Technical Report: TR-95-19, December 1, 1995

Elimination of Conditions

Viry, Patrick

Dipartimento di Informatica, Università di Pisa
corso Italia 40, 56125 Pisa, Italy

Abstract

We present a transformation from any conditional rewrite systems into non conditional ones and prove its correctness. The transformed systems are quite intuitive and well suited for parallel execution. We also show how termination and confluence of the original system are preserved in the transformed one.

ACM–CR Subject Classification:

Keyword and phrases:

UNIVERSITÀ DEGLI STUDI DI PISA
DIPARTIMENTO DI INFORMATICA

Technical Report TR-19/95

Elimination of Conditions

Patrick Viry*

Abstract. We prove the correctness of a transformation from conditional rewrite systems (CTRS) into non conditional ones. This particular transformation has the advantage of expliciting all the possible parallelism in the evaluation of conditions. The main result states that this transformation applies to any CTRS, and that derivations are preserved up to a mapping between terms. We also prove that termination and confluence of the original system are preserved in the transformed one under some natural assumptions.

December 1995

* Stay in Pisa supported by an HCM fellowship, EuroFOCS network.

Dipartimento di Informatica – Università di Pisa – Corso Italia, 40 – 56100 Pisa – Italy
Tel. +39 50 887001 – Fax. +39 50 887226 – Tlx. 590291 DIPISAI – Email. di.unipi.it

Elimination of Conditions

Patrick Viry

Dipartimento di Informatica, Università di Pisa
Email: viry@di.unipi.it

1 Introduction

Conditional rewrite rules appear in many specification frameworks, where they allow more natural specifications. They are also used in logico-functional languages to specify non equational deduction.

But conditional rules are more difficult to manage theoretically than unconditional ones, and also pose implementation problems, especially for parallel implementations where condition evaluation usually induces sequentiality ; this work was first motivated by the need of a formal proof for the implementation of concurrent rewriting in the conditional case (Kirchner and Viry, 1992). One would like to find for every conditional rewrite system (CTRS) an equivalent unconditional one, ie. inducing the same derivations.

This is not possible in general if we demand to keep the signature unchanged (Bergstra and Meyer, 1984), but possible if we allow a looser correspondence between the two systems, for instance with so-called hidden functions, since any Turing machine can be encoded in an unconditional TRS (Dauchet, 1989).

Constructive transformations have been sketched in various ways for some classes of systems in (Bergstra and Klop, 1986), (Giovannetti and Moiso 1987), (Dershowitz and Plaisted, 1988) and (Aida, Goguen and Meseguer, 1990), but were never given a formal proof, or only for some restrictive cases (a totally different approach can be found in (Hintermeier, 1994), proved correct for convergent systems). The transformation presented here follows the lines of these works, but the aim of this paper is to prove the correctness of the transformation, after having given it a formal definition, and to study its properties. The correctness proof is quite technical, because it is based on the properties of derivations which are objects with a complex structure.

We prove that the transformation applies to all CTRS's without any kind of restriction, and that it preserves derivations modulo a simple mapping between terms. We also prove sufficient conditions for the preservation of confluence and termination. Finally, we show why this transformation is especially well suited for parallelism.

A practical result is that an interpreter for unconditional rewriting can be used as well for conditional rules, without loss of parallelism. On a theoretical point of view, it is interesting to note that deduction is usually encoded by conditional rules (Martì-Oliet and Meseguer, 1993). The existence of this transformation and its absence of restrictive conditions shows that unconditional rules are sufficient to encode deduction, thus putting it on the same ground with other interpretations of rewrite steps, such as equational computations or transitions between states.

1.1 basic concepts and notations

Terms and positions. A term $t \in T_{\Sigma,X}$ (over a signature Σ and with variables in X) can be seen as a partial function $t : N^* \longrightarrow \Sigma$. If $t(p)$ is defined, then p is a position in t. $Pos(t)$ denotes the set of all positions in t, $FPos(t) = \{p \in Pos(t) | t(p) \in \Sigma\}$ the set of non variable positions and $VPos(t) = \{p \in Pos(t) | t(p) \in X\}$ the set of variable positions.

The empty position is denoted ε and concatenation $p.q$. If p and q are positions, $p \geq q$ means q is a substring of p, or p is below q. $p\#q$ means that the positions are incomparable, ie. neither $p \geq q$ nor $q \geq p$ (we reserve the word "disjoint" for redexes, in order to avoid confusion between "disjoint redexes" and "redexes at incomparable positions").

If $p \in Pos(u)$, $u|_p$ is the subterm at position p, $u[p \leftarrow v]$ the term u where the subterm at position p is equal to or replaced by v.

Rewrite steps. For each (non conditional) rule $\rho(\vec{x}) : l \longrightarrow r \in R$ (where $\vec{x} = Var(l) \cup Var(r)$ is the set of all variables appearing in l or r), for each context $C(.)_p$ (a term with a single "hole" at position p) and each sequence of terms $\vec{u}$, there is a rewrite step

$$\delta = C(\rho(\vec{u}))_p \ : \ s = C(l(\vec{u}/\vec{x}))_p \longrightarrow C(r(\vec{u}/\vec{x}))_p = t \ \text{ also written } \ \xrightarrow[p]{C(\rho(\vec{u}))}$$

where $\vec{u}/\vec{x}$ denotes the multiple substitution $u_1/x_1, ..., u_n/x_n$. $C(.)_p$ and $\vec{u}$ will be omitted when evident from the context. s and t are uniquely defined and are respectively called the *source* and *target* of δ. For instance, with a rule $\rho : f(x) \longrightarrow g(x)$, we have the following rewrite steps :

$$\begin{array}{rl} \rho(a) : & f(a) \longrightarrow g(a) \\ h(\rho(b)) : & h(f(b)) \longrightarrow h(g(b)) \end{array}$$

This definition allowing to put explicitly rewrite steps "in context" is similar to the *proof terms* of rewriting logic (Meseguer, 1992), except that it is restricted to sequential steps, and does not allow congruence over a composition. We will try to stick to the arrow notation whenever possible for readability.

In the δ step above, the couple (ρ, p) is called a *redex* of s. A term s and a redex in s uniquely determine a rewrite step, hence in the following we will often identify a redex and the corresponding rewrite step.

Derivations are possibly empty sequences of rewrite steps with the same source and target. An arbitrary sequence of terms is said *well-formed* when this condition is verified. Derivations are written with a double arrow $\twoheadrightarrow$ when using the arrow notation. Composition is denoted by ";" or by juxtaposition of arrows and the empty derivation is denoted ε. We write $\alpha \sim \beta$ if α and β are two derivations with the same source and target. As for single steps, a whole derivation can also be "put in context" according to the rule

$$\xrightarrow[p]{C(\delta_1;\ldots;\delta_n)} \ = \ \xrightarrow[p]{C(\delta_1)} \cdots \xrightarrow[p]{C(\delta_n)}$$

where $\delta_1, ..., \delta_n$ are single steps.

Disjoint redexes. If $\delta = (l \rightarrow r, p)$ is a redex of u and q a position of u, we write $q \geq \delta$ if there exist a position p' of a variable in l such that $q \geq p.p'$ (ie. q is in the "substitution part" of the redex).

Two rewrite steps (or the associated redexes) $\delta_1 = C_1(l_1 \rightarrow r_1)_{p_1}$ and $\delta_2 = C_2(l_2 \rightarrow r_2)_{p_2}$ are *disjoint* if either $p_1 \# p_2$, $p_1 \geq \delta_2$ or $p_2 \geq \delta_1$.

Residuals and Antecedents The notion of residuals allows to keep track of positions in a term after a reduction. If $\delta = C(\rho(\vec{s}))_q : u \longrightarrow v$ is a rewrite step applying a rule ρ at position q, then for any position p of u, the set of residuals of p after δ is defined as :

- $p \backslash \delta = p$ if $p \not\geq q$
- $p \backslash \delta = \{q.q''.p' \mid p = q.q'.p'\}$ if $p \geq \delta$, where q' and q'' are position of the same variable respectively in the left-hand side and the right-hand of ρ.
- $p \backslash \delta = \emptyset$ if $p \geq q$ and $p \not\geq \delta$

This definition extends to residuals of redexes or rewrite steps. In a derivation $\delta_1 ; \delta_2$, the set of residuals of δ_1 after δ_2 is defined as

$$\delta_1 \backslash \delta_2 \ = \xrightarrow[q_1]{\rho_1} \cdots \xrightarrow[q_n]{\rho_1} \quad \text{if} \quad \delta_1 = \xrightarrow[p_1]{\rho_1} \quad \text{and} \quad \delta_2 = \xrightarrow[p_2]{\rho_2}$$

where $\{q_1, ..., q_n\} = p_1 \backslash \delta_2$. Note that any two redexes of $\delta_1 \backslash \delta_2$ are at incomparable positions: if $i \neq j$, then $q_i \# q_j$.

Dually, if q is a residual of p after some derivation, we say that p is an antecedent of q. Note that if all rules are left-linear, each position has at most one antecedent, and exactly one if the variables of the right-hand side of a rule are included in its left-hand side.

The permutation lemma (see (Boudol, 1985) for left-linear systems, or (Viry, 1992) for the general case). If δ_1 and δ_2 are disjoint redexes in a same term, and δ_2 corresponds to a left-linear rule, then $\delta_1; \delta_2$ is a well-formed derivation and

$$\delta_1; \delta_2 \quad \sim \quad \delta_2; (\delta_1 \backslash \delta_2)$$

Conventions

R, R' and Σ, Σ' will denote respectively the original and transformed rewrite system and the signature it is based upon,

s, t denote terms over Σ, u, v, w terms over Σ',

p, q denote positions,

δ, γ denote single rewriting steps or redexes,

α, β denote derivations.

1.2 conditional systems

Different types of conditional rewrite systems (CTRS) are identified in the literature. Rules are all of the form $l \rightarrow r$ if $s_1 = t_1, \ldots, s_n = t_n$, but the equality sign in the conditional part is interpreted in different ways:

(1) In normal CTRS, t is a normal form, and $s = t$ denotes reachability $s \twoheadrightarrow t$.

(2) In join CTRS, $s = t$ denotes joinability $s \twoheadrightarrow \twoheadleftarrow t$.

(3) In semi-equational CTRS, $s = t$ denotes conversion $s \twoheadleftarrow\!\!\twoheadrightarrow t$.

If a semi-equational system is confluent, then it is equivalent to the corresponding join CTRS, since $s \twoheadleftarrow\!\!\twoheadrightarrow t \iff s \twoheadrightarrow \twoheadleftarrow t$. Join CTRS can be mapped to normal ones by use of an extra variable, or by adding an equality rule $x = x \longrightarrow \mathsf{true}$ if one does not want to introduce variables that do not appear in the left-hand side of the rule. Using again this equality rule, any condition $s \twoheadrightarrow t$ of a normal CTRS can be written in the form $(s = t) \twoheadrightarrow \mathsf{true}$, giving an equivalent system in the following sense: for two terms u and v not containing the new symbol "=", $s \longrightarrow t$ in the original system iff. $s \longrightarrow t$ in the transformed system.

We will thus consider here normal CTRS with conditions of the form $s \twoheadrightarrow \mathsf{true}$, hence handling all the following cases: normal CTRS, join CTRS and confluent semi-equational CTRS. This includes most practical examples of CTRS, one notable exception being conditional rewriting logic (Meseguer, 1992) in all its generality, since the rules considered there are those of normal CTRS, but without the requirement that t be a normal form. However, such systems may also be taken into account using a trick to forbid evaluation inside t while evaluating a condition.

The rewrite systems considered here have the two usual restrictions: the left-hand side of a rule is not reduced to a variable, and all variables appearing in a right-hand side also appear in the corresponding left-hand side.

Conditional rewrite steps and derivations. A conditional rewrite step takes as parameters the derivations evaluating the conditions. For each conditional rule $\rho(\vec{x}) : l \longrightarrow r$ if $c_1, ..., c_n$, for each context $C(.)_p$ and each sequence of terms $\vec{u}$, if there are (conditional) derivations $\alpha_1 : c_1 \twoheadrightarrow \mathsf{true}$, ..., $\alpha_n : c_n \twoheadrightarrow \mathsf{true}$, then there is a conditional rewrite step

$$\delta = C(\rho(\vec{u})[\alpha_1, ..., \alpha_n])_p \;:\; s = C(l(\vec{u}/\vec{x}))_p \longrightarrow C(r(\vec{u}/\vec{x}))_p = t$$

or $s \xrightarrow[p]{C(\rho[\alpha_1, \ldots, \alpha_n])} t$. Conditional derivations are built from conditional steps as before.

2 The transformation

2.1 an example

Consider the CTRS over the signature Σ

$$R = \begin{cases} \rho_1 : f(g(x)) \longrightarrow p(x) \ \mathtt{if}\ c(x) \twoheadrightarrow \mathsf{true} \\ \rho_2 : f(h(x)) \longrightarrow q(x) \ \mathtt{if}\ d(x) \twoheadrightarrow \mathsf{true} \\ \rho_3 : \quad c(a) \longrightarrow \mathsf{true} \end{cases}$$

Construct a new signature Σ' by adding to each symbol f a new position for each conditional rule where f is at the top of the left-hand side. Here f will take 3 arguments, the last two corresponding to the rules ρ_1 and ρ_2. We will write $f(x \mid y_1, y_2)$ to distinguish easily between "original positions" and "conditional positions", but the vertical bar $\mid$ is nothing else than a comma as in $f(x, y_1, y_2)$.

Construct a new rewrite system R' with two rules ρ'_i and ρ''_i for each conditional rule in R. The first one will start the evaluation of a condition when a match is detected, the second will actually apply the rule if the condition evaluates to true.

The conditional positions will contain either $\perp$, a special symbol indicating that no evaluation of a condition has started, or a couple $[c, (s_1, ..., s_n)]$ where c is the condition to be evaluated and $(s_1, ..., s_n)$ are the substitutions of the variables at the time a match was detected. Carrying the substitutions is necessary, as nothing prevents rewrites under the original positions during the evaluation of a condition, making it impossible to find the right substitution to use when actually applying the rule.

Continuing the example, the transformed system over Σ' is

$$R' = \begin{cases} \rho'_1 : \quad f(g(x) \mid \perp, z) \longrightarrow f(g(x) \mid [c(x), (x)], z) \\ \rho''_1 : f(x \mid [\mathsf{true}, (y)], z) \longrightarrow p(y) \\ \rho'_2 : \quad f(h(x) \mid z, \perp) \longrightarrow f(h(x) \mid z, [d(x), (x)]) \\ \rho''_2 : f(x \mid z, [\mathsf{true}, (y)]) \longrightarrow q(y) \\ \rho_3 : \quad c(a) \longrightarrow \mathsf{true} \end{cases}$$

The conditional derivation $f(g(a)) \xrightarrow{\rho_1[\rho_3]} p(a)$ is simulated by the following unconditional one, starting from the $\perp$-term $f(g(a) \mid \perp, \perp)$ representing $f(g(a))$:

$$\begin{aligned} f(g(a) \mid \perp, \perp) &\xrightarrow{\rho'_1} f(g(a) \mid [c(a), (a)], \perp) \\ &\xrightarrow{f(\ldots\mid\ldots[\rho_3,\ldots]\ldots)} f(g(a) \mid [\mathsf{true}, (a)], \perp) \\ &\xrightarrow{\rho''_1} p(a) \end{aligned}$$

The three steps needed to apply a conditional rule are exhibited here : matching, evaluation of the condition, and application. Note that :

- A ρ' rule only applies when a $\perp$ is present in the corresponding position. If there were a variable instead, the same ρ' rule could be applied again and again at the same position, causing non termination problems, but helping preserving confluence (see the discussion in section 4).
- The substitution used in the right-hand side of a ρ'' rule is the one previously recorded in the corresponding conditional position.
- The left-hand side of a ρ' rule is used for matching and contains as non-variables subterms the subterms of the left-hand side of the original rule (eg. $f(g(x) \mid ...)$). On the contrary, the left-hand side of a ρ'' rule contains only variables at the non-conditional positions (eg. $f(x \mid ...)$). It may be the case for instance that the subterms at non-conditional positions are rewritten between the application of ρ' and that of ρ''.

- Parallelism: at the same position, different conditions can be evaluated concurrently, together with the original term. When some condition becomes true, the corresponding ρ'' rule can be applied. If no condition ever evaluates to true, no special action has to be taken, the rewriting of the original term will simply continue.
  It is of course possible to follow a strategy that will try all conditions in sequence, "removing" any condition that has been evaluated to a normal form different from true. The trade-off is between memory usage and amount of parallelism.
- Extra variables in the conditional part become extra variables in the right-hand side.

When a rule ρ of R has more than one condition, there should be as many conditional positions as there are conditions, and ρ'' should check that they all have evaluated to true. However, for trying to keep notations concise, we will only formalize the case when each rule has exactly one condition. The extension to the general case is straightforward.

Call f-rule a rule with f as top symbol of its left-hand side. With this notation all rules of R can be labelled as $\rho_{f,i}$, denoting the i-th f-rule (remember that no rule has a single variable as its left-hand side).

Correspondence between original and extended terms

The function $u \mapsto \overline{u} : \Sigma' \longrightarrow \Sigma$ discards all conditional positions in u:

$$\overline{f(t_1, ..., t_n \mid s_1, ..., s_k)} = f(\overline{t_1}, ..., \overline{t_n})$$

In the other direction, we have to choose what to put in the new conditional positions. The function $s \mapsto s^{\perp} : \Sigma \longrightarrow \Sigma'$ fills these positions with $\perp$, and $s \mapsto s^X : \Sigma \longrightarrow \Sigma'$ fills them with new variables:

$$\begin{aligned} f(t_1, ..., t_n)^{\perp} &= f(t_1^{\perp}, ..., t_n^{\perp} \mid \perp, ..., \perp) \\ f(t_1, ..., t_n)^{X} &= f(t_1^{X_1}, ..., t_n^{X_n} \mid y_1, ..., y_k) \end{aligned}$$

where $X = \{y_1, ..., y_k\} \cup X_1 \cup \cdots \cup X_n$ is any set of variables not appearing in s.

A term $u \in T_{\Sigma',X}$ is called a $\perp$-term if all its conditional positions contain the symbol $\perp$. This is equivalent to say that there exists a term $s \in T_{\Sigma,X}$ such that $u = s^{\perp}$.

2.2 definition

First of all, we want to consider only left-linear rules in the correctness proof, except for the lone equality rule $x = x \longrightarrow \mathsf{true}$. This restriction can be handled by replacing any non left-linear rule in the original system by an equivalent left-linear one expliciting the equality checks in the conditional part. For instance, $f(x, x) \longrightarrow g(x)$ becomes $f(x_1, x_2) \longrightarrow g(x_1)$ if $x_1 = x_2 \longrightarrow \mathsf{true}$. This assumption is fundamental for the correctness proof, as it implies the following properties :

- the permutation lemma applies to any sequence of disjoint steps $\xrightarrow[p_1]{\rho_1} \xrightarrow[p_2]{\rho_2}$ provided that ρ_2 is not the identity rule or that $p_1 \not\geq p_2$.
- in a step $u \xrightarrow[p]{\rho} v$, if ρ is not the identity rule, then any position of v not in the "redex part" has exactly one antecedent.

Note that this trick is used only for the correctness proof, and not relevant when actually performing the transformation. However, it may also be convenient to do it systematically, in order to isolate the treatment of non linearity in a rewrite interpreter.

Our starting point is thus a conditional rewrite system R over Σ consisting of rules of the form:

$$\rho_{f,i} \;:\; l \longrightarrow r \text{ if } c \longrightarrow \mathsf{true}$$

where $l = f(l_1, ..., l_n)$ is a non-variable linear term, and possibly also of the non left-linear identity rule.

The transformed system R' over Σ' consists of the rule for equality $\rho^= : x = x \longrightarrow \mathsf{true}$ (which will be the only non left-linear rule) and the following two rules for each $\rho_{f,i} : l \longrightarrow r \in R$, where $l = f(l_1, ..., l_n)$ and m is the number of f-rules :

$$\begin{array}{rcrcl} \rho'_{f,i} & : & f(\vec{t} \mid y_1, ..., \bot_i, ..., y_m) & \longrightarrow & f(\vec{t} \mid y_1, ..., [c(\vec{x}), (\vec{x})]_i, ..., y_m) \\ \rho''_{f,i} & : & f(\vec{z} \mid y_1, ..., [\mathsf{true}, (\vec{x})]_i, ..., y_m) & \longrightarrow & r^\bot(\vec{x}) \end{array}$$

where $\vec{x}$ is the set of variables appearing in $\rho_{f,i}$, $\vec{t} = l^X|_1, ..., l^X|_n$ is the vector of the strict subterms of l where fresh variables have been put in all conditional positions, $\vec{z} = z_1, ..., z_n$ are fresh variables, and $r^\bot$ is the right-hand side of the original rule where all conditional positions have been filled with $\bot$.

During the correctness proof, we will want to use ρ'' steps for "collecting" parts of the derivation evaluating a condition, so we introduce the notation

$$\xrightarrow{\rho''_{f,i}[\alpha]} \; = \; \xrightarrow{f(...|...,[\alpha,...]_i,...)}\!\!\!\!\!\!\twoheadrightarrow \xrightarrow{\rho''_{f,i}}$$

In order for this to be a well-formed derivation, α must end in true.

When α is "complete", namely $\alpha : c \twoheadrightarrow \mathsf{true}$ where c is the condition to be evaluated in order to apply $\rho_{f,i}$, we introduce a new symbol ρ^* for "packing" the derivation $\rho''_{f,i}[\alpha]$ and the corresponding initial $\rho'_{f,i}$ step :

$$\xrightarrow{\rho^*_{f,i}[\alpha]} \; = \; \xrightarrow{\rho'_{f,i}} \xrightarrow{\rho''_{f,i}[\alpha]} \; = \; \xrightarrow{\rho'_{f,i}} \xrightarrow{f(...|...,[\alpha,...]_i,...)}\!\!\!\!\!\!\twoheadrightarrow \xrightarrow{\rho''_{f,i}}$$

The $\rho^*_{f,i}$ symbol corresponds to the application of the conditional rule

$$\rho^*_{f,i} : l^* \longrightarrow r^\bot \;\text{ if }\; c^\bot \longrightarrow \mathsf{true}$$

These ρ^* rules will never actually appear in the transformed system R'. This extended system is denoted $R^* = R' \cup \{\rho^*_{f,i} | \rho_{f,i} \in R\}$.

2.3 correctness

Our main theorem states that :

> **Theorem 1.** *Let R be a CTRS and R' the corresponding non conditional system as defined above :*
> *(1) For each derivation $\alpha : s \longrightarrow t$ in R, there is a derivation $\alpha' : s^\bot \longrightarrow t^\bot$ in R'.*
> *(2) Reciprocally for each derivation $\alpha : u \twoheadrightarrow v$ in R' starting with a $\bot$-term, there is a derivation $\overline{\alpha} : \overline{u} \twoheadrightarrow \overline{v}$ in R.*

Proof (1) As in the example above, replace every conditional step $s \xrightarrow{\rho_{f,i}[\alpha]} t$ with

$$s^\bot \xrightarrow{\rho'_{f,i}} \xrightarrow{f(...|...,[\alpha,...]_i,...))}\!\!\!\!\!\!\twoheadrightarrow \xrightarrow{\rho''_{f,i}} t^\bot$$

Proof (2) The idea is to identify subsequences $\xrightarrow{\rho'_{f,i}} \cdots \xrightarrow{\rho''_{f,i}}$ as above in α, and show that there is a corresponding $\rho_{f,i}$ step in R.

Firstly, we will identify these subsequences and "pack" them into ρ^* steps, obtaining an equivalent derivation α^* not containing ρ'' steps anymore. Then, we will show how to find a derivation $\overline{\alpha}$ in R corresponding to α^*.

The two following sections describe these two steps.

2.4 removing ρ'' steps

Proposition 2. *Consider a R' derivation*

$$\alpha \quad = \quad u \xrightarrow{\alpha'}\!\!\!\rightarrow v \xrightarrow[p]{\rho''_{f,i}} \xrightarrow{\alpha''}\!\!\!\rightarrow$$

where u is a $\bot$-term, then there is a step in α' applying $\rho'_{f,i}$ at the position antecedent of p.

This is true because v must contain a [...] symbol at the i-th conditional position ; if u is a $\bot$-term, it does not contain any, by definition, and since the rewrite rules do not contain extra variables in their right-hand sides, the only way to introduce a [...] is by the application of a ρ' rule.

Given a derivation $\alpha : u \twoheadrightarrow v$ in R' where u is a $\bot$-term, define $\alpha_0 = \alpha$ and construct a sequence $\alpha_0 \mapsto \alpha_1 \mapsto \cdots$, where α_{i+1} is obtained from α_i as follows. If α_i contains a ρ'' step, decompose α_i as :

$$\alpha_i = \quad \xrightarrow{\beta_1}\!\!\!\rightarrow \xrightarrow[p_1]{\rho} \overbrace{\xrightarrow[q_1]{\gamma_1} \cdots \xrightarrow[q_n]{\gamma_n}}^{\beta_2} \xrightarrow[p_2]{\rho''_{f,i}[\gamma]} \xrightarrow{\beta_3}\!\!\!\rightarrow$$

with

- $q_i \not\geq p_2 \backslash \gamma_n ... \gamma_{i+1}$ for all $1 \leq i \leq n$ (no step of β_2 is below an antecedent of p_2)
- $p_1 \geq p_2 \backslash \beta_2$ (p_1 is below the antecedent of p_2)

By proposition 2, such a decomposition is always possible.

Using the permutation lemma, we can "push forward" the ρ step until right next to the ρ'' step. We have $q_1 \not\geq p_1$, ie. either $p_1 \# q_1$ or $p_1 \geq q_1$. The permutation lemma always applies in the former case. In the latter case, the rule in γ_1 cannot be $\rho^=$ (its right-hand side is `true` and no residual of it could be ever rewritten by a ρ'' rule), hence it is a left-linear rule and the permutation lemma applies :

$$\xrightarrow[p_1]{\rho} \xrightarrow[q_1]{\gamma_1} \cdots \xrightarrow[q_n]{\gamma_n} \xrightarrow[p_2]{\rho''_{f,i}[\gamma]} \quad \mapsto \quad \xrightarrow[q_1]{\gamma_1} \xrightarrow[p'_1]{\rho}\!\!\!\rightarrow \cdots \xrightarrow[p'_n]{\rho}\!\!\!\rightarrow \xrightarrow[q_2]{\gamma_2} \cdots \xrightarrow[q_n]{\gamma_n} \xrightarrow[p_2]{\rho''_{f,i}[\gamma]}$$

where $\{p'_1, ..., p'_n\} = p_1 \backslash \gamma_1$. Exactly one of the ρ steps at $p_1 \backslash \gamma_1$ is under an antecedent of p_2, we can again push it to the right in a similar manner. Eventually, one of the residuals of the initial ρ step will be next to the ρ'' step :

$$\alpha'_i = \quad \xrightarrow{\beta_1}\!\!\!\rightarrow \xrightarrow{\beta'_2}\!\!\!\rightarrow \left(\xrightarrow[p]{\rho} \xrightarrow[p_2]{\rho''_{f,i}[\gamma]} \right) \xrightarrow{\beta_3}\!\!\!\rightarrow$$

with $p \geq p_2$. We can now transform this sequence by "pushing forward" the ρ step, using the permutation lemma, in order to get α_{i+1} :

$$\alpha_{i+1} = \quad \xrightarrow{\beta_1}\!\!\!\rightarrow \xrightarrow{\beta'_2}\!\!\!\rightarrow \left(\xrightarrow[p_2]{\rho''_{f,i}[\gamma]} \xrightarrow{\beta'_3}\!\!\!\rightarrow \right) \xrightarrow{\beta_3}\!\!\!\rightarrow$$

according to the following rules :

1. Non corresponding ρ' and ρ'' next to each other, remove the useless ρ' step :

$$\xrightarrow[p_2]{\rho'_{g,j}} \xrightarrow[p_2]{\rho''_{f,i}[\gamma]} \quad \mapsto \quad \xrightarrow[p_2]{\rho''_{f,i}[\gamma]} \qquad \text{if } (f,i) \neq (g,j)$$

2. Corresponding ρ' and ρ'' next to each other, "pack" these two steps into a unique ρ^* step :

$$\xrightarrow[p_2]{\rho'_{f,i}} \xrightarrow[p_2]{\rho''_{f,i}[\gamma]} \quad \mapsto \quad \xrightarrow[p_2]{\rho^*_{f,i}[\gamma]}$$

3. The first step rewrites a non conditional position, discard it :

$$\xrightarrow[p_2]{f(\ldots,C(\rho),\ldots|\ldots)} \xrightarrow[p_2]{\rho''_{f,i}[\gamma]} \quad\mapsto\quad \xrightarrow[p_2]{\rho''_{f,i}[\gamma]}$$

4. The first step rewrites the condition used by ρ'', remove it and keep track of it in the argument of ρ'' :

$$\xrightarrow[p_2]{f(\ldots|\ldots,[C(\rho),\ldots]_i,\ldots)} \xrightarrow[p_2]{\rho''_{f,i}[\gamma]} \quad\mapsto\quad \xrightarrow[p_2]{\rho''_{f,i}[C(\rho);\gamma]}$$

5. The first step rewrites the k-th substitution variable used by ρ'', move it after the ρ'' step :

$$\xrightarrow[p_2]{f(\ldots|\ldots,[\ldots,(\ldots,C(\rho)_k,\ldots)]_i,\ldots)} \xrightarrow[p_2]{\rho''_{f,i}[\gamma]} \quad\mapsto\quad \xrightarrow[p_2]{\rho''_{f,i}[\gamma]} \xrightarrow[p_2]{r^{\perp}(C(\rho)/x_k)}\!\!\!\!\rightarrow$$

$r^{\perp}(C(\rho)/x_k)$ denotes the term $r^{\perp}$ where x_k has been substituted by $C(\rho)$; since x_k may appear more than once in $r^{\perp}$, this is not necessary a single rewrite step, but a set of disjoint steps that can be performed in any order.

6. The first step rewrites an unused condition or substitution, remove it :

$$\xrightarrow[p_2]{f(\ldots|\ldots,C(\rho)_j,\ldots)} \xrightarrow[p_2]{\rho''_{f,i}[\gamma]} \quad\mapsto\quad \xrightarrow[p_2]{\rho''_{f,i}[\gamma]} \qquad \text{if } i \neq j$$

Correctness of this transformation is stated by :

Lemma 3. *(a) If α_i is well-formed, then so is α_{i+1} and $\alpha_{i+1} \sim \alpha_i$ (α_{i+1} and α_i have the same source and target).*
(b) The transformation terminates : there is no infinite sequence $\alpha_0 \mapsto \alpha_1 \mapsto \cdots$.
(c) Call α_∞ the last term of a sequence. α_∞ does not contain any ρ'' step (ie. only ρ', ρ^* and $\rho^=$).

Proof (a) $[\alpha_{i+1} \sim \alpha_i]$
Rules 1 to 6 are applications of the permutation lemma.

Proof (b) [the transformation terminates]
In a derivation α, for each step $\delta = (\rho'', p)$, define $M_\alpha(\delta)$ as the number of steps rewriting antecedents of p or their subterms. Now define the sequence $M(\alpha) = M_0(\alpha) \cdots M_k(\alpha)$, where $M_i(\alpha)$ is the sum of all $M_\alpha(\delta)$ such that δ applies a rule at depth i.
We have $M(\alpha_i) = M(\alpha'_i)$ and $M(\alpha'_i) > M(\alpha_{i+1})$, where $>$ is the lexicographic ordering on integers, which is well-founded.

Proof (c) [α_∞ does not contain any ρ'' step]
By proposition 2, one of the antecedents of p_2 must be a ρ' step. Either no subterm of an antecedent of p_2 is rewritten in β_2, in which case rule (1) or (2) applies, either there exists such an antecedent, in which case one of the rules (3), (4), (5) or (6) applies.

2.5 back to the original system

In the previous section, we have shown how to obtain from α a derivation α^* not containing any ρ'' step. Define recursively the transformation $\alpha^* \mapsto \overline{\alpha}$ as :

- If $\alpha^* = \varepsilon : u \twoheadrightarrow u$, then $\overline{\alpha} = \varepsilon : \overline{u} \twoheadrightarrow \overline{u}$.
- If $\alpha^* = u \xrightarrow{\alpha'}\!\!\!\!\rightarrow v \xrightarrow[p]{C(\rho)} w$:
  - if $\rho = \rho'$, then $\overline{\alpha} = \overline{\alpha'} : \overline{u} \twoheadrightarrow \overline{w}$
  - if $\rho = \rho^=$, then $\overline{\alpha} = \overline{\alpha'} : \overline{u} \twoheadrightarrow \overline{w}$
  - if $\rho = \rho^*_{f,i}[\beta]$, then $\overline{\alpha} = \overline{\alpha'}; C(\rho_{f,i}[\overline{\beta}])_p : \overline{u} \twoheadrightarrow \overline{w}$

Lemma 4. *If* $\alpha^* : u \twoheadrightarrow v$*, then* $\overline{\alpha} : \overline{u} \twoheadrightarrow \overline{v}$.

Proof. Check the source and target of α^* and $\overline{\alpha}$ for each case in the definition of $\alpha^* \mapsto \overline{\alpha}$. By induction over the nesting of conditions: suppose that it is true for all derivations β evaluating conditions in α^*, then

- Case (1): trivial.
- Case (2a):

$$\begin{array}{rrcl}
 & \alpha^* & = & u \xrightarrow{\alpha'}\!\!\!\!\!\rightarrow C(f(\vec{u} \mid ..., \perp, ...))_P \xrightarrow{\rho'} C(f(\vec{u} \mid ..., [...], ...))_p \\
\text{Hence} & \overline{\alpha} & = & \overline{u} \xrightarrow{\overline{\alpha'}}\!\!\!\!\!\rightarrow C(f(\vec{u}))_p = C(f(\vec{u}))_p
\end{array}$$

- Case (2b). The symbol $=$ is introduced only by ρ' rules and can only appear under a conditional position, so the only possible typing for α^* is:

$$\begin{array}{rrcl}
 & \alpha^* & = & u \xrightarrow{\alpha'}\!\!\!\!\!\rightarrow C(f(\vec{u} \mid ..., [s = s, (...)], ...))_p \\
 & & & \xrightarrow{\rho^=} C(f(\vec{u} \mid ..., [\mathsf{true}, (...)], ...))_p \\
\text{Hence} & \overline{\alpha} & = & \overline{u} \xrightarrow{\overline{\alpha'}}\!\!\!\!\!\rightarrow C(f(\vec{u}))_p = C(f(\vec{u}))_p
\end{array}$$

- Case (2c). If the original rewrite rule corresponding to ρ^* is $\rho_{f,i}(\vec{x}) : l \longrightarrow r$ if $c \longrightarrow \mathsf{true}$, then $\overline{\beta} : c^{\perp} \twoheadrightarrow \mathsf{true}$ by induction hypothesis and

$$\begin{array}{rl}
 & \alpha^* = u \xrightarrow{\alpha'}\!\!\!\!\!\rightarrow C(l^X(\vec{u}))_p \xrightarrow{\rho^*_{f,i}[\beta]} C(r^X(\vec{u}))_p \\
\text{Hence} & \overline{\alpha} = \overline{u} \xrightarrow{\overline{\alpha'}}\!\!\!\!\!\rightarrow C(l(\vec{u}))_p \xrightarrow{\rho_{f,i}[\overline{\beta}]} C(r(\vec{u}))_p
\end{array}$$

3 Preservation of termination

Even if R is terminating, R' may not be. The reason is that even if a CTRS is terminating, its rewriting relation may not be decidable. One has to evaluate conditions before deciding whether a rule applies, and it may be the case that the rewrite relation itself is terminating, but the process of evaluating conditions is not, as in the example

$$R = \{a \longrightarrow \mathsf{true} \text{ if } a \longrightarrow \mathsf{true}\}$$

The rewrite relation is empty, but in order to see if a can be rewritten, one has to evaluate a... The right notion guaranteeing the termination of this evaluation process is *decreasingness*:

Definition (Dershowitz and Okada, 1990) $>$ is a *decreasing* order for R if it is a well-founded conservative extension of the rewrite relation $\longrightarrow_R$, contains the proper subterm relation and satisfies $\sigma(l) > \sigma(c)$ for any rule $\rho : l \longrightarrow r$ if $c \longrightarrow \mathsf{true}$

The fundamental result is that the rewrite relation $\longrightarrow_R$ is decidable if R has a decreasing ordering. Preservation of termination is thus formulated as:

Theorem 5. *If R is decreasing, then R' is terminating.*

Proof. By hypothesis, there is a decreasing ordering $>$ for the original system R. Adding the new symbols $\perp$ and $[...]$ to the signature and extending $>$ with $\perp > [...]$ and $x = y > \mathsf{true}$, we still have a decreasing ordering.

Define the ordering $\gg$ on the transformed system R' as the least order stable by instanciation and context such that

- $l^* \gg r^{\perp}$ for each rule $l \longrightarrow r$ if $c \longrightarrow \mathsf{true}$ of R
- $l^* \gg c^{\perp}$ for each rule $l \longrightarrow r$ if $c \longrightarrow \mathsf{true}$ of R
- $\perp \gg [...]$

- $... = ... \gg$ true

Then $\gg$ is well-founded : $u \gg v$ implies $\overline{u} > \overline{v}$ and $>$ is well-founded. However, the standard proof techniques do not work, since $\gg$ is not a conservative extension of the rewrite relation of R', because of the ρ'' rules.

The proof goes by induction over the structure of the terms. Consider a $\perp$-term $u \in T_{\Sigma',X}$ and suppose that for every subterm v of u

(1) There is no infinite rewriting sequence starting from v
(2) If $v \xrightarrow{+}\!\!\!\!\rightarrow v'$ (derivation with at least one step), then $v \gg v'$.

This is true for all constants. Now consider a derivation

$$\alpha : \quad u = u_0 \xrightarrow[p_1]{\rho_1} u_1 \xrightarrow[p_2]{\rho_2} \cdots \xrightarrow[p_n]{\rho_n} u_n = u'$$

If $p_i \neq \varepsilon$ (ρ_i is not applied at the top), then $u_{i-1} \gg u_i$ by induction hypothesis. Otherwise, for $p_i = \varepsilon$, if $\rho_i = \rho^=$, then $u_{i-1} \gg u_i$, by definition of $\gg$. If $\rho_i = \rho'$, then $u_{i-1} = f(...|..., \perp, ...)$, $u_i = f(...|..., [...], ...)$, and $u_{i-1} \gg u_i$ since $\perp \gg [...]$ by definition.

If $\rho_i = \rho''$, then there must be a corresponding ρ' step before, by proposition 2, and there is no other ρ'' step between these two, otherwise the derivation would not be well-formed. We can partition α as :

$$u = u_0 \xrightarrow{\alpha'_1}\!\!\!\!\rightarrow \xrightarrow{\rho'_{f,i}} v_0 \xrightarrow{\alpha''_1}\!\!\!\!\rightarrow \xrightarrow{\rho''_{f,i}} u_1 \longrightarrow\!\!\!\!\rightarrow \cdots \longrightarrow\!\!\!\!\rightarrow u_{i-1} \xrightarrow{\alpha'_i}\!\!\!\!\rightarrow \xrightarrow{\rho'_{f,i}} v_{i-1} \xrightarrow{\alpha''_i}\!\!\!\!\rightarrow \xrightarrow{\rho''_{f,i}} u_n \xrightarrow{\alpha'_{i+1}}\!\!\!\!\rightarrow u'$$

where the α'_i, α''_i do not contain ρ'' steps at the top.

Note that in $v_{i-1} \xrightarrow{\alpha''_i}\!\!\!\!\rightarrow w_i \xrightarrow{\rho''_{f,i}} u_i$, we do not have $w_i \gg u_i$. But we do have $v_i \gg u_{i+1}$: v_i is of the form $l(\vec{v}'|...)$, where l is the left-hand side of a rule of R, and u_{i+1} of the form $r^{\perp}(\vec{v}'')$, where r is the corresponding right-hand side. $v' \gg v''$ by induction hypothesis, hence $v_i \gg u_{i+1}$ by definition of $\gg$. So the picture is

$$u = u_0 \gg v_0 \gg u_1 \gg \cdots \gg u_n \gg v_n \gg u'$$

Since $\gg$ is well founded, there can only be a finite number of ρ'' steps. Since all α_i are finite as well and verify $u_{i_k} \gg v_{i_k}$, α is also finite and verifies $u \gg u'$.

This concludes the proof of theorem 5.

4 Preservation of confluence

Confluence is not preserved in general. We can identify three reasons.

1. It may happen that a condition evaluation is being launched at position p with some substitution, but does not evaluate to true. However, through rewrites below p (in the "redex" part), the substitution may become different (not simply a reduct of the previous substitution), and in this context the condition would have evaluated to true. The following example illustrates such a case :

$$R = \begin{cases} \rho_1 : f(g(x)) \longrightarrow h(x) \text{ if } \ c(x) \longrightarrow \mathsf{true} \\ \rho_2 : c(a) \qquad\ \ \longrightarrow \mathsf{true} \\ \rho_3 : g(g(x)) \longrightarrow g(x) \end{cases}$$

   R is confluent, and the transformed system is

$$R' = \begin{cases} \rho'_1 : f(g(x) \mid \perp) \qquad \longrightarrow f(g(x) \mid [c(x), x]) \\ \rho'_2 : f(x \mid [\mathsf{true}, y])) \longrightarrow h(y) \\ \rho_2 : c(a) \qquad\qquad\quad \longrightarrow \mathsf{true} \\ \rho_3 : g(g(x)) \qquad\quad\ \longrightarrow g(x) \end{cases}$$

but R' is not confluent as shown by the following critical peak :

$$\begin{array}{rcl}
f(g(a) \mid \bot) \xleftarrow{\rho_3} & f(g(g(a)) \mid \bot) \xrightarrow{\rho_1'} & f(g(g(a)) \mid [c(g(a)), g(a)]) \\
f(g(a) \mid [c(a), a]) \xleftarrow{\rho_1'} & & \xrightarrow{\rho_3} f(g(a) \mid [c(g(a)), g(a)]) \\
f(g(a) \mid [\mathsf{true}, a]) \xleftarrow{\rho_2} & & \\
h(a) \xleftarrow{\rho_1''} & &
\end{array}$$

We can imagine two ways of addressing this problem:

(a) Use some kind of "bottom-up strategy", allowing to check the conditions in a term $f(t_1, \ldots, t_n \mid \bot, \ldots, \bot)$ only when all the t_i's are in normal form.
This strategy is commonly used in interpreters for conditional rewriting. Its drawback is that it restricts the possible parallelism.

(b) Have a way to know when the redex part has been rewritten, and in this case restart the evaluation of the conditions from scratch.
This is feasible in practice in a parallel implementation based on "bottom-up matching", like the one described in (Kirchner and Viry, 1992), since whenever this redex part is modified, the node corresponding to the top of the redex will eventually receive a "matching message".
Formalizing this approach would require to define a slightly different transformed system R', where the same condition could be evaluated many times, by replacing $\bot$ by a variable in the left-hand side of ρ' rules. The correctness proof of the transformation given here still applies, but proving the preservation of termination becomes tricky.

We will describe here the first approach.

2. Confluence is only preserved for terms originating from a $\bot$-term. When this is not the case, it is possible to have in the conditional parts conditions not making sense for the original system. An example is

$$R = \begin{cases} \rho_1 : f(x) \longrightarrow a \text{ if } x = a \\ \rho_2 : f(x) \longrightarrow b \text{ if } x = b \end{cases}$$

$$R' = \begin{cases} \rho_1' : f(x \mid \bot, y_2) \longrightarrow f(x \mid [x = a, (x)], y_2) \\ \rho_1'' : f(x \mid [true, (x)], y_2)) \longrightarrow a \\ \rho_2' : f(x \mid y_1, \bot) \longrightarrow f(x \mid y_1, [x = b, (x)]) \\ \rho_2'' : f(x \mid y_1, [true, (x)]) \longrightarrow b \end{cases}$$

R is confluent, but the following critical peak is not confluent in R' :

$$a \xleftarrow{\rho_1''} f(a \mid [true, (a)], [true, (a)]) \xrightarrow{\rho_2''} b$$

3. Although I have not been able to provide a counterexample, it seems that confluence is preserved only up to the mapping to the original system $u \mapsto \overline{u}$, ie. if $u \twoheadrightarrow v_1$ and $u \twoheadrightarrow v_2$, then there exist w_1 and w_2 such that $v_1 \twoheadrightarrow w_1$, $v_2 \twoheadrightarrow w_2$ and $\overline{w_1} = \overline{w_2}$.
However, since our aim of transforming a conditional system in a non conditional one is to implement the former one, this is all what we need.

Based on these three remarks, preservation of confluence is stated as follows.

Definition. A derivation α in R' is bottom-up if whenever

$$\xrightarrow{\alpha}\!\!\!\!\rightarrow \;=\; \xrightarrow{\alpha_1}\!\!\!\!\rightarrow f(\vec{u} \mid \ldots) \xrightarrow{\rho'} f(\vec{u} \mid \ldots) \xrightarrow{\alpha_2}\!\!\!\!\rightarrow$$

then α_2 does not rewrite any residual of the $\vec{u}$'s. This is true in particular if all the $\vec{u}$'s are in normal form.

Definition. R' is $\perp$-confluent if for any u being a reduct of a $\perp$-term, if there are bottom-up derivations $u \longrightarrow\!\!\!\!\rightarrow v_1$ and $u \longrightarrow\!\!\!\!\rightarrow v_2$, then there exist w_1 and w_2 such that $v_1 \longrightarrow\!\!\!\!\rightarrow w_1$ and $v_2 \longrightarrow\!\!\!\!\rightarrow w_2$.

Theorem 6. *If R is confluent, then R' is $\perp$-confluent.*

We will need the following lemma:

Lemma 7. *If $\alpha : u \longrightarrow\!\!\!\!\rightarrow v$ is a bottom-up derivation starting from a $\perp$-term u, then there exist derivations $\alpha_1 : u \longrightarrow\!\!\!\!\rightarrow u^\perp$ and $\alpha_2 : u^\perp \longrightarrow\!\!\!\!\rightarrow v$, where α_2 only contains ρ' steps.*

Proof of lemma 7

The idea is to "push away" to the right all the ρ' steps at a position p such that p has at one or more residuals in v (v would be a $\perp$-term if there were no such steps). We will see that this is always possible for a bottom-up derivation, but we have to take care that moving around ρ' steps preserves the property of being bottom-up.

Define a sequence of derivations $\alpha_i : u \xrightarrow{\alpha_i'}\!\!\!\!\rightarrow w_i \xrightarrow{\alpha_i''}\!\!\!\!\rightarrow v$ by $\alpha_0 = \alpha : u \longrightarrow\!\!\!\!\rightarrow w_0 = v$ and constructing α_{i+1} as follows. If α_i is a bottom-up derivation, and there exist a position q in w_i such that $w_i|_q = f(...|..., [...], ...)$, decompose α_i as

$$\alpha_i \;=\; \underbrace{u \xrightarrow{\beta_1}\!\!\!\!\rightarrow \; \xrightarrow[p]{\rho} \; \xrightarrow{\beta_2}\!\!\!\!\rightarrow}_{\alpha_i'} w_i \underbrace{\xrightarrow{\beta_3}\!\!\!\!\rightarrow}_{\alpha_i''} v$$

where the ρ step is the last step at or below an antecedent of q. With the same argument as for proposition 2, such a decomposition always exist. Using the permutation lemma, as in section 2.4, we can push this ρ step to the right until after w_i, and we get

$$\alpha_{i+1} = \underbrace{u \xrightarrow{\beta_1}\!\!\!\!\rightarrow \; \xrightarrow{\beta_2'}\!\!\!\!\rightarrow}_{\alpha_{i+1}'} w_{i+1} \underbrace{\xrightarrow[p]{\rho} \; \xrightarrow{\beta_3}\!\!\!\!\rightarrow}_{\alpha_{i+1}''} v$$

In α_i, call q' the antecedent of q before β_2. Then either $p = q'$ and ρ is a ρ' rule, or p is under a conditional position below q (but not under an original position), by the bottom-up hypothesis. We thus have $\overline{w_{i+1}} = \overline{w_i}$.

After having "pushed" the ρ step using the permutation lemma, some of its residuals may still be present in α_{i+1}'. No rewrite is present under the residuals of the original positions of p, by the bottom-up hypothesis, hence $\alpha_{i+1}'; \alpha_{i+1}''$ is still a bottom-up derivation, and we can iterate the construction.

The sequence $\alpha_0 \mapsto \alpha_1 \mapsto \cdots$ terminates, because the number of steps below the antecedents of all such positions q in w_i is greater by one than this number in w_{i+1}.

When the transformation terminates with w_n, then w_n is a $\perp$-term (by definition of the transformation), hence we have exhibited a derivation

$$\alpha_n : u \xrightarrow{\alpha_n'}\!\!\!\!\rightarrow w_n \xrightarrow{\alpha_n''}\!\!\!\!\rightarrow v$$

where w_n is a $\perp$-term, $\overline{w_n} = \overline{v}$ and α_n'' only contains ρ' steps.

Proof of theorem 6. Let u be a reduct of a $\perp$-term, ie. there exists $u^\perp$ such that $u^\perp \longrightarrow\!\!\!\!\rightarrow u$. Suppose that there are bottom-up derivations $u \longrightarrow\!\!\!\!\rightarrow v_1$ and $u \longrightarrow\!\!\!\!\rightarrow v_2$. Then we have the following diagram:

$$\begin{tikzcd}
 & v_1^\perp \arrow[r, dotted, ->>] \arrow[dr, dotted, ->>] \arrow[d, phantom, "(1)"] & v_1 \arrow[r, dotted, ->>] \arrow[d, phantom, "(3)"] & w_1 \\
u^\perp \arrow[r, ->>] \arrow[ur, dotted, ->>] \arrow[dr, dotted, ->>] & u \arrow[ur, ->>] \arrow[dr, ->>] \arrow[r, phantom, "(2)"] \arrow[d, phantom, "(1)"] & w^\perp \arrow[ur, dotted, ->>] \arrow[dr, dotted, ->>] \arrow[d, phantom, "(3)"] & \\
 & v_2^\perp \arrow[r, dotted, ->>] \arrow[ur, dotted, ->>] & v_2 \arrow[r, dotted, ->>] & w_2
\end{tikzcd}$$

with $\overline{w_1} = \overline{w_2}$.

(1) If $u^\perp \longrightarrow\!\!\!\!\!\rightarrow u \longrightarrow\!\!\!\!\!\rightarrow v_1$, then $u^\perp \longrightarrow\!\!\!\!\!\rightarrow v_1^\perp \longrightarrow\!\!\!\!\!\rightarrow v_1$:

By lemma 7.

(2) If $u^\perp \longrightarrow\!\!\!\!\!\rightarrow v_1^\perp$ and $u^\perp \longrightarrow\!\!\!\!\!\rightarrow v_2^\perp$, then there exists $w^\perp$ such that $v_1^\perp \longrightarrow\!\!\!\!\!\rightarrow w^\perp$ and $v_2^\perp \longrightarrow\!\!\!\!\!\rightarrow w^\perp$

By theorem 1, there exist derivations $\overline{u^\perp} \longrightarrow\!\!\!\!\!\rightarrow \overline{v_1^\perp}$ and $\overline{u^\perp} \longrightarrow\!\!\!\!\!\rightarrow \overline{v_1^\perp}$ in R. By hypothesis, R is confluent, hence there exists a w such that $\overline{v_1^\perp} \longrightarrow\!\!\!\!\!\rightarrow w$ and $\overline{v_1^\perp} \longrightarrow\!\!\!\!\!\rightarrow w$. Using again theorem 1, we have $v_1^\perp \longrightarrow\!\!\!\!\!\rightarrow w^\perp$ and $v_2^\perp \longrightarrow\!\!\!\!\!\rightarrow w^\perp$.

(3) If $v_1^\perp \xrightarrow{\alpha}\!\!\!\!\!\rightarrow v_1$ and $v_1^\perp \xrightarrow{\beta}\!\!\!\!\!\rightarrow w^\perp$, where α contains only ρ' steps, then there exists w_1 such that $v_1 \longrightarrow\!\!\!\!\!\rightarrow w_1$ and $w^\perp \longrightarrow\!\!\!\!\!\rightarrow w_1$:

Using the permutation lemma, we can commute all ρ' steps in α with all the steps in β applying left-linear rules. This applies also in the case of a $\rho^=$ step not above a ρ' step. The problematic case is that of a ρ' step at position p and a $\rho^=$ step at position $q \leq p$ (above p). The general permutation lemma (Viry, 1992) says that the following commutation is possible:

$$\xrightarrow[p_1]{\rho'} \xrightarrow[p_2]{\rho'} \xrightarrow[q]{\rho^=} \quad \Rightarrow \quad \xrightarrow[q]{\rho^=}$$

where p_1 and p_2 are "symmetric" positions (see (Viry, 1992)). In other words, we have to guarantee that two corresponding ρ' steps are present before being able to commute. This is of course not the case in general, but since the derivation β ends in a $\perp$-term, ie. all ρ' steps are necessary followed by ρ^* or $\rho^=$, we can add all the "missing" ρ' steps without changing the source and target of β, as in

$$\left.\begin{array}{ll} & f(\ldots|[g(\ldots|\perp) = g(\ldots|\perp)]) \\ \xrightarrow{\rho'} & f(\ldots|[g(\ldots|[\ldots]) = g(\ldots|\perp)]) \\ \xrightarrow{\rho^*} & h(\ldots) \end{array}\right\} \sim \left\{\begin{array}{ll} & f(\ldots|[g(\ldots|\perp) = g(\ldots|\perp)]) \\ \xrightarrow{\rho'} & f(\ldots|[g(\ldots|[\ldots]) = g(\ldots|\perp)]) \\ \xrightarrow{\rho'} & f(\ldots|[g(\ldots|[\ldots]) = g(\ldots|[\ldots])]) \\ \xrightarrow{\rho^*} & h(\ldots) \end{array}\right.$$

When first doing this transformation to β, we are assured to be able to commute in all cases.

Finally, we have $\overline{w_1} = \overline{w_2}$, since the derivations $w^\perp \longrightarrow\!\!\!\!\!\rightarrow w_1$ and $w^\perp \longrightarrow\!\!\!\!\!\rightarrow w_2$ contain only ρ' steps.

5 Conclusion

Given a conditional system R over a signature Σ, we have proved that there exists a non conditional system R' over Σ' equivalent to R in the sense that for any derivation $\alpha : s \longrightarrow t$ in R, there exists $\alpha' : s^\perp \longrightarrow t^\perp$ in R', and reciprocally for each derivation $\alpha : u \longrightarrow\!\!\!\!\!\rightarrow v$ in R' starting with a $\perp$-term, there exists an $\overline{\alpha} : \overline{u} \longrightarrow\!\!\!\!\!\rightarrow \overline{v}$ in R.

We also showed under which conditions termination and confluence of the original are preserved. Although these conditions may be seen restrictive, they are necessarily found in all CTRS interpreters, for instance one of them is necessary for the conditional rewriting relation to be decidable.

On a practical aspect, this transformation can be used to implement directly and quite efficiently *any* CTRS using a non conditional rewrite interpreter. It is especially efficient in the case of a parallel interpreter since different conditions can be evaluated concurrently.

On a theoretical aspect, note that deduction is often represented as application of conditional rules, as in (Martì-Oliet and Meseguer, 1993). This transformation shows that even non conditional systems are able to formalize deduction steps.

As a final remark, it may be interesting to investigate in some future work whether it is possible to derive properties of the original system by looking at the non conditional transformed system. This would allow to use directly all theoretical and practical tools developed for non conditional systems, like completion and termination orderings.

References

H. Aida, G. Goguen, and J. Meseguer (1990). Compiling concurrent rewriting onto the rewrite rule machine. In S. Kaplan and M. Okada, editors, *Proceedings 2nd International Workshop on Conditional and Typed Rewriting Systems, Montreal (Canada)*, volume 516 of *LNCS*, pages 320–332. Springer-Verlag.

J. A. Bergstra and J. W. Klop (1986). Conditional rewrite rules: Confluency and termination. *Journal of Computer and System Sciences*, 32(3):323–362.

J. A. Bergstra and J.-Ch. Meyer (1984). On specifying sets of integers. *Elektronishe Informations-verarbeitung und Kybernetics*, 20(10 & 11):531–541.

G. Boudol (1985). Computational semantics of term rewriting systems. In M. Nivat and J. C. Reynolds, editors, *Application of Algebra to Language definition and Compilation.* Prentice Hall. Also INRIA research report 192 (1983).

M. Dauchet (1989). Simulation of Turing machines by a left-linear rewrite rule. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *LNCS*, pages 109–120. Springer-Verlag.

N. Dershowitz and M. Okada (1990). A rationale for conditional equational programming. *Theoretical Computer Science*, 75:111–138.

N. Dershowitz and D. A. Plaisted (1988). Equational programming. In J. E. Hayes, D. Michie, and J. Richards, editors, *Machine Intelligence 11: The logic and acquisition of knowledge*, chapter 2, pages 21–56. Oxford Press, Oxford.

E. Giovannetti and C. Moiso (1987). Notes on the elimination of conditions. In J.-P. Jouannaud and S. Kaplan, editors, *Proceedings 1st International Workshop on Conditional Term Rewriting Systems, Orsay (France)*, volume 308 of *LNCS*. Springer-Verlag.

C. Hintermeier (1994). A transformation of canonical conditional trs's into equivalent canonical trs's. In *Proceedings of the 4th International Workshop on Conditional Rewriting Systems, Jerusalem (Israel).*

C. Kirchner and P. Viry (1992). Implementing parallel rewriting. In B. Fronhöfer and G. Wrightson, editors, *Parallelization in Inference Systems*, volume 590 of *Lecture Notes in Artificial Intelligence*, pages 123–138. Springer-Verlag.

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

N. Martì-Oliet and J. Meseguer (1993). Rewriting logic as a logical and semantical framework. Technical report, SRI International.

P. Viry (1992). *La réécriture concurrente.* Thèse de Doctorat d'Université, Université de Nancy 1.