} \xrightarrow{[R]_{E\cup ER}} [t]_{E\cup ER}$$

but the opposite is not true in general. As an example, consider

$$E = \emptyset, \qquad ER = \{x + 0 \xrightarrow{=} x\}, \qquad R = \{f(x + y) \to g(x) + g(y)\}.$$

We have $[f(x)]_{E\cup ER} \xrightarrow{[R]_{E\cup ER}} [g(x) + g(0)]_{E\cup ER}$, since $[f(x)]_{E\cup ER} = [f(x+0)]_{E\cup ER}$, but $[f(x)]_E$ is not reducible by $\xrightarrow{[R(ER)]_E}$.

In the next section, we introduce different notions of *coherence*. Strong coherence is a sufficient condition for the above implication to be an equivalence, but may be difficult to verify. Weaker notions of coherence correspond to weaker correspondences between $\xrightarrow{[R(ER)]_E}$ and $\xrightarrow{[R]_{E\cup ER}}$.

Then in the following sections, we provide some techniques for checking or establishing these coherence properties.

[1] In the original definition, rules may also be labelled.

### 5.1. Coherence

Depending on how close a correspondence we seek between the operational and semantic views of oriented rewriting logic, we introduce three notions of coherence.

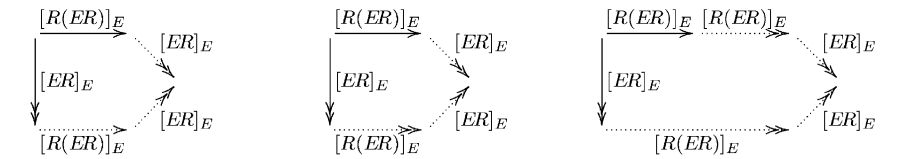
- Strong coherence states that the exact number of steps of a derivation is preserved (one may wish also to impose that the same rule is used — or a rule generated by completion from the original one, see Section 5.5 — the results extend straightforwardly to this case).

  It is the property needed when one wishes to preserve the whole derivation space, namely being able to identify single steps of the semantic view and of the operational view, for instance when implementing automata or process algebras.
- Weak coherence states that derivability is preserved, but not necessarily the intermediate steps.
- Equational coherence states that the set of normal forms is preserved, except possibly for some cycles.

  This property makes sense mainly in an equational context ($R$ equationally interpreted), where one is only interested in normal forms with respect to $\xrightarrow{[R]_{E\cup ER}}$. In such a context, $\xrightarrow{[R]_{E\cup ER}}$ is usually supposed to be confluent.

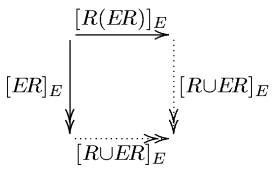
Equational coherence is the most general property, it can be checked by looking at critical pairs. Weak coherence may be more difficult to establish in the presence of non-left-linear rules in $R$, and strong coherence in the presence of non-right-linear rules as well. We will see sufficient conditions in the next sections.

**Definition 4** (Coherence). $\xrightarrow{[R(ER)]_E}$ is strongly (1), weakly (2) or equationally (3) coherent if




*(1) strong coherence* *(2) weak coherence* *(3) equational coherence*

Properties (1) and (2) differ only in the number of $\xrightarrow{[R(ER)]_E}$ steps allowed to close the diagram. Note that property (3) is equivalent to property (3′) (below).




*(3′) equational coherence*

The intuition behind the definition of these three notions of coherence is justified by the following theorem.

**Theorem 5** (Coherence). *Remember that we suppose* $\xrightarrow{[ER]_E}$ *convergent.*

- *If* $\xrightarrow{[R(ER)]_E}$ *is strongly coherent, then*

$$[u]_{E\cup ER} \xrightarrow{[R]_{E\cup ER}} [v]_{E\cup ER} \Rightarrow [u]_E \xrightarrow{[R(ER)]_E} [v']_E \xleftarrow{[ER]_E} [v]_E.$$

- *If* $\xrightarrow{[R(ER)]_E}$ *is weakly coherent, then*

$$[u]_{E\cup ER} \xrightarrow{[R]_{E\cup ER}} [v]_{E\cup ER} \Rightarrow [u]_E \xrightarrow{[R(ER)]_E} [v']_E \xleftarrow{[ER]_E} [v]_E.$$

- *If* $\xrightarrow{[R(ER)]_E}$ *is equationally coherent, then*
  *if* $[u]_E$ *is a normal form for* $\xrightarrow{[R(ER)]_E}$*, then either* $[u]_{E\cup ER}$ *is a normal form for* $\xrightarrow{[R]_{E\cup ER}}$*, or there is a non-empty cycle* $[u]_{E\cup ER} \xrightarrow{[R]_{E\cup ER}} [u]_{E\cup ER}$.

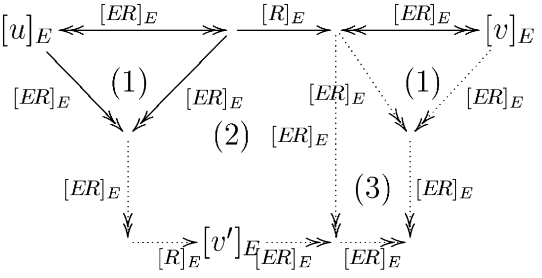
In the case of equational coherence, the set of normal forms is not exactly preserved, since a cycle in $\xrightarrow{[R]_{E\cup ER}}$ may be matched by an empty derivation of $\xrightarrow{[R(ER)]_E}$. This is the reason why in the identity rewriting example (Section 8), a non-terminating relation modulo AC1 can be simulated by a terminating relation, and the reason why we choose this definition of equational coherence.

If an exact correspondence between the sets of normal forms is required, the definition of equational coherence has to be adapted in order to close the diagram with at least one $\xrightarrow{[R(ER)]_E}$ step. This latter definition rules out the possibility of a non-empty cycle.

**Proof** (Strong case). Suppose that $[u]_{E\cup ER} \xrightarrow{[R]_{E\cup ER}} [v]_{E\cup ER}$, which is equivalent to

$$[u]_E \xleftarrow{[ER]_E} \xrightarrow{[R]_E} \xleftarrow{[ER]_E} [v]_E$$

We have

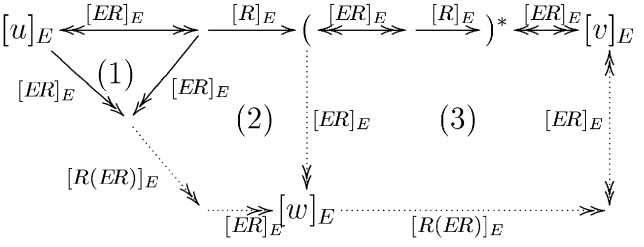


(1) confluence of $\xrightarrow{[ER]_E}$
(2) strong coherence of $\xrightarrow{[R(ER)]_E}$
(3) confluence of $\xrightarrow{[ER]_E}$.
Hence $[u]_E \xrightarrow{[R(ER)]_E} [v']_E \xleftarrow{[ER]_E} [v]_E$.

*Weak case*. Suppose that $[u]_{E\cup ER} \overset{[R]_{E\cup ER}}{\twoheadrightarrow} [v]_{E\cup ER}$, which is equivalent to

$$[u]_E \underbrace{(\xleftrightarrow{[ER]_E} \xrightarrow{[R]_E})^*}_{n} \xleftarrow{[ER]_E} [v]_E.$$

We want to prove that $[u]_E \overset{[R(ER)]_E}{\twoheadrightarrow} \overset{[ER]_E}{\twoheadleftarrow} [v]_E$ by induction over the number $n$ of $\xrightarrow{[R]_E}$ steps in the derivation. Either $n=0$ and the theorem is trivially true, or we have



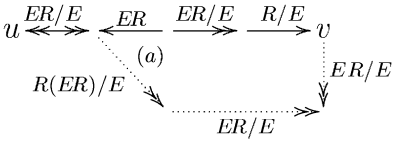

(1) confluence of $\xrightarrow{[ER]_E}$
(2) weak coherence of $\xrightarrow{[R(ER)]_E}$
(3) induction hypothesis
with $n-1$ $R$-steps
Hence $[u]_E \overset{[R(ER)]_E}{\twoheadrightarrow} \overset{[ER]_E}{\twoheadleftarrow} [v]_E$.

*Equational case*. Remember that $[u]_E \xrightarrow{[R]_E} [v]_E$ is defined as $u \overset{E}{\twoheadleftarrow\!\!\twoheadrightarrow} \xrightarrow{R} \overset{E}{\twoheadleftarrow\!\!\twoheadrightarrow} v$.

Suppose that $[u]_E$ is a normal form for $\xrightarrow{[R(ER)]_E}$, namely there does not exist any $v$ such that $[u]_E \overset{[ER]_E}{\twoheadrightarrow} \xrightarrow{[R]_E} [v]_E$, i.e. $u(\overset{E}{\twoheadleftarrow\!\!\twoheadrightarrow} \xrightarrow{ER} \overset{E}{\twoheadleftarrow\!\!\twoheadrightarrow})^* \xrightarrow{R} \overset{E}{\twoheadleftarrow\!\!\twoheadrightarrow} v$.

Now consider a derivation $[u]_E \overset{[ER]_E}{\twoheadleftarrow} \xrightarrow{[R]_E} [v]_E$, i.e. $u(\overset{E}{\twoheadleftarrow\!\!\twoheadrightarrow} \overset{ER}{\leftrightarrow} \overset{E}{\twoheadleftarrow\!\!\twoheadrightarrow})^* \xrightarrow{R} \overset{E}{\twoheadleftarrow\!\!\twoheadrightarrow} v$, which can be written in the more compact form $u \overset{ER/E}{\twoheadleftarrow\!\!\twoheadrightarrow} \xrightarrow{R} \overset{E}{\twoheadleftarrow\!\!\twoheadrightarrow} v$:

- If no such derivation exists, then $[u]_{E\cup ER}$ is a normal form for $\xrightarrow{[R]_{E\cup ER}}$, by definition.
- If such a derivation exists, then there must be at least one "backward" $\xleftarrow{ER}$ step, since we supposed $[u]_E$ to be a normal form for $\xrightarrow{[R(ER)]_E}$. Consider the last of these steps, we have

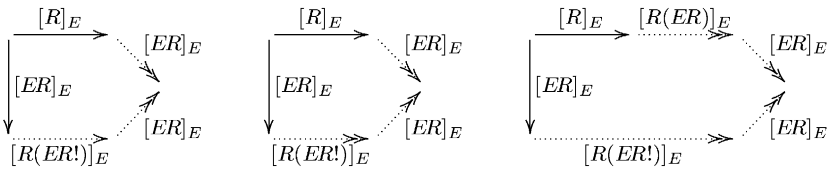


by equational coherence. Since we supposed that $[u]_E$ is a normal form for $\xrightarrow{[R(ER)]_E}$, the derivation $(a)$ is empty, hence there is a derivation $u \overset{E\cup ER}{\leftrightarrow\!\!\!\!\twoheadrightarrow} v$, which means that $[u]_{E\cup ER} = [v]_{E\cup ER}$ and there is a cycle $[u]_{E\cup ER} \xrightarrow{[R]_{E\cup ER}} [v]_{E\cup ER}$.
This ends the proof of Theorem 5. □

### 5.2. Local coherence

In order to be able to check coherence syntactically (by looking only at the rewrite systems $R$ and $ER$), we first define the local versions of strong, weak and equational coherence:

**Definition 6** (Local coherence). $\xrightarrow{[R(ER)]_E}$ is locally strongly (1), locally weakly (2) or locally equationally (3) coherent if




*(1) local strong coherence* *(2) local weak coherence* *(3) local equational coherence*

Note that coherence diagrams use the relation $\xrightarrow{[R(ER)]_E}$, while local coherence diagrams use $\xrightarrow{[R(ER!)]_E}$, i.e. when closing local coherence diagrams we request that terms be in $ER$-normal form before a rule of $R$ may be applied. This is a change from [40], allowing to discharge termination hypotheses on $R$.

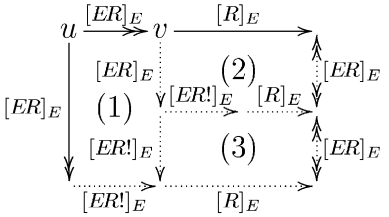
Now we boil down coherence to local coherence:

**Theorem 7** (Local coherence). *Remember that we suppose* $\xrightarrow{[ER]_E}$ *convergent.*

- *If* $\xrightarrow{[R(ER)]_E}$ *is locally strongly coherent, then* $\xrightarrow{[R(ER)]_E}$ *is strongly coherent.*
- *If* $\xrightarrow{[R(ER)]_E}$ *is locally weakly coherent, then* $\xrightarrow{[R(ER)]_E}$ *is weakly coherent.*
- *If* $\xrightarrow{[R(ER)]_E}$ *is locally equationally coherent, then* $\xrightarrow{[R(ER)]_E}$ *is equationally coherent.*

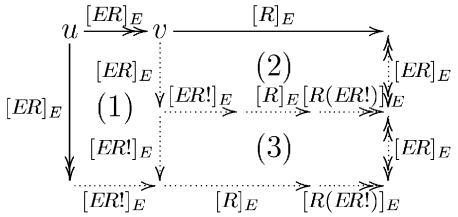
**Proof.** In all cases we use the fact that $\xrightarrow{[ER!]_E} \subseteq \xrightarrow{[ER]_E}$.

*Strong case.* Consider a derivation $\xleftarrow{[ER]_E}[u]_E \xrightarrow{[ER]_E}[v]_E \xrightarrow{[R]_E}$. Either $[v]_E$ is a $\xrightarrow{[ER]_E}$-normal form and the result is trivial, or we have



(1) convergence of $\xrightarrow{[ER]_E}$
(2) local strong coherence
(3) uniqueness of $\xrightarrow{[ER]_E}$-normal forms

*Weak case.* Consider a derivation $\xleftarrow{[ER]_E}[u]_E \overset{[ER]_E}{\twoheadrightarrow} [v]_E \xrightarrow{[R]_E}$. Either $[v]_E$ is a $\xrightarrow{[ER]_E}$-normal form and the result is trivial, or we have two cases: either the local coherence diagram is closed with an empty $\overset{[R(ER!)]_E}{\twoheadrightarrow}$ derivation, and again the result is trivial, or we have




(1) convergence of $\xrightarrow{[ER]_E}$
(2) local weak coherence with at least one $\xrightarrow{[R(ER!)]_E}$-step
(3) uniqueness of $\xrightarrow{[ER]_E}$-normal forms

*Equational case.* Similar to the weak case. □

### 5.3. Checking local coherence

Now, checking coherence reduces to ensuring that for all *critical peaks* $\xleftarrow{[ER]_E}\ \xrightarrow{[R]_E}$, the definition of local strong, local weak or local equational coherence is verified.

Checking all the possible critical peaks can be reduced to checking *critical pairs* modulo $E$ using the permutation lemma (Lemma 3), which is feasible when they are in finite number (this is the case for free and AC symbols; in the presence of A symbols, finiteness cannot be guaranteed).

Checking local coherence properties reduces to ensuring that the property is verified for all critical pairs, and that it is true in the non-superposition case. The first part depends on the particular rewrite theory and must be checked individually, but we can give general results for the second part.

Note that the results of this section are based on the permutation lemma, and thus are only valid for $E$ containing axioms for A and AC symbols.

- *Equational coherence*. The definition of equational coherence is verified in the non-superposition case.
- *Weak coherence*. The definition of weak coherence is not necessarily verified in the non-superposition case: no $\overset{[R(ER)]_E}{\twoheadrightarrow}$ derivation is allowed after the first $\xrightarrow{[R(ER)]_E}$ step.

  This is guaranteed by the permutation lemma if a non-left-linear rule of $ER$ cannot be applied above a rule of $R$. We will see some sufficient conditions below.

- *Strong coherence*. Similarly, the definition of strong coherence is not necessarily verified in the non-superposition case: the diagram has to be closed by exactly one $\overset{[R(ER)]_E}{\longrightarrow}$ step.

  This is guaranteed by the permutation lemma if any rule of $ER$ that can be applied above a rule of $R$ is left-linear, right-linear and regular (the same set of variables appears in the left- and right-hand sides).

There is no general way of ensuring the conditions about non-linear rules, but we can, however, give sufficient conditions ensuring that such critical peaks will never happen. The idea is to restrict the set of possible terms in such a way that a rule of $ER$ can never apply above a rule of $R$. For instance, separate "deterministic" and "non-deterministic" function symbols, and impose sort conditions so that the former cannot appear above the latter. This reduces somewhat the expressiveness, but nevertheless allows to handle the case of abstract data types: terms are built upon constructors and defined symbols, the constructors having to be "deterministic". It also includes the example of LOTOS (Section 10.3): processes may contain values, but not the opposite.

It is not possible to impose conditions on rules, except the very brutal one that forbids non-linear rules in $ER$, since the kind of critical peaks that we have to avoid does not correspond to a superposition of left-hand sides of rules.

In the next section, we give some examples of what may happen when the linearity conditions are not verified.

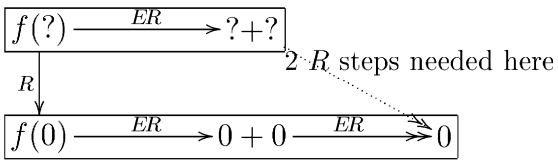
### 5.4. Examples of non-linearity

Consider

$$ER = \begin{cases} 0+0 \to 0 \\ f(x) \to x+x \\ g(x,x) \to x \end{cases} \qquad R = \begin{cases} ? \to 0 \\ ? \to 1 \end{cases}$$

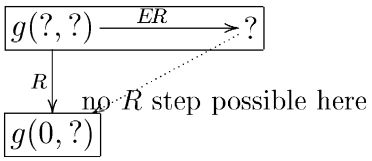
$ER$ is convergent and there are no critical pairs between $R$ and $ER$.

*Non-right-linearity*: There is an equational rewrite step $[f(?)]_{ER} \overset{[R]_{ER}}{\longrightarrow} [f(0)]_{ER}$, that we cannot simulate by a single rewrite step if we start from the representative $?+? \in [f(?)]_{ER}$. Two $\overset{R}{\to}$ steps would be needed, hence $\overset{[R]_{ER}}{\longrightarrow}$ is not strongly coherent:




*Non-left-linearity*: Though there is an equational rewrite step $[g(?,?)]_{ER} \overset{[R]_{ER}}{\longrightarrow} [g(0,?)]_{ER}$, there is no derivation $? \overset{R}{\twoheadrightarrow} t \in [g(0,?)]_{ER}$, hence $\overset{[R]_{ER}}{\longrightarrow}$ is not even weakly

coherent:




### 5.5. Coherence completion

If not all critical pairs are coherent, we can try to achieve coherence by adding new rules to $R$ according to the inference rule

$$(Deduce) \quad R \vdash R \cup \{u!_{ER} \to v!_{ER}\} \quad \text{if } (u,v) \in CP_E(R,ER)$$

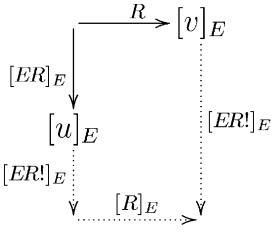
(Deduce) adds rules to $R$ in order that all critical peaks can be closed according to the local equational coherence diagram. $u!_{ER}$ is a representative of the $\overset{[ER]_E}{\longleftarrow}$-normal form of $[u]_E$.

Applying this rule as much as possible to an original system $R_0$ may not terminate, but if it does it provides an equivalent system which is locally equationally coherent:

**Theorem 8.** *Suppose that $R_0 \vdash \cdots \vdash R_n$ using the above inference rule, with the requisite that (Deduce) is never applied twice to the same critical pair, and does not apply to $R_n$. Then $\overset{[R_n]_{E\cup ER}}{\longrightarrow} = \overset{[R_0]_{E\cup ER}}{\longrightarrow}$ and $\overset{[R_n]_E}{\longrightarrow}$ is locally equationally coherent with $\overset{[ER]_E}{\longrightarrow}$.*

**Proof.** One can easily check that if $R_i \vdash R_{i+1}$ using the rule (Deduce), then $\overset{[R_i]_{E\cup ER}}{\longrightarrow} = \overset{[R_{i+1}]_{E\cup ER}}{\longrightarrow}$.

If (Deduce) does not apply to $R_n$, then all critical pairs in $CP_E(R_n, ER_n)$ have already been computed. A rule added by (Deduce) cannot disappear later, hence for each $(u,v) \in CP_E(R_n, ER_n)$ we have




This diagram is an instance of the local equational coherence diagram, thus $\overset{[R_n(ER)]_E}{\longrightarrow}$ is equationally coherent by the permutation lemma and Theorem 7.

For the cases of weak and strong coherence, the same completion procedure can be used, but one still has to check separately that the conditions about critical peaks involving non-linear rules are verified.

This completion procedure is not related to the classical Knuth–Bendix completion procedure. In particular:

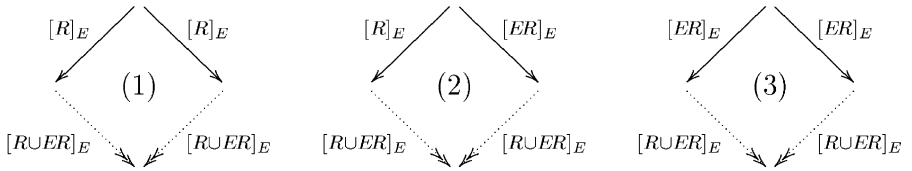
- Critical pairs are computed between two systems, not within one.
- (Deduce) generates oriented rules, not equations.
- The procedure does not rely on a given ordering, rather the orientation of the new rules is determined by the orientation of the rules already in $R$. □

### 5.6. Completion modulo a rewrite system

In the particular case where $R$ (in addition to $ER$) has an equational interpretation, there is a strong relationship between coherence completion and classical Knuth–Bendix completion:

**Theorem 9.** *Remember that we always suppose* $\xrightarrow{[ER]_E}$ *confluent. Suppose that* $\xrightarrow{[R\cup ER]_E}$ *terminates. If* $\xrightarrow{[R]_E}$ *is confluent and* $\xrightarrow{[R(ER)]_E}$ *is equationally coherent, then* $\xrightarrow{[R\cup ER]_E}$ *is confluent.*

**Proof.** Any of the following local critical peaks can be closed




(1) by confluence of $\xrightarrow{[R]_E}$, (2) by equational coherence of $\xrightarrow{[R(ER)]_E}$, and (3) by confluence of $\xrightarrow{[ER]_E}$. Since $\xrightarrow{[R\cup ER]_E}$ terminates, applying the above properties to a critical peak $u \overset{[R\cup ER]_E}{\twoheadleftarrow} \overset{[R\cup ER]_E}{\twoheadrightarrow} v$ eventually terminates and we get $u \overset{[R\cup ER]_E}{\twoheadrightarrow} \overset{[R\cup ER]_E}{\twoheadleftarrow} v$. □

## 6. Conditional rules

Until now we have considered only the non-conditional fragment of rewriting logic. In the presence of conditional rules, Theorems 5 and 7 are still valid since they refer only to abstract relations. The question now is how to prove local coherence (Section 5.3), since the permutation lemma from Section 4 does not hold for

conditional systems, as shown by the following example [4]: [2]

$$f(x) \to a \quad \text{if } x = f(x)$$
$$b \to f(b)$$

There are no critical pairs, but the critical peak $a \leftarrow f(f(b)) \to f(a)$ cannot be closed.

However, the permutation lemma does still hold in many cases. Consider a critical peak $\underset{p_1}{\overset{\rho_1}{\leftarrow}} t \underset{p_2}{\overset{\rho_2}{\to}}$ where rule $\rho_1$ (resp. $\rho_2$) is applied at position $p_1$ (resp. $p_2$) in a flattened term $t$, where $\rho_1$ is a conditional rule. Let us call "conditional part of $t$" the set of subterms of $t$ corresponding to instances of variables appearing in the conditional part of $\rho_1$. The problem appears in the non-superposition case, if $p_2$ is in the conditional part of $t$: the application of $\rho_2$ may modify the conditional part, and the condition of $\rho_1$ may not be true anymore. In all other cases, the conditional part is not affected and the permutation lemma still holds.

A number of strategies can be devised in order to avoid this problematic case:

- Impose syntactic restrictions on the rewrite systems, in order to guarantee that no rule will ever apply in the conditional part [4,10,18].
- Use the hierarchical approach [6] to guarantee that the evaluation of conditions is confluent.

## 7. Oriented matching

A well known technique for implementing a relation over equivalence classes $\xrightarrow{[R]_E}$ is to replace it by a weaker one $\xrightarrow{R,E}$ over terms. If an appropriate coherence property [3] is verified [23], then the former can be "simulated" by the latter. The major interest of using the relation $\xrightarrow{R,E}$ is that it can be implemented by matching modulo $E$.

We can extend these ideas to the case of matching modulo oriented rules. Define

$$\xrightarrow{R,(E\cup ER)} = \left( \underset{\geqslant p}{\overset{E}{\leftrightarrow}} \cup \underset{\geqslant p}{\overset{ER}{\longrightarrow}} \right)^* \underset{p}{\overset{R}{\to}}$$

where $\underset{p}{\to}$ denotes a step at position $p$ and $\underset{\geqslant p}{\to}$ a step at any position $q \geqslant p$ ($q$ below $p$).

This relation can be implemented by an *oriented matching algorithm* that, given two terms $u$ and $v$, returns a minimal set of most general substitutions $\sigma$ such that $\sigma u$ and $v$ are joinable by applying equations of $E$ and rules of $ER$ *according to their orientation*, i.e. $\sigma u (\underset{\geqslant p}{\overset{E}{\leftrightarrow}} \cup \underset{\geqslant p}{\overset{ER}{\to}})^* v$. Refer to [1] for the design of matching algorithms.

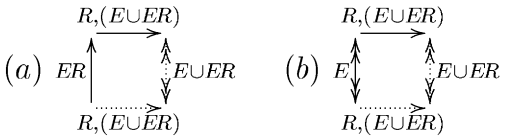
---

[2] A condition $a = b$ is usually interpreted as joinability ($a \xrightarrow{*} \xleftarrow{*} b$). In rewriting logic, $a = b$ is interpreted as reducibility ($a \xrightarrow{*} b$).

[3] The name "coherence" was originally inspired by Jouannaud and Kirchner [23], although the two notions of coherence are different: we consider two rewrite systems while Jouannaud and Kirchner [23] consider only one.

Under some conditions, the relation $\overset{R,(E\cup ER)}{\longrightarrow}$ can be used in place of $\overset{[R(ER)]_E}{\longrightarrow}$:

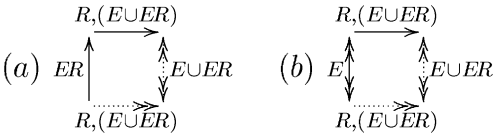
- *Strong case.* If the two following properties are verified




then $[u]_E \overset{[R(ER)]_E}{\longrightarrow} [v]_E \Leftrightarrow u \overset{R,(E\cup ER)}{\longrightarrow} v' \overset{E\cup ER}{\longleftrightarrow} v$.

**Proof.** $[u]_E \overset{[R\cup ER]_E}{\longrightarrow} [v]_E$ implies $u(\overset{E}{\twoheadleftarrow} \overset{ER}{\longrightarrow})^* \overset{E}{\twoheadleftarrow} \overset{R,(E\cup ER)}{\longrightarrow} \overset{E}{\twoheadleftarrow} v$. Apply repeatedly properties (a) and (b) to this derivation.

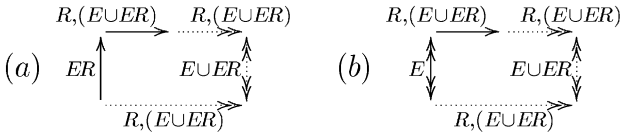
- *Weak case.* If $\overset{[R\cup ER]_E}{\longrightarrow}$ is terminating and the two following properties are verified:




then $[u]_E \overset{[R(ER)]_E}{\twoheadrightarrow} [v]_E \Leftrightarrow u \overset{R,(E\cup ER)}{\twoheadrightarrow} v' \overset{E\cup ER}{\twoheadleftarrow} v$.

The proof is similar to the one in the previous case, but requires the termination of $\overset{[R\cup ER]_E}{\longrightarrow}$.

- *Equational case.* If $\overset{[R\cup ER]_E}{\longrightarrow}$ is terminating and the two following properties are verified:




then $\overset{R,(E\cup ER)}{\longrightarrow}$ and $\overset{R(ER)/E}{\longrightarrow}$ have the same normal forms.

The proof goes as follows: suppose that $u \overset{R(ER)/E}{\longrightarrow} v$, using the two properties above, one can show that there is a derivation $\overset{R,(E\cup ER)}{\longrightarrow}$ starting from $u$, and that if this derivation is empty, then there is a cycle $u \overset{[R\cup ER]_E}{\twoheadrightarrow} u$, which contradicts the termination hypothesis.

If $ER = \emptyset$, we are in the usual situation of rewriting modulo an equational theory, and property (b) for the equational case coincides with the notion of coherence of [23], from which we can borrow the techniques to check or ensure property (b).

Property (a) is more unusual. Seemingly, checking it involves critical pairs between a left-hand side of $R$ and a right-hand side of $ER$. We can notice, however, that it holds in an important subcase, including identity and idempotency rules. Assume that all right-hand sides of the rules in $ER$ are variables, and consider a derivation

$$u \underset{p}{\overset{ER}{\to}} \underset{q}{\overset{R,(E\cup ER)}{\to}} v \Leftrightarrow u \underset{p}{\overset{ER}{\to}} \left( \underset{\geqslant q}{\overset{E}{\to}} \cup \underset{\geqslant q}{\overset{ER}{\to}} \right)^{*} \underset{q}{\overset{R}{\to}} v$$

If $p \geqslant q$, then $u \overset{R,(E\cup ER)}{\longrightarrow} v$, since the first $ER$ step is below $q$, and property (a) is trivially satisfied. In all other cases, the two steps commute (since by hypothesis the rules in $ER$ have variables as right-hand sides), and property (a) is verified for the weak and equational case. For the strong case, we also need to require that the variable forming the right-hand side of a rule appears exactly once in its left-hand side.

This technique extends to the case where only a subset of rules in $ER$ have variable right-hand sides, using the fact that if $ER = S \cup T$, then

$$\overset{[R(S\cup T)]_E}{\longrightarrow} \Leftrightarrow \overset{[R(S)(T(S))]_E}{\longrightarrow} \qquad \square$$

## 8. Rewriting modulo identity

As an example for equational coherence, we consider here the case of rewriting modulo identity, famous for its termination problems. Consider $R = \{f(x+y) \to f(x)+f(y)\}$ and $E = \{x+0 = x\}$. $\overset{[R]_E}{\longrightarrow}$ does not terminate: we have

$$[f(x)]_E \overset{[R]_E}{\to} [f(x)+f(0)]_E \overset{[R]_E}{\to} \cdots$$

since $f(x) \overset{E}{\leftrightarrow\!\!\!\!\twoheadrightarrow} f(x+0) \overset{R}{\longrightarrow} f(x)+f(0)$.

The problem here is that the equation $x+0=x$ can be used as expansion (from right to left). Rewriting modulo identity would better be specified as an oriented rule $x+0 \overset{=}{\to} x$ in $ER$. In this case, $\overset{[R(ER)]_E}{\longrightarrow}$ is not coherent, but coherence completion is able to find an equivalent strongly coherent system by adding a rule to $R$:

$$ER = \{x+0 \overset{=}{\to} x\}, \qquad R = \left\{ \begin{array}{c} f(x+y) \to f(x)+f(y) \\ f(x) \to f(x)+f(0) \end{array} \right\}$$

Note that the orientation of the rule $f(x) \longrightarrow f(x)+f(0)$ is fixed by the completion procedure.

$\overset{[R(ER)]_E}{\longrightarrow}$ is still non-terminating, but one cannot blame coherence (or identity) completion: since completion tries to simulate the relation $\overset{[R]_{E\cup ER}}{\longrightarrow}$, which does not terminate, one cannot expect to get a terminating relation. However, there is a solution if we are only interested in equational coherence. Suppose that $R$ contains additional rules for $f$,

for instance:

$$ER = \{x + 0 \xrightarrow{=} x\}, \qquad R = \left\{ \begin{array}{c} f(x+y) \to f(x) + f(y) \\ f(0) \to 0 \end{array} \right\}$$

Then $\xrightarrow{[R(ER)]_E}$ is equationally coherent, and $\xrightarrow{[R(ER)]_E}$ is terminating even though $\xrightarrow{[R]_{E \cup ER}}$ is not, basically because equational coherence does not require that we simulate the identity step

$$[f(x)]_{ER} \xrightarrow{[R]_{ER}} [f(x)]_{ER}$$

Of course, it would be impossible to get a terminating relation if we demand strong coherence (weak coherence is possible by adding rules to $ER$ instead of $R$).

## 9. Calculi with bound variables

Bound variables appear in many different calculi. In the $\lambda$-calculus, they are higher-order variables, bound by $\lambda$. In the $\pi$-calculus, they are channel names bound by $\nu$ and $\bar{x}$. Other examples of variable binders are the logical quantifiers $\forall$ and $\exists$, or the integration operator. In all these calculi, terms are taken up to the renaming of bound variables ($\alpha$-conversion), and substitution is considered as a meta-theoretic operator.

Trying to describe such calculi in a first-order setting, bound variables become just *names*, and substitution has to be made explicit as a first-order operator. Bound variables are usually replaced by De Bruijn indices in order to "build-in" $\alpha$-conversion. We end up with two different notions of variables and substitutions, however usually only ground terms (not containing first-order variables) are considered. First-order variables may be introduced for solving equations (e.g. higher-order unification [12]), with restrictions over where they may appear.

The explicit substitution approach has the advantage of modelling very precisely the possible implementations of a calculus. It is shown in [20] how different strategies for the application of substitution rules of the weak $\lambda\sigma$-calculus correspond to different models of machines implementing this calculus, and moreover that a machine can be derived quite easily from a given strategy.

Many substitution calculi have been proposed, with various confluence, termination or efficiency properties. A survey of substitution calculi for the $\lambda$-calculus can be found in [25]. As an example, we will consider here the $\lambda\upsilon$-calculus introduced in [3]; this choice is motivated by the fact that it is the simplest in the literature (it has the fewest number of rules and extra operators).

Bound variables are represented by De Bruijn indices $\underline{n}$ (they can be formally defined as Peano integers — for better readability, we sometimes subscript indices and their corresponding binders with variables names, as e.g. in $\lambda_x.\underline{0}_x$). A substitution operator $[\_/]$ is introduced, with the intuition that $[a/]$ corresponds to the substitution

$$\underline{1} \mapsto a \qquad \underline{2} \mapsto \underline{1} \quad \cdots \quad \underline{n+1} \mapsto \underline{n}$$

$\beta$-reduction is simulated by the only reduction rule

$$R = \{(\lambda a)b \to a[b/]\}$$

Note that the fact of putting $\beta$-reduction as a rule of $R$ does not imply that it cannot be equationally interpreted. If it is, then the partitioning of rules between $R$ and $ER$ is still useful as a hierarchical composition mechanism. At the lower level, one will see all the implementation details including the application of substitutions, while they will be hidden at the upper level.

The equational rules specify how to reduce terms containing substitutions (two auxiliary operators $\uparrow$ and $\Uparrow$ are introduced):

$$ER = \left\{ \begin{array}{lll} (ab)[s] & \xrightarrow{=} & a[s]b[s] \\ (\lambda a)s & \xrightarrow{=} & \lambda(a[\Uparrow(s)]) \\ \underline{1}[a/] & \xrightarrow{=} & a \\ \underline{n+1}[a/] & \xrightarrow{=} & \underline{n} \end{array} \quad \begin{array}{lll} \underline{1}[\Uparrow(s)] & \xrightarrow{=} & \underline{1} \\ \underline{n+1}[\Uparrow(s)] & \xrightarrow{=} & \underline{n}[s][\uparrow] \\ \underline{n}[\uparrow] & \xrightarrow{=} & \underline{n+1} \end{array} \right\}$$

It is shown in [3] that

- The $\lambda\upsilon$-calculus is a conservative extension of the $\lambda$-calculus: the relation $\xrightarrow{R} \xrightarrow{ER!}\!\!\!\!\rightarrow$ over pure terms (not containing $[\_/]$, $\uparrow$ or $\Uparrow(\_)$) coincides with $\beta$-reduction, hence also $\xrightarrow{R(ER!)}$ (putting aside the last $\xrightarrow{ER!}\!\!\!\!\rightarrow$ steps of a derivation).
- $\xrightarrow{ER}$ is convergent.
- $\xrightarrow{R\cup ER}$ is confluent over ground terms. This cannot be shown by the usual critical pair analysis, because there is a non-confluent critical pair between the rule in $R$ and the first rule of $ER$.

  If confluence over open terms is required, the system $\lambda\sigma_{\Uparrow}$ from [19] can be used, it is, however, much more complex than our example.

The relation $\xrightarrow{R(ER)}$ is equationally coherent over ground terms (or over all terms in the case of $\lambda\sigma_{\Uparrow}$): this is a trivial consequence of the confluence of $\xrightarrow{R\cup ER}$. It is not strongly coherent, because of the following critical peak:

$$\begin{array}{ccc} (ab)[(\lambda c)d] & \xrightarrow{\quad R \quad} & (ab)[c[d/]] \\ \Big\downarrow{\scriptstyle ER} & & \\ a[(\lambda c)d]b[(\lambda c)d] & & \end{array}$$

Two $\xrightarrow{R}$ steps are required in order to close the diagram.

We do not know about weak coherence: seemingly, proving it would require a careful examination of the proof of confluence of $\xrightarrow{R\cup ER}$. Weak coherence for $\lambda\sigma_{\Uparrow}$ can be proved or refuted by examining the critical pairs, but this would require an automated tool because of the great number of rules.

The same specification is also possible in the framework of classical rewriting logic [29], considering the equational rules as non-oriented axioms, but it is restricted

to a semantic description since rewriting modulo such a complex theory is not practically feasible. The fact of having two sets of rules is useful for relating $\beta$-reduction and its implementation in a single framework.

## 10. Process algebras

In this second example, we will develop the case of the $\pi$-calculus augmented with abstract data types.

The $\pi$-calculus [34] and LOTOS [5] are two modern representatives of the process algebra tradition that emerged from earlier works on CSP [21] and CCS [34]. The $\pi$-calculus describes mobile processes, where mobility is achieved by the introduction of binder operators and bound variables allowing name passing.

LOTOS integrates processes and data types in a two-level framework. The first is based on CSP/CCS, the second is based on ACT-ONE, a specification language for abstract data types. This layer structure is not completely satisfactory, as noticed in an introductory book [34, p. 70]:

> (...) many elements of a system can be specified both as processes and as data types. (...) a deeper understanding of the relation between the two components could be beneficial, and some harmonization between them could be attempted (...), for instance a common semantic model.

We will see here how an oriented rewrite theory provides such a common semantic model.

### 10.1. Structure of $\pi$-calculus processes

As in the previous section, the bound variables of the $\pi$-calculus will be represented by indices and the substitution operation will be made explicit using the substitution calculus part of $\lambda\upsilon$ (i.e. without $\beta$-reduction).

The terms of the $\pi$-calculus with indices are defined as follows (using a syntax more digestible to rewrite interpreters than the usual one):

- indices and substitution operators are as in the previous section.
- processes

$$P := \mathsf{nil} \mid (v)P \mid g.P \mid P_1 + P_2 \mid P_1 \mid P_2 \mid !P \mid P\sigma(\text{substitution})$$

- guards

$$\begin{array}{rcll} g := & \mathsf{in}(c) & & \text{input on channel } c \text{ (binder)} \\ \mid & \mathsf{out}(c,x) & & \text{output } x \text{ on channel } c \\ \mid & \mathsf{bout}(x) & & \text{bound output (binder)} \\ \mid & \tau & & \text{internal choice} \end{array}$$

Bound output can be defined in terms of other basic operators ($\mathsf{bout}(x).P = (v)_c\mathsf{out}(\underline{\mathbf{0}}_c, x).P$), but we choose to introduce it explicitly for technical reasons.

The structure of processes is defined by a set of equations $E$ and a set of equational rules $ER$.

Processes are considered modulo the equations AC(+) and AC(|) (associativity and commutativity of the + and | operators) and the following equational rules:

$$P + \mathsf{nil} \xrightarrow{=} P$$
$$P \mid \mathsf{nil} \xrightarrow{=} P$$
$$P \mid (v)\,Q \xrightarrow{=} (v)([\uparrow]P \mid Q)$$

The substitution rules of $\lambda\upsilon$ from the previous section are also part of the equational rules.

The last two equational rules allow to replace | and ! operators with non-deterministic choice:

*The expansion rule*: Let $P = \alpha_1.P_1 + \cdots + \alpha_n.P_n$ and $Q = \beta_1.Q_1 + \cdots + \beta_m.Q_m$, then

$$P \mid Q \quad \xrightarrow{=} \quad \sum_{i=1\ldots n} \alpha_i.(P_i \mid Q) + \sum_{i=1\ldots m} \beta_i.(P \mid Q_i) + \sum_{\substack{i=1\ldots n \\ j=1\ldots m}} Sync(\alpha_i.P_i, \beta_j.Q_j)$$

where

$$Sync(\alpha_i.P_i, \beta_j.Q_j) = \begin{cases} \tau.(P_i[z/]|Q_j) & \text{if } \alpha_i = \mathsf{in}(x) \text{ and } \beta_j = \mathsf{out}(x,z) \\ \tau.(v)(P_i|Q_j) & \text{if } \alpha_i = \mathsf{in}(x) \text{ and } \beta_j = \mathsf{bout}(x) \\ \text{(and similarly by swapping the arguments)} & \\ \mathsf{nil} \text{ in all other cases} & \end{cases}$$

*The replication rule*: Let $P = g_1.P_1 + \cdots + g_n.P_n$, then

$$!P \xrightarrow{=} g_1.(P_1|!P) + \cdots + g_n.(P_n|!P)$$

A process is in weak disjunctive normal form [34] if it is of the form

$$(v)\ldots(v)(g_1.P_1 + \cdots + g_n.P_n)$$

with $n \geqslant 0$. Using the above equational rules, any process can be put in weak disjunctive normal form.

Expansion and replication are rule schemes rather than proper rewrite rules because of the variable $n$, but can be easily simulated using extra hidden operators.

The rewrite relation defined by the above equational rules (modulo $AC$) does not terminate because of the replication rule that can be applied repeatedly into its own right-hand side. In order to ensure termination, the rewrite rules must be applied according to a strategy $\Sigma$ that allows application of the replication rule only when needed in order to compute a weak disjunctive normal form.

$\Sigma$ is defined as follows, considering a strategy as a function from terms to terms:

$$\begin{array}{lll} \Sigma(P) = & \text{if } P = nil & \text{then } P \\ & \text{if } P = g.Q & \text{then } P \text{ (for } g \text{ a guard)} \\ & \text{if } P = (v)Q & \text{then } (v)\Sigma(Q) \\ & \text{if } P = P_1 + P_2 & \text{then } \Sigma(P_1) + \Sigma(P_2) \\ & \text{if } P = P_1|P_2 & \text{then } expand(\Sigma(P_1)|\Sigma(P_2)) \\ & \text{if } P = !Q & \text{then } repl(\Sigma(P)) \end{array}$$

where $expand(P)$ applies the expansion theorem at the top of $P$ and $repl(P)$ the replication theorem. The remaining rules are applied freely. A simple induction shows that *expand* and *repl* are always applied to arguments where the corresponding rule can be applied, and the maximal number of application of $\Sigma$ is bounded by the size of the input term.

Let us denote $\overset{[ER]_E}{\mapsto}$ the derivations using all the above equational rules according to this strategy $\Sigma$, then we have the following correspondence result:

**Proposition 10** (Viry [41]). *There is a one-to-one correspondence between processes of the original $\pi$-calculus (modulo the usual structural axioms) and normal forms with respect to $\overset{[ER]_E}{\mapsto}$.*

**Proof.**

- By construction, there is a one-to-one correspondence between processes of the original $\pi$-calculus (modulo the usual structural axioms) and equivalence classes modulo $ER \cup E$.
- All critical pairs of $ER$ modulo $E$ are ground-confluent.

  The substitution calculus is confluent on ground terms, hence over processes since we did not introduce process variables (note that there also exists substitution calculi convergent even on open terms, for instance $\lambda\sigma_{\Uparrow}$-calculus [9,25]). All other critical pairs are easily shown to be confluent except for

$$\begin{array}{ccc} & (P \mid (\nu)\, Q)[s] & \\ {\scriptstyle *}\swarrow & & \searrow{\scriptstyle *} \\ (\nu)\,(P[\uparrow][\Uparrow(s)] \mid Q[\Uparrow(s)]) & & (\nu)\,(P[s][\uparrow] \mid Q[\Uparrow(s)]) \end{array}$$

  This critical pair is shown to be confluent on ground terms by Lemma 5 used in the proof of confluence of $\lambda\upsilon$-calculus in [26].
- The reduction relation $\overset{[ER]_E}{\mapsto}$ terminates.
- If there is a derivation leading to a normal, there is a derivation following the strategy $\Sigma$ leading to the same normal form, hence $\overset{[ER]_E}{\mapsto}$ computes unique normal forms. □

### *10.2. The reduction relation of the $\pi$-calculus*

Whereas the equations ($E$) and equational rules ($ER$) specify the structure of processes, the non-equational rules ($R$) are interpreted as irreversible transitions between states, specifying the internal transitions of processes.

The reduction relation of the $\pi$-calculus corresponds to internal transitions of processes (the so-called $\tau$-transitions). It is denoted $\overset{\tau}{\rightarrow}$ and is traditionally defined by

the following *inference rules* [22]:

$$\frac{}{\tau.P + Q \xrightarrow{\tau} P} \qquad \text{(Choice)} \qquad \frac{P \xrightarrow{\tau} P'}{P|Q \xrightarrow{\tau} P'|Q} \qquad \frac{P \xrightarrow{\tau} P'}{(\nu)P \xrightarrow{\tau} (\nu)P'}$$

If we want to consider (Choice) as a rewrite rule, we have to make sure that it can be applied only under the allowable contexts. Such a strategy can be easily implemented by first reducing processes with the equational rules, since we know the shape of weak disjunctive normal forms.

We will thus consider (Choice) as the only non-equational rule in $R$. Its application according to the above strategy will be denoted $\overset{[R]_E}{\mapsto}$. The correspondence result is as follows:

**Proposition 11** (Viry [41]). *There is a reduction step $P \xrightarrow{\tau} Q$ in the original $\pi$-calculus if and only if there is a rewrite derivation $[P]_E \overset{[ER]_E}{\mapsto} \overset{[R]_E}{\mapsto} [Q']_E$, where $[Q'] \overset{[ER]_E}{\leftrightarrow} Q$.*

This result is proved by showing strong coherence between equational and reduction rules (note that (Choice) is a linear rule). Although the rules in our specification must be applied according to some specific strategies, Theorems 5 and 7 are still valid since they refer only to abstract relations. As for the proof of local coherence, one has to ensure that the strategies are preserved by the permutation lemma, and that the local coherence diagram can be closed according to these strategies.

We have implemented the relation $\overset{[ER]_E}{\mapsto} \overset{[R]_E}{\mapsto}$ in ELAN [24], which offers a powerful strategy specification mechanism and a efficient AC-rewriting engine (we believe that our specification would be very difficult to encode using the OBJ [17] strategy mechanism).

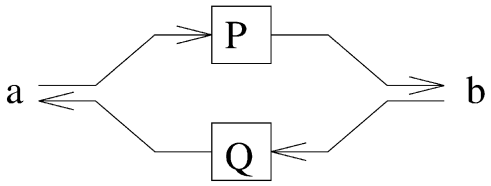
With the addition of a few new built-in operators, we have been able to design an efficient and semantically clean input–output system for ELAN based on this relation [41].

### *10.3. Addition of data types*

So far, we have considered the plain monadic $\pi$-calculus. As shown in [34], this calculus is powerful enough to encode any kind of structured data, so the game may end here. However, for practical purposes one would like to be able to define the data exchanged by processes using abstract data types (ADTs).

ADTs are straightforwardly encoded with equational rules, but we have to guarantee coherence in order for the above correctness results to still hold. As long as the operators of the ADTs are disjoint from those of the $\pi$-calculus, this does not introduce new critical pairs, and if we impose that data types appear only within processes, as in LOTOS, then the linearity conditions are trivially verified.

*10.4. Example: double-way buffer*




A double-way buffer consists of two processes in parallel, each of them repeatedly reading a data element from a channel and writing it on the other channel. These processes are most naturally specified using recursive equations:

$$P = \mathsf{in}(a)_x.\mathsf{out}(b, \underline{\mathbf{0}}_x).P$$
$$Q = \mathsf{in}(b)_x.\mathsf{out}(a, \underline{\mathbf{0}}_x).Q$$

We cannot implement directly recursive equations, and need to restate this definition using the replication operator. Note that this is always possible [34] and that the two specifications are weakly bisimilar (i.e. equivalent up to the internal actions). $P$ and $Q$ are redefined as

$$P = (\nu)_p(\mathsf{out}(\underline{\mathbf{0}}_p, \mathsf{void}).\mathsf{nil} \mid !\mathsf{in}(\underline{\mathbf{0}}_p).\mathsf{in}(a)_x.\mathsf{out}(b, \underline{\mathbf{0}}_x).\mathsf{out}(\underline{\mathbf{1}}_p, \mathsf{void}).\mathsf{nil})$$
$$Q = (\nu)_q(\mathsf{out}(\underline{\mathbf{0}}_q, \mathsf{void}).\mathsf{nil} \mid !\mathsf{in}(\underline{\mathbf{0}}_q).\mathsf{in}(b)_x.\mathsf{out}(a, \underline{\mathbf{0}}_x).\mathsf{out}(\underline{\mathbf{1}}_q, \mathsf{void}).\mathsf{nil})$$

where void is the only value of the single-valued type of channels that only exchange synchronizations. Intuitively, the process below the replication operator of $P$ (resp. $Q$) can only be "activated" by an input from channel $p$ (resp. $q$).

The process $P \mid Q$ reduces to the weak disjunctive normal form

$$(\nu)_p(\nu)_q(\mathsf{in}(a)_x.P' + \mathsf{in}(b)_x.Q')$$

with

$$\begin{aligned} P' = {} & \mathsf{out}(b, \underline{\mathbf{0}}_x).\mathsf{out}(p, \mathit{void}).\mathsf{nil} \\ & \mid \mathsf{in}(b)_x.\mathsf{out}(a, \underline{\mathbf{0}}_x).\mathsf{out}(q, \mathit{void}).\mathsf{nil} \\ & \mid !\mathsf{in}(\underline{\mathbf{0}}_p).\mathsf{in}(a)_x.\mathsf{out}(b, \underline{\mathbf{0}}_x).\mathsf{out}(\underline{\mathbf{1}}_p, \mathit{void}).\mathsf{nil} \\ & \mid !\mathsf{in}(\underline{\mathbf{0}}_q).\mathsf{in}(a)_x.\mathsf{out}(b, \underline{\mathbf{0}}_x).\mathsf{out}(\underline{\mathbf{1}}_q, \mathit{void}).\mathsf{nil} \end{aligned}$$

and similarly for $Q'$ by swapping $p$ with $q$ and $a$ with $b$.

This normal form exhibits the two possible external communications $\mathsf{in}(a)$ and $\mathsf{in}(b)$. As soon as one of them is possible, the communication takes place and the computation proceeds with either $P'$ or $Q'$. In $P'$, term (1) is the continuation of the buffer process $P$, term (2) is the buffer process $Q$, and terms (3) and (4) are the "pools" of processes that are activated by an input on channel $p$ or $q$.

So far we remained within the framework of the $\pi$-calculus. As an example of the addition of ADTs, the two-way buffer example can be extended by adding computation of the output values, for instance

$$P = \mathsf{in}(a)_x.\mathsf{out}(b, f(\underline{\mathbf{0}}_x)).P$$
$$Q = \mathsf{in}(b)_x.\mathsf{out}(a, g(\underline{\mathbf{0}}_x)).P$$

where $f$ and $g$ are functions defined by equational rules. Since these defined operators appear only strictly below process operators, we are guaranteed that strong coherence is preserved and thus that our implementation remains correct.

### *10.5. Example: filter*

In the previous example, in order to enforce the non-linearity conditions for strong coherence, data types appear only below processes. Actually, the non-linearity conditions are verified as well if process expressions appear below defined symbols, as long as no non-linear rewrite rule may ever be applied above a process expression.

This is the case for instance with the if...then...else... operator, defined by the following linear rules

$$\mathsf{if\ true\ then}\ x\ \mathsf{else}\ y \to x$$
$$\mathsf{if\ false\ then}\ x\ \mathsf{else}\ y \to y$$

Then we can write a FILTER process, that repeatedly inputs values on a channel $i$ and copies them on an output channel $o$ only when they verify a given condition:

$$\mathrm{FILTER} = \mathsf{in}(i)_x.\ \mathsf{if}\ c(x)\ \mathsf{then}\ \mathsf{out}(o, \underline{\mathbf{0}}_x).\mathrm{FILTER}\ \mathsf{else}\ \mathrm{FILTER}$$

This recursive definition is then restated using the replication operator as in the previous example.

This possibility allows for a much more "natural" programming style, similar to CSP/Occam [21], but the constructs that may appear above processes must be clearly identified in order to ensure the condition about non-linear rules.

## 11. Sequent calculus modulo

Our third and last example concerns the case where the rewrite relation is interpreted as deduction steps.

### *11.1. An oriented rewriting logic theory for the sequent calculus*

The sequent calculus is a quite complex logical system which has been thoroughly studied in the literature. Our motivation for choosing to study this particular calculus is to show a non-trivial example; other than that, this choice is purely arbitrary, and in particular we will not develop logical considerations but will rather remain on a purely syntactic level.

$$
R = \left\{
\begin{array}{ll}
\text{Axiom} & \underline{A},\ C \vdash C, \underline{B} \to \diamond \\
\mathscr{L}\forall & \underline{A}, \forall\zeta.C \vdash \underline{B} \to \underline{A}, C[a/\zeta] \vdash \underline{B}
\end{array}
\right.
$$

$$
ER = \Sigma \cup \left\{
\begin{array}{ll}
\mathscr{L}\vee & \underline{A},\ C \vee D \vdash \underline{B} \xrightarrow{=} \underline{A}, C \vdash \underline{B} \ \bullet\ \underline{A}, D \vdash \underline{B} \\
\mathscr{R}\wedge & \underline{A} \vdash C \wedge D, \underline{B} \xrightarrow{=} \underline{A} \vdash C, \underline{B} \ \bullet\ \underline{A} \vdash D, \underline{B} \\
\mathscr{R}\forall & \underline{A} \vdash \forall\zeta.C, \underline{B} \xrightarrow{=} \underline{A} \vdash C, \underline{B} \\
 & \\
Uc & \underline{A}, \Box \xrightarrow{=} \underline{A} \\
U\bullet & \underline{A} \bullet \diamond \xrightarrow{=} \underline{A} \\
 & \\
f\neg & \neg\neg A \xrightarrow{=} A \\
f\Rightarrow & A \Rightarrow B \xrightarrow{=} \neg A \vee B \\
 & \\
U\vee & A \vee 0 \xrightarrow{=} A \\
U\wedge & A \wedge 1 \xrightarrow{=} A \\
 & \\
Z\vee & A \vee \neg A \xrightarrow{=} 1 \\
Z\wedge & A \wedge \neg A \xrightarrow{=} 0 \\
 & \\
fI\wedge & A \wedge A \xrightarrow{=} A \\
fI\vee & A \vee A \xrightarrow{=} A \\
 & \\
f\forall & \forall x.A \xrightarrow{=} \neg\exists x.\neg A \\
 & \\
sIl & \underline{A}, C, C \vdash \underline{B} \xrightarrow{=} \underline{A}, C \vdash \underline{B} \\
sIr & \underline{A} \vdash C, C, \underline{B} \xrightarrow{=} \underline{A} \vdash C, \underline{B} \\
s\neg l & \underline{A}, C \vdash \underline{B} \xrightarrow{=} \underline{A} \vdash \neg C, \underline{B} \\
s\wedge & \underline{A}, C \wedge D \vdash \underline{B} \xrightarrow{=} \underline{A}, C, D \vdash \underline{B} \\
s\vee & \underline{A} \vdash C \vee D, \underline{B} \xrightarrow{=} \underline{A} \vdash C, D, \underline{B}
\end{array}
\right.
$$

$$
E = \left\{
\begin{array}{ll}
Ac & A, (B, C) = (A, B), C \\
Cc & A, B = B, A \\
 & \\
A\bullet & \underline{A} \bullet (\underline{B} \bullet \underline{C}) = (\underline{A} \bullet \underline{B}) \bullet \underline{C} \\
C\bullet & \underline{A} \bullet \underline{B} = \underline{B} \bullet \underline{A}
\end{array}
\right.
$$

Fig. 1. An oriented rewrite theory for the sequent calculus. $\Sigma$ is the substitution part of $\lambda\upsilon$, as in the previous section. Underlined variables range over multisets of formulas. $\Box$ and $\diamond$ denote, respectively, empty multisets of formulas and sequents.

A rewriting logic presentation of the sequent calculus can be found in [29]. In [42], we showed that the sequent calculus (as defined in [16], assuming identity elements for $\vee$ and $\wedge$) can be presented by the oriented rewrite theory $(E, ER, R)$ shown in Fig. 1, where $E$ only contains associativity and commutativity axioms. This rewrite theory was obtained as follows:

- Put in $E$ the equations making explicit the implicit structure of formula and sequent composition (such as associativity and commutativity) together with the equations defining the explicit substitution calculus. Turn all inference rules of the original sequent calculus definition into $R$ rules:

$$\frac{\textit{premisses}}{\textit{consequences}} \quad \text{becomes} \quad \textit{consequences} \rightarrow \textit{premisses}$$

  A proof of a sequent $\Gamma$ is thus encoded into a rewriting logic sentence $\alpha : \Gamma \rightarrow \diamond$, corresponding to a rewrite derivation $[\Gamma]_E \overset{[R]_E}{\twoheadrightarrow} [\diamond]_E$.
- Then, add to $E$ equational consequences of this system, such as idempotency of formula composition. This allows in turn to simplify the presentation by removing rules in $R$ that have become redundant.
- Finally, turn selected equations of $E$ and selected rules of $R$ into equational rules in $ER$. This selection (and the orientation given to equations) is rather arbitrary: the only requirement is that the resulting oriented rewrite theory be weakly coherent, and one also usually wishes to keep in $E$ only axioms for A and AC symbols.

  A different selection or a different orientation of equations would provide a different view of sequent calculus proofs. For instance, moving a rule from $R$ to $ER$ corresponds to what is usually called "building-in equality" in theorem provers [7]. Building-in equality can make proofs much smaller, at the expense of hiding logical details.

The main result of [42] states that:

**Proposition 12.** *The sequent $\Gamma$ is provable in the sequent calculus if and only if there is a rewrite derivation* $[\Gamma]_E \overset{[R(ER)]_E}{\twoheadrightarrow} [\diamond]_E$.

**Proof.**

- By construction, and using two lemmas stating that adding equational consequences and removing redundant rules preserves derivability, we have that the sequent $\Gamma$ is provable in the sequent calculus if and only if there is a rewrite derivation $[\Gamma]_{E \cup ER} \overset{[R]_{E \cup ER}}{\twoheadrightarrow} [\diamond]_{E \cup ER}$ (where $\diamond$ denotes the empty sequent multiset).
- $\overset{ER}{\longrightarrow}$ is convergent and $\overset{[R(ER)]_E}{\longrightarrow}$ is weakly coherent.

  All critical pairs of $ER$ are confluent and termination is shown using an AC-compatible recursive path ordering [2].

  Weak coherence is shown by examining critical pairs. Since we reduce multisets of sequents, and rules of $R$ only apply at the "sequent level", the situation where a non-left linear rule of $ER$ is applied above a rule of $R$ cannot happen. □

### *11.2. Theorem proving modulo*

Now, we can think of adding additional equations or equational rules modulo which we wish to do reasoning. These equations may for instance define new operators or specify the internal structure of the objects we reason about. By doing so, we end up with a framework similar to that of "theorem proving modulo" introduced in [13].

As long as weak coherence is still verified, new equations and equational rules can be added, while preserving the correctness of the implementation. Because the additional equations one may wish to introduce most often involve only new symbols not present in the original calculus, weak coherence is trivially preserved (no new critical pairs are introduced), and termination of $\xrightarrow{[ER]_E}$ may be proved using modularity results [36,15].

In this way, we can handle the example of higher-order logic given in [13]. The formulas of the sequent calculus are extended with a few operators, and considered modulo equations. Intuitively, the operators $\dot{\neg}$, $\dot{\vee}$ and $\dot{\forall}$ are external versions of their sequent calculus counterparts, $\alpha$ denotes application, and $\varepsilon$ maps the external version of a term to its internal version. The three equations giving the specification of $\varepsilon$ are better expressed as equational rules:

$$\varepsilon(\alpha(\dot{\neg}, x)) \xrightarrow{=} \neg\varepsilon(x)$$

$$\varepsilon(\alpha(\alpha(\dot{\vee}, x), y)) \xrightarrow{=} \varepsilon(x) \vee \varepsilon(y)$$

$$\varepsilon(\alpha(\dot{\forall}_T, x)) \xrightarrow{=} \forall y.\varepsilon(\alpha(x, y))$$

Then it is possible to show that *ER* augmented with these rules is still convergent and that weak coherence is preserved, hence we immediately obtain a correct implementation of this extended sequent calculus modulo equations.

## 12. Conclusion

The introduction of equational rules solves the dilemma, inherent to rewriting logic, of whether to specify something as a rule or as an equation. Turning equations into equational rules can lead to better efficiency (some rules can even be internalized into an oriented matching algorithm), while turning rules into equational rules can be seen as "building-in" equality. Equational rules can also serve as a way of hierarchically structuring systems.

When using equational rules, one must ensure that their operational behavior corresponds to their intended meaning. We proposed various sufficient conditions (various notions of coherence) and verification techniques based on critical pair criteria.

The three examples developed in the paper cover the most important interpretations of rewriting logic (transitions, deductions and equality), and should provide a template for most applications. In each case, we obtained an oriented rewrite theory containing only the associativity and commutativity axioms, which can be easily implemented on usual rewriting interpreters or compilers.

Rewriting logic as a logical framework offers a natural and simple way of specifying most calculi encountered in practice, but the lack of equational rules makes its implementation difficult. We believe that by solving this behavior vs. semantics dilemma, we removed one of the most important barriers in the way of a wide practical use of rewriting logic.

## Acknowledgements

My thanks go to the anonymous referees who provided many constructive comments.

## References

[1] M. Adi, C. Kirchner, AC-unification race: the system solving approach, implementation and benchmarks, J. Symbolic Comput. 14 (1) (1992) 51–70.
[2] F. Baader, T. Nipkow, Term Rewriting and All That, Cambridge University Press, Cambridge, 1999.
[3] Z. Benaissa, D. Briaud, P. Lescanne, J. Rouyer-Degli, $\lambda v$, a calculus of explicit substitutions which preserves strong normalisation, Technical Report 2477, INRIA, February 1995.
[4] J.A. Bergstra, J.W. Klop, Conditional rewrite rules: confluency and termination, J. Comput. System Sci. 32 (3) (1986) 323–362.
[5] T. Bolognesi, E. Brinksma, Introduction to the ISO specification language LOTOS. in: P.H.J. van Eijk, C.A. Vissers, M. Diaz (Eds.), The Formal Description Technique LOTOS, Elsevier Science Publishers B.V., North-Holland, 1989, pp. 23–73.
[6] W. Bousdira, J.-L. Rémy, Hierarchical contextual rewriting with several levels, in: R. Cori, M. Wirsing (Eds.), 5th Ann. Symp. on Theoretical Aspects of Computer Science, Bordeaux, France, Lecture Notes in Computer Science, Vol. 294, Springer, Berlin, 1988, pp. 193–206.
[7] S. Boutin, Using reflection to build efficient and certified decision procedures, in: M. Abadi, T. Ito (Eds.), TACS'97, Lecture Notes in Computer Science, Vol. 1281, Springer, Berlin, 1997.
[8] H. Comon, J.-P. Jouannaud, Unification and disunification in shallow theories, in: H. Comon (Ed.), Proc. 5th Internat. Workshop on Unification, Barbizon, France, 1991.
[9] P.-L. Curien, Th. Hardin, J.-J. Lévy, Confluence properties of weak and strong calculi of explicit substitutions, RR 1617, INRIA, Rocquencourt, February 1992.
[10] N. Dershowitz, M. Okada, G. Sivakumar, Confluence of conditional rewrite systems, in: J.-P. Jouannaud, S. Kaplan (Eds.), Proc. 1st Internat. Workshop on Conditional Term Rewriting Systems, Orsay, France, Lecture Notes in Computer Science, Vol. 308, Springer, Berlin, July 1987, pp. 31–44.
[11] E. Domenjoud, Outils pour la déduction automatique dans les théories associatives–commutatives, Thèse de Doctorat d'Université, Université de Nancy 1, September 1991.
[12] G. Dowek, Th. Hardin, C. Kirchner, Higher-order unification via explicit substitutions, in: D. Lugiez (Ed.), Proc. 8th Internat. Workshop on Unification, Val d'Ajol, France, CRIN, June, 1994.
[13] G. Dowek, Th. Hardin, C. Kirchner, Theorem proving modulo, Technical Report 3400, INRIA, 1998.
[14] F. Durán, Coherence checker and completion tools for maude specifications, Technical report, Dpto. de Lenguajes y Ciencias de la Computation, Universidad de Málaga, 2000, Available from `http://maude.csl.sri.com/papers/coherence`.
[15] M. Fernandez, J.-P. Jouannaud, Modular termination of term rewriting systems revisited, in: Proc. of 11th Workshop on Specification of Abstract Data Types, Lecture Notes in Computer Science, Vol. 906, Springer, Berlin, 1995.
[16] J.-Y. Girard, Y. Lafont, P. Taylor, Proofs and types, Cambridge Tracts in Theoretical Computer Science, Vol. 7, Cambridge University Press, Cambridge, 1989.
[17] J.A. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, J.-P. Jouannaud, Introducing OBJ, in: J.A. Goguen, G. Malcolm (Eds.), Software Engineering with OBJ: Algebraic Specification in Action, Kluwer Academic Publishers, Dordrecht, 2000, pp. 3–167.
[18] B. Gramlich, On termination and confluence of conditional rewrite systems, in: Proc. 4th Internat. Workshop on Conditional and Typed Rewriting Systems (CTRS'94), Jerusalem, Lecture Notes in Computer Science, Vol. 968, Springer, Berlin, 1994.
[19] Th. Hardin, J.-J. Lévy, A confluent calculus of substitutions, in: France–Japan Artificial Intelligence and Computer Science Symposium, Izu, 1989.
[20] T. Hardin, L. Maranget, B. Pagano, Functional Back-Ends within the Lambda-Sigma Calculus. Proceedings of ICFP 1996, Philadelphia, Pennsylvania, May 24–26, 1996, SIGPLAN Notices 31(6), June 1996, ACM Press, ISBN 0-89791-770-7.

[21] C.A.R. Hoare, Communicating sequential processes, Commun. ACM 21 (8) (1978) 666–677.
[22] K. Honda, N. Yoshida, On reduction-based process semantics, in: Proc. of 13th Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science, Vol. 761, Springer, Berlin, 1993, pp. 371–387.
[23] J.-P. Jouannaud, H. Kirchner, Completion of a set of rules modulo a set of equations, SIAM J. Comput. 15(4) (1986) 1155–1194, Preliminary version in Proc. 11th ACM Symposium on Principles of Programming Languages, Salt Lake City, USA, 1984.
[24] C. Kirchner, H. Kirchner, M. Vittek, Designing constraint programming languages using computational systems, in: P. Van Hentenryck, S. Saraswat (Eds.), Principles and Practice of Constraint Programming, The MIT Press, Cambridge, MA, 1995.
[25] P. Lescanne, From $\lambda\sigma$ to $\lambda\upsilon$, a journey through calculi of explicit substitutions, in: Hans Boehm (Ed.), Proc. of the 21st Annual ACM Symposium on Principles of Programming Languages, Portland, OR, USA, ACM, 1994, pp. 60–69.
[26] P. Lescanne, J. Rouyer-Degli, The calculus of explicit substitutions $\lambda\upsilon$, Technical Report RR-2222, INRIA-Lorraine, January 1994.
[27] C. Marché, Réécriture modulo une théorie présentée par un système convergent et décidabilité du problème du mot dans certaines classes de théories équationnelles, Thèse de Doctorat d'Université, Université de Paris-Sud, Orsay, France, October 1993.
[28] C. Marché, Normalised rewriting and normalised completion, in: Proc. 9th Symp. on Logic in Computer Science, IEEE Computer Society Press, Silverspring, MD, 1994, pp. 394–403.
[29] N. Martí-Oliet, J. Meseguer, Rewriting logic as a logical and semantic framework, Technical report CSL-93-05, SRI International, 1993.
[30] The Maude system, `http://maude.csl.sri.com`.
[31] J. Meseguer, Conditional rewriting logic as a unified model of concurrency, Theoret. Comput. Sci. 96 (1) (1992) 73–155.
[32] J. Meseguer, A logical theory of concurrent objects and its realisation in the Maude language, in: G. Agha, P. Wegner, A. Yonezawa (Eds.), Research Directions in Object-Based Concurrency, The MIT Press, Cambridge, MA, 1993.
[33] R. Milner, Communication and Concurrency, International Series in Computer Science, Prentice-Hall, Englewood Cliffs, NJ, 1989.
[34] R. Milner, The polyadic $\pi$-calculus: a tutorial, Technical report ECS-LFCS-91-180, LFCS, University of Edinburgh, 1991.
[35] G. Peterson, M.E. Stickel, Complete sets of reductions for some equational theories, J. ACM 28 (1981) 233–264.
[36] A. Rubio, Extension orderings, in: Proc. of ICALP'95, Lecture Notes in Computer Science, Vol. 944, Springer, Berlin, 1995.
[37] A. Rubio, Theorem proving modulo associativity, in: Proc. of the Conference of the European Association for Computer Science Logic, Lecture Notes in Computer Science, Vol. 1092, Springer, Berlin, 1995.
[38] P. Viry, La réécriture concurrente, Thèse de Doctorat d'Université, Université de Nancy 1, October 1992.
[39] P. Viry, Rewriting: an effective model of concurrency, in: Proc. of PARLE'94, Lecture Notes in Computer Science, Vol. 817, Springer, Berlin, 1994.
[40] P. Viry, Rewriting modulo a rewrite system, Technical report TR-20/95, Università di Pisa, 1995, available from `http://www.di.unipi.it/ricerca/rapporti.html`.
[41] P. Viry, Input/Output for ELAN, in: J. Meseguer (Ed.), Proc. of 1st Internat. Workshop on Rewriting Logic and its Applications, Electronic Notes in Theoretical Computer Science, Vol. 4, Elsevier, Amsterdam, 1996, `http://www.elsevier.nl/locate/entcs`.
[42] P. Viry, Adventures in sequent calculus modulo equations, in: J. Meseguer (Ed.), Proc. of 2nd International Workshop on Rewriting Logic and its Applications, Electronic Notes in Theoretical Computer Science, Vol. 15, Elsevier, Amsterdam, 1998. `http://www.elsevier.nl/locate/entcs`.
[43] M. Vittek, ELAN: Un cadre logique pour le prototypage de langages de programmation avec contraintes, Thèse de Doctorat d'Université, Université Henri Poincaré - Nancy 1, October 1994.