Conditional Concurrent Rewriting

Iliès Alouini, Claude Kirchner

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Conditional Concurrent Rewriting

Iliès ALOUINI , Claude KIRCHNER

N ° 2777

Mars 1996

THÈME 2

ISSN 0249-6399

INRIA
LORRAINE

Conditional Concurrent Rewriting

Iliès ALOUINI* , Claude KIRCHNER†

Thème 2 — Génie logiciel
et calcul symbolique
Projet PROTHEO

Rapport de recherche n ° 2777 — Mars 1996 — 24 pages

Abstract: For implementing conditional concurrent rewriting on distributed memory machines, we have designed a transformation of conditional term rewriting systems to unconditional rewriting systems. The transformation provides a terminating rewriting system that simulates in a correct and complete way the application of the initial conditional rewrite system, which is assumed to be decreasing and confluent. Furthermore, this allows to define a model for the implementation of conditional concurrent rewriting using a fine-grained parallelism on terms and based on term hypergraph rewriting. Finally, we present first experimental results based on our implementation RECO of conditional concurrent rewriting and show that significant speed-ups can be obtained when computing normal forms of terms.

Key-words: Concurrent rewriting, Rewrite derivations, Conditional rewriting, MIMD architecture, Parallel algorithms.

(Résumé : tsvp)

* E-mail: Ilies.Alouini@loria.fr
† E-mail: Claude.Kirchner@loria.fr

La Réécriture Conditionnelle Concurente

Résumé : Pour implémenter la réécriture conditionnelle concurrente sur des machines à mémoires distribuées, nous avons conçu une transformation de systèmes de réécriture conditionnelles en des systèmes de réécriture non conditionnelles. La transformation donne un système de réécriture terminant qui simule d'une manière correcte et complète l'application du système de réécriture conditionnel initial en supposant ce dernier décroissant et confluent. De plus, ceci permet de définir un modèle pour l'implémentation de la réécriture conditionnelle concurrente utilisant un parallèlisme de grains fins sur les termes. Finalement, nous présentons les premiers résultats expérimentaux basés sur notre implémentation de la réécriture conditionnelle concurrente RECO et nous montrons que des facteurs d'accélération sont possibles lors du calcul de formes normales de termes.

Mots-clé : Réécriture concurrente, Dérivations de réécriture, Réécriture conditionnelle, Architecture MIMD, Algorithmes parallèles.

Contents

1 Introduction

The fundamental interest of using term rewriting as an executable specification tool was shown in the last two decades, through many theoretical works but also many implementations of the concept, like (without exhaustivity) OBJ, PLUSS, LPG, ASF+SDF. More recently the usefulness of the rewriting logic as a logical framework [MOM93] and the resulting implementation of the ELAN [KKV95] and MAUDE [Mes93] languages have shown the importance of having efficient implementations of conditional term rewriting, in particular in parallel.

Initiated in the mid eighties, the idea of using concurrent term rewriting as a model of computation [GKM87] has drawn lot of attention, either when designing specific hardware for implementing the concepts [GKL+86], or for the theoretical interest of conditional rewriting as a model of fine grain concurrency [Mes92, Vir94, DL95], or when implementing rewriting on distributed memory machines [KV92, Vir92]. This last work leads to the development in our research group of RECO, an implementation of concurrent rewriting on distributed memory machines, and the goal of this paper is to show how *conditional* concurrent term rewriting can be implemented on distributed memory machines (or a network of machines). This is achieved by providing a specially designed reduction of conditional concurrent rewriting to unconditional concurrent rewriting.

Implementing concurrent rewriting is not an easy task since one has to first design the right implementation model and then to design and implement the right environment, including in particular garbage collection. In our case we have shown that it is correct and complete to use the notion of jungle [HP88] in order to provide the basis for the implementation [KV92] and that in this context a concurrent GC can be designed and implemented [Alo95]. In this context it appears of course much more interesting to use the model and implementation of unconditional concurrent term rewriting (which we abbreviate by CcRw) in order to design and implement conditional concurrent term rewriting (CdCcRw). This has as least two advantages: the first is to allow a modular and thus easier to understand design and implementation of each part of the model, and the second is to reuse the existing implementation. Several transformations of conditional term rewriting system into unconditional term rewriting systems already existed as surveyed in [Hin94]. So our first idea was simply to reuse one of them in order to do the job. Unfortunately none of them is able to take into account the following two main specificities of the model. First, it is fundamental to preserve as much as possible the inherent concurrency present in the initial conditional system, and the existing transformations are either unnecessarily blocking some redexes for concurrent evaluation or fixing a reduction strategy. Second, specific properties of the implementation model could be used in order to ease the transformation. This is typically the case of the normalisation information which is already present in the unconditional implementation due to its role in the detection of termination of the concurrent reduction process.

Thus the results described in this paper concern the implementation model of concurrent decreasing conditional term rewriting systems in the class of confluent normal joined TRS [DO90]. We first present a transformation from CdCcRw into CcRw and the proof that this transformation provides a terminating rewrite system that simulates in a correct and complete way the application of the initial conditional rewrite rules, provided that the unconditional rewrite system is applied in a fair way. Second we show how the transformation is plugged-into the current implementation model and we give the first account of the results we have obtained for the parallel evaluation of conditional term rewriting using RECO.

Let us outline how the transformation we are proposing works. First, all the terms and subterms, both in the transformed rewrite system and in the term to be reduced, are once and for all decorated with two pieces of information: a normal form flag and a natural number for counting the number of conditional rules unsuccessfully applied to that term.

Second, in the spirit of other transformations [AGM90], the application of a conditional rewrite rule (for example $f(s_1, s_2) \to d$ if c), at some occurrence, is realised in the following way. We first made the condition and the right hand side of this rule, part of the term to be evaluated by using a new function symbol. For example if the term to be deduced is $g(f(t_1, t_2))$, we introduce $g(f_{new}(t_1, t_2, \sigma(c), \sigma(d)))$ where σ is the matching substitution from $f(s_1, s_2)$ to $f(t_1, t_2)$. Then this term is reduced using the rewrite system up to the state where the subterm representing the condition is evaluated to true or false. In which case either we step back to the (possibly reduced) initial term when the condition is false (e.g. $g(f_{new}(t'_1, t'_2, false, \sigma(d))) \to g(f(t'_1, t'_2))$), or we replace the redex by the stored right hand side (e.g. $g(f_{new}(t'_1, t'_2, true, \sigma(d))) \to g(\sigma(d))$). This is quite simple, but not terminating of course, in particular when the condition reduces to false! This is why we need the additional information stored in the term and to assume that the transformed rewrite rules are applied in a fair and ordered way.

The paper is organised as follows. After introducing the necessary preliminaries in section 2 we describe the transformation in detail in section 3. Then in section 4, we show how we simulate each rewrite derivation in the conditional system with a derivation in the transformed system and vice-versa. In section 5, we describe the way we have chosen to implement the transformation based on the existing concurrent rewriting model, and we also present first experimental results. Finally, we summarize our work in section 6.

2 Preliminaries

We assume the reader to be familiar with conditional term rewriting, and we refer to the surveys [DO90, DJ90] for the basic notions used in this paper..

We consider a set $\mathcal{F}$ of ranked function symbols where $\mathcal{F}_n$ is the subset of functions of arity n, a set $\mathcal{X}$ of variables and the set of first-order terms $\mathcal{T}(\mathcal{F}, \mathcal{X})$ built on $\mathcal{F}$ using variables in $\mathcal{X}$. The set of variables of a term t is denoted $\mathcal{V}ar(t)$. The set of ground terms (i.e. terms without variables) is denoted $\mathcal{T}(\mathcal{F})$. We use the following conventions for notation: u, v, w, s, t will denote terms on a given $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and p, q, r denote positions which are sequences of natural numbers. By $\mathcal{P}os(t)$ we denote the set of positions in t. We use the usual partial order on terms positions $p \geq q$ iff $p = q.r$.

Let $\mathcal{R}$ be a conditional rewrite system. Following [DO90], the rules of $\mathcal{R}$ are assumed to be of the form $(u \to v$ if $s \bowtie s')$ where $u, v, s, s' \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $\mathcal{V}ar(v) \cup \mathcal{V}ar(s) \cup \mathcal{V}ar(s') \subseteq \mathcal{V}ar(u)$ and $\bowtie$ is one of the relations:

1- $\overset{*}{\longleftrightarrow}_R$, for *natural* conditional rewriting,
2- $\downarrow^!_R$, for *normal-joined* conditional rewriting,
3- $\overset{*}{\longrightarrow}_R$, for *normal* conditional rewriting.

We restrict ourselves in this work to normal-joined conditional rewriting. A conditional rewrite system $\mathcal{R}$ is called *decreasing*, if there is a well-founded extension $>$ of the proper subterm ordering such that $>$ contains the reduction ordering $>_R$ (the irreflexive transitive closure of the rewrite relation $\to_R$) , and satisfies for each rule $u \to v$ if $\downarrow^!_R (s, s')$ in $\mathcal{R}$ and for all substitution σ: $\sigma(u) > \sigma(s), \sigma(s')$. We denote $\mathcal{R} = \bigcup_{f \in \mathcal{F}} \mathcal{R}_f$, where $\mathcal{R}_f$ is the set of rules in $\mathcal{R}$ having f as the top symbol of the left-hand side ($\mathcal{R}_f = \emptyset$ if no rule of $\mathcal{R}$ has f as a top symbol of a left-hand side). i_f denotes the number of rules in $\mathcal{R}_f$. $i_{\mathcal{R}}$ denotes the number of symbols f in $\mathcal{F}$ having at least one rule in $\mathcal{R}_f$. $i_{\mathcal{F}}$ is the total number of symbols in $\mathcal{F}$. To ease the description of the transformation, we use labels denoted $\ell^j_{i,k}$ to name and distinguish the rules. Thus the set of rules $\mathcal{R}_f$ is assumed to be of the

form: $\mathcal{R}_f = \left\{ \begin{array}{lllll} \ell^j_{1,k}: & f(t^1_1,\ldots,t^1_n) & \rightarrow & r_1 & \text{if } c_1 \\ & \vdots & & & \\ \ell^j_{i_f,k}: & f(t^{i_f}_1,\ldots,t^{i_f}_n) & \rightarrow & r_{i_f} & \text{if } c_{i_f} \end{array} \right.$

where $i \in \{1,\ldots,i_f\}$, j a natural number strictly greater than 1, $k \in \{1,\ldots,i_{\mathcal{R}}\}$. i is used for enumerating the rules of $\mathcal{R}_f$. Each symbol f in $\mathcal{F}$ has a unique indice k. A detailed explanation of the use of j (which may be omitted when convenient) will become clear in the presentation of the transformation given in section 3.

We use capital letters like A, B, C to denote rewrite derivations. An elementary derivation is denoted $A : u \xrightarrow{\ell_i,p} v$ where u reduces to v at occurrence p using a rule ℓ_i. In this notation, ℓ_i, p or A may be omitted. A derivation is a sequence $A = A_1 A_2 \ldots A_n$ of elementary derivations $A_i : u_i \rightarrow u_{i+1}$. We use the notation $A : u \xrightarrow{*} v$ to indicate the derivation A that starts with u and ends in v. We use 0 for the empty derivation, $\mathcal{D}(u)$ for the set of derivations issued from u, $\mathcal{F}(A)$ for the final term reached by a derivation A, $|A|$ for the length of derivation A and $\mathcal{R}edex(u)$ for the set of redex positions in term u. We introduce here two assumptions on rewrite derivations: the lexicographic rule priority and the rule fair rewrite derivation. The basic idea of the lexicographic rule priority is illustrated on the following example.

Exemple 2.1 *Given two labelled rewrite rules:*

$$\ell_1 : f_1(b,x) \rightarrow f(a) \qquad\qquad \ell_2 : f_1(b,True) \rightarrow g(a)$$

The following derivation A does not respect the lexicographic rule priority:

$$A : f_1(b,True) \xrightarrow{\ell_2,\Lambda} g(a)$$

since ℓ_1 can be applied at the top (Λ) position of the term $f_1(b,True)$.

Dfinition 2.2 *A rewrite derivation A respects the* lexicographic rule priority *if whenever $\ell^j_{i,k}$ is applied at position p of some term t in the derivation, then there is no $(i',j',k') <_{lex} (i,j,k)$, s.t. $\ell^{j'}_{i',k'}$ could be applied at t in p instead ($<_{lex}$ is the lexicographic extension of the ordering $<$ on naturals).*

Standard rewriting leads to two kinds of nondeterminism. First the choice of the rule to be applied from the set of the rules that can be applied and second, the choice of the subterm to be replaced. Both choices could be made nondeterministically.

Dfinition 2.3 *[PF94] A rewrite derivation A is* rule fair *when A is finite or A is infinite and every rule that can be applied infinitely often in A is effectively infinitely often applied in A.*

The lexicographic rule priority assumption is needed in order to ensure that the irreducibility flag will only be set appropriately as described in detail in section 3. The fairness hypothesis is needed in order to ensure that no rule will be delayed indefinitely.

Exemple 2.4 *Let $\mathcal{R} = \{a \rightarrow f(a), b \rightarrow c\}$. The infinite rewrite derivation $a \rightarrow f(a) \rightarrow f(f(a)) \rightarrow \ldots$ is fair as we apply infinitely the first rule and the second one can never be applied.*

Exemple 2.5 *Let $\mathcal{R} = \{a \rightarrow f(a), a \rightarrow b\}$. The infinite rewrite derivation $a \rightarrow f(a) \rightarrow f(f(a)) \rightarrow f(f(f(a)))\ldots$ is not rule fair as the second rule is never applied although it can be applied on every term along the rewrite derivation.*

Since we will be working on derivation equivalence, let us recall briefly the standard definitions we need in the following. For full details see [HL91, KM91].

A rewrite system is *orthogonal* if all rules are left-linear (every variable occurs only once on a left-hand-side of a rewrite rule) and if the non-overlapping condition holds. We briefly recall now the background on derivations in orthogonal rewrite systems.

Dfinition 2.6 *(disjoints redexes) Two redexes $(p_1, u_1 \to v_1)$ and $(p_2, u_2 \to v_2)$ are disjoints if:*

- $p_1 \not\leq p_2$ *and* $p_1 \not\geq p_2$ *(incomparable positions)*
- $p_2 > p_1$ *and there exists a variable position p in u_1 such that $p_1.p < p_2$, or symmetrically if $p_2 < p_1$ (compatible positions)*

The following notion of residuals formalises the duplication or the deletion of redexes during reduction.

Dfinition 2.7 *(residuals [HL91]) Given an elementary derivation $A : u \xrightarrow{\ell,p} v$ and $q \in \mathcal{R}edex(u)$, the set $q\backslash A$ of residuals of q by A is the following subset of $\mathcal{P}os(v)$:*

$$q\backslash A = \begin{cases} \emptyset & \text{if } p{=}q \\ \{q\} & \text{if } p \text{ and } q \text{ are incomparable positions or } p > q \\ \{p.r'.q_1\} & \text{if } q = p.r.q_1 \text{where } r, r' \text{ are positions of the same variable} \\ & \text{in the left-hand side and the right-hand side of } \ell. \end{cases}$$

For non elementary derivations A, we define $p\backslash A$ by:

$$p\backslash 0 = \{p\},$$
$$p\backslash (AB) = \{q\backslash B \mid q \in p\backslash A\}.$$

The residual mapping is extended to sets of redex positions by defining:

$$U\backslash B = \cup\{u\backslash B \mid u \in U\}.$$

Given two derivations $A, B \in \mathcal{D}(u)$ with $|A| = 1$, contracting the set $U \subseteq \mathcal{R}edex(u)$. We define the residual $A\backslash B$ of A by B as the elementary derivation in $\mathcal{D}(\mathcal{F}(B))$, contracting the set $U\backslash B$.

Given two derivations $A, B \in \mathcal{D}(u)$ with $|B| = 1$. We define $A\backslash B \in \mathcal{D}(\mathcal{F}(B))$ by induction on $|A|$:

$$0\backslash B = 0$$
$$(A_1A_2)\backslash B = (A_1\backslash B)(A_2\backslash (B\backslash A_1)) \text{ with } |A_1| = 1$$

Now for $A, B \in \mathcal{D}(u)$ of arbitrary lengths, we define $A\backslash B \in \mathcal{D}(\mathcal{F}(B))$ by induction on $|B|$:

$$A\backslash 0 = A$$
$$A\backslash (B_1B_2) = (A\backslash B_1)\backslash B_2 \text{ with } |B_1| = 1$$

.

Dfinition 2.8 *Let A,B two derivations, A is equivalent to B denoted $A \sim B$ if $\mathcal{F}(A) = \mathcal{F}(B)$.*

The result we will be using is the well know parallel move lemma:

Lemma 2.9 *Let A and B be two derivations in $\mathcal{D}(u)$, we have $A(B\backslash A) \sim B(A\backslash B)$.*

3 The transformation

We now describe our transformation of conditional rewrite systems into a unconditional rewrite systems.

3.1 Introduction of the new symbols

In order to perform the transformation, we proceed as follows:

— First we introduce for each symbol in $\mathcal{F}$ such that f is a top symbol of the left-hand side of a rule, i_f new function symbols f_i, $i \in [1..i_f]$ whose arity is the arity of f plus two. We introduce the set of function symbols $\mathcal{F}^{top} = \bigcup_{f \in \mathcal{F}} \mathcal{F}^f$ where $\mathcal{F}^f = \{f_i, i \in [1..i_f]\}$ and f is the top symbol of the left-hand side of a rule. If f is the top symbol of the left-hand side of a conditional rule $\ell_i : f(t_1^i, \ldots, t_n^i) \to r_i$ if $s_i \downarrow_R^! s_i'$, then the two new arguments introduced in the f_i allow us to refer first to the condition denoted $\downarrow (s_i, s_i')$ and second to the right-hand side of the rule. The right-hand side of the rules are introduced here only as a means to store the substitution that will be used when matching the new rule. It could be replaced if one wishes by the list of variables of the right-hand side. the following we denote by $\overline{r}$ the terms where rewriting is forbidden in non-variables positions of r (see example 3.1).

— Second, the term to be reduced, and thus the subsequent conditions introduced, are all evaluated using the transformed system. If the first extra argument of a function symbol f_i becomes $True$, we replace the term with top symbol f_i by its corresponding right-hand side.

This transformation of the rules in $\mathcal{R}_f$, is formally described as follows:

$$\mathcal{R}'_f = \left\{ \begin{array}{lll} \ell_1 : & f(t_1^1, \ldots, t_n^1) & \to f_1(t_1^1, \ldots, t_n^1, \downarrow (s_1, s_1'), \overline{r_1}) \\ & \vdots & \\ \ell_{i_f} : & f(t_1^{i_f}, \ldots, t_n^{i_f}) & \to f_{i_f}(t_1^{i_f}, \ldots, t_n^{i_f}, \downarrow (s_{i_f}, s_{i_f}'), \overline{r_{i_f}}) \\ \ell_{i_f+1} : & f_i(x_1, \ldots, x_n, True, \overline{r_i}) & \to r_i \\ \ell_{i_f+2} : & \downarrow (x, x) & \to True \end{array} \right.$$

This example explains why memorising the substitution is necessary.

Exemple 3.1 *Consider the following CTRS*

$$\mathcal{R} = \left\{ \begin{array}{l} f(g(x) \to p(x) \text{ if } c(x) \downarrow_{\mathcal{R}} tt \\ c(a) \to tt \end{array} \right.$$

The transformed system over the new signature is

$$\mathcal{R}' = \left\{ \begin{array}{l} f(g(x)) \to f_1(g(x), \downarrow (c(x), tt), \overline{p}(x))) \\ f_1(x_1, True, \overline{p}(x_2)) \to p(x_2) \\ \downarrow (x, x) \to True \\ c(a) \to tt \end{array} \right.$$

The condition part becomes an additional argument of f_1, the third argument $\overline{p}(x)$ (which can be replaced by x) makes possible to find the right substitution when the condition is evaluated to $True$. The conditional derivation $f(g(a)) \to p(a)$ is simulated by the unconditional one $f(g(a)) \to f_1(g(a), \downarrow (c(a), tt), \overline{p}(a))) \to f_1(g(a), True, \overline{p}(a))) \to p(a)$

The (unconditional) rewrite system $\mathcal{R}'_f$ resulting from the transformation of $\mathcal{R}_f$ is not quite satisfactory since when the condition is false, then the evaluation is blocked i.e. no rule further applies. This is due to the presence of the f_i operators that do not occur *inside* any rule of the transformed system. In fact the first idea would be to perform a completion of this new rewrite system. But we prefer, in particular for efficiency reasons, to directly handle

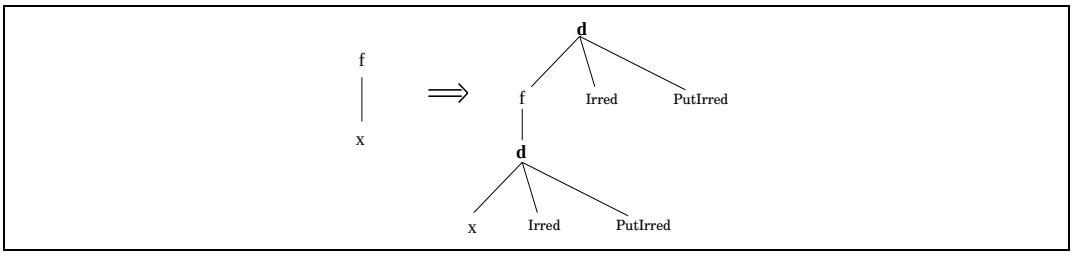

Figure 1: The Φ_{intro} transformation

the problem and to see how the set of derivations can be preserved by the transformation. What is needed in the situation above is to be able to return to the initial signature, to allow further reductions to occur. To this end we introduce a new set of unconditional rules $\mathcal{R}''_f$. These rules use an irreducibility flag in order to explicitly handle terms in normal form.

So, we add new function symbols to the signature to explicitely handle irreducibility of terms in the term structure. These new symbols can be viewed as decorations of the initial term. We will see later how this additional information in the term structure is used to prove the termination of the transformed rules. We introduce the set of symbols $\mathcal{F}' = \mathcal{F} \cup \mathcal{F}^{top} \cup \{\downarrow, True, False, \mathbf{d}, 0, \ldots, n, +\}$ where $True, False, 0, \ldots, n$ are new constants, $\mathbf{d}$ is a new function symbol of arity 3, $+, \downarrow$ are two new function symbols of arity 2, and n is the maximum number of rules in $\mathcal{R}_f$ for all $f \in \mathcal{F}$. The set $\{True, False, \mathbf{d}, 0, \ldots, n, +\}$ is denoted by $\mathcal{F}^{add}$ and the set $\{\mathbf{d}, 0, \ldots, n\}$ is denoted by $\mathcal{F}^{deco}$. We assume already defined the rewriting rules corresponding to the symbol $+$ that add two natural numbers. To simplify the notations, $\mathbf{d}^u_v(t)$ denotes the term $\mathbf{d}(t, u, v)$. All terms in $\mathcal{T}(\mathcal{F})$ are mapped on terms in $\mathcal{T}(\mathcal{F} \cup \mathcal{F}^{deco})$ using the transformation Φ_{intro} defined by the following rewrite rules.

Introduction of decorations

$$\begin{array}{lcll} \Phi_{intro}(f(x_1, \ldots, x_n)) & \rightarrow & \mathbf{d}^0_1(f(\Phi_{intro}(x_1), \ldots, \Phi_{intro}(x_n)) & n \geq 0 \\ \Phi_{intro}(\mathbf{d}^y_{y'}(x)) & \rightarrow & \mathbf{d}^y_{y'}(x) & \end{array}$$

The Φ_{intro} transformation is pictured in Figure 1.

Terms in $\mathcal{T}(\mathcal{F} \cup \mathcal{F}^{deco})$ are easily mapped back to the original term by just forgetting the $\mathbf{d}$ symbols with its two extra arguments, i.e. those decorations are removed. The mapping disregarding the term decoration is denoted Φ_{dis} defines as follows.

Removing extra arguments

$$\Phi_{dis}(\mathbf{d}^0_1(f(x_1, \ldots, x_n)) \rightarrow f(\Phi_{dis}(x_1), \ldots, \Phi_{dis}(x_n)) \quad n \geq 0$$

The second argument of $\mathbf{d}$ is intended to be 0 or 1, where 1 means that the corresponding first argument of $\mathbf{d}$ is in normal form. We use the third argument of $\mathbf{d}$ to decide when we put the irreducibility flag, i.e. this is a counter storing the number of failed conditional rules applied when all subterms of f are in $\mathcal{R}$-normal form. A term starting with d_i can be rewritten only by a rule with index higher then i. We define the mapping Φ_{irr} which changes the second argument of $\mathbf{d}$ to 1.

Adding the irreducibility flag

$$\Phi_{irr}(f(x_1, \ldots, x_n) \rightarrow \mathbf{d}^1_1(f(\Phi_{irr}(x_1), \ldots, \Phi_{irr}(x_n)) \quad n \geq 0$$

Condition introduction when all subterms of f are in $\mathcal{R}''$-normal form

$$\ell^1_{i,1}: \quad \mathbf{d}^0_i(f(\mathbf{d}^1_{y'_1}(t^i_1),\ldots,\mathbf{d}^1_{y'_n}(t^i_n))) \rightarrow \mathbf{d}^0_i(f_i(\mathbf{d}^1_{y'_1}(t^i_1),\ldots,\mathbf{d}^1_{y'_n}(t^i_n),\mathbf{d}^0_1(\downarrow(\Phi_{intro}(s_i))),\Phi_{intro}(s'_i)),r_i))$$

Condition introduction

$$\ell^2_{i,1}: \quad \mathbf{d}^0_x(f(\mathbf{d}^{y_1}_{y'_1}(t^i_1),\ldots,\mathbf{d}^{y_n}_{y'_n}(t^i_n))) \rightarrow \mathbf{d}^0_x(f_i(\mathbf{d}^{y_1}_{y'_1}(t^i_1),\ldots,\mathbf{d}^{y_n}_{y'_n}(t^i_n),\mathbf{d}^0_1(\downarrow(\Phi_{intro}(s_i))),\Phi_{intro}(s'_i)),r_i))$$

Conditional rule application

$$\ell^3_{i,1}: \quad \mathbf{d}^0_x(f_i(\mathbf{d}^{y_1}_{y'_1}(x_1),\ldots,\mathbf{d}^{y_n}_{y'_n}(x_n),True,y)) \rightarrow \Phi_{intro}(y)$$

Disregarding False condition when all x_i of f_i are in $\mathcal{R}''$-normal form

$$\ell^4_{i,1}: \quad \mathbf{d}^0_x(f_i(\mathbf{d}^1_{y'_1}(x_1),\ldots,d^1_{y'_n}(x_n),False,y)) \rightarrow \mathbf{d}^0_{x+1}(f(\mathbf{d}^1_{y_1}(x_1),\ldots,\mathbf{d}^1_{y_n}(x_n)))$$

Disregarding False condition

$$\ell^5_{i,1}: \quad \mathbf{d}^0_x(f_i(d^{y_1}_{y'_1}(x_1),\ldots,\mathbf{d}^{y_n}_{y'_n}(x_n),False,y)) \rightarrow \mathbf{d}^0_x(f(\mathbf{d}^{y_1}_{y'_1}(x_1),\ldots,\mathbf{d}^{y_n}_{y'_n}(x_n)))$$

Set the irreducibility flag for f symbol

$$\ell^6_{i_f,1}: \quad \mathbf{d}^0_x(f(\mathbf{d}^1_{y_1}(x_1),\ldots,\mathbf{d}^1_{y_n}(x_n))) \rightarrow \mathbf{d}^1_x(f(\mathbf{d}^1_{y_1}(x_1),\ldots,\mathbf{d}^1_{y_n}(x_n)))$$

Introduction of decorations

$$\ell^7_{i_f,1}: \Phi_{intro}(f(x_1,\ldots,x_n)) \rightarrow \mathbf{d}^0_1(f(\Phi_{intro}(x_1),\ldots,\Phi_{intro}(x_n)))$$
$$\ell^8_{i_f,1}: \Phi_{intro}(\mathbf{d}^y_{y'}(x)) \rightarrow \mathbf{d}^y_{y'}(x)$$

Set the irreducibility flag for other symbols

$$\ell^1_{1,i_{\mathcal{R}}+j}: \mathbf{d}^0_y(g(\mathbf{d}^1_{y_1}(x_1),\ldots,\mathbf{d}^1_{y_n}(x_n))) \rightarrow \mathbf{d}^1_y(g(\mathbf{d}^1_{y_1}(x_1),\ldots,\mathbf{d}^1_{y_n}(x_n)))$$

Testing syntactic equality

$$\ell^1_{1,i_{\mathcal{F}}+1}: \quad \mathbf{d}^0_1(\downarrow(\mathbf{d}^1_z(x),\mathbf{d}^1_{z'}(x))) \rightarrow True$$

$$\ell^2_{1,i_{\mathcal{F}}+1}: \quad \mathbf{d}^0_1(\downarrow(\mathbf{d}^1_z(x),\mathbf{d}^1_{z'}(y))) \rightarrow False$$

Figure 2: Transformed rules

3.2 The transformed rules

We assume that $\ell_{1,1}\ldots\ell_{i_f,1}$ are the labels of rewrite rules in $\mathcal{R}_f$. Using the previous notation, each rewrite rule $\ell_{i,1}$ in $\mathcal{R}_f$ $(1 \leq i \leq i_f)$ is transformed into rewrite rules in $\mathcal{R}''_f$ described in Figure 2.

$$
\begin{array}{ll}
\ell_{1,1} : ins(x, c(y,z)) & \rightarrow c(x, c(y,z)) \text{ if } x <= y \downarrow^{!} tt \\
\ell_{2,1} : ins(x, c(y,z)) & \rightarrow c(y, ins(x,z)) \text{ if } x <= y \downarrow^{!} ff \\
\ell_{3,1} : ins(x, nil) & \rightarrow c(x, nil) \\
\ell_{1,2} : isort(nil) & \rightarrow nil \\
\ell_{2,2} : isort(c(x,y)) & \rightarrow ins(x, isort(y)) \\
\ell_{1,3} : 0 <= x & \rightarrow tt \\
\ell_{2,3} : s(x) <= 0 & \rightarrow ff \\
\ell_{3,3} : s(x) <= s(y) & \rightarrow x <= y
\end{array}
$$

Figure 3: Rewrite specification of insertion sort $\mathcal{R}ewSort$

Hypothesis $\mathcal{H}_{\mathcal{R}''}$: *We assume throughout the paper that rewriting with $\mathcal{R}''$ obeys the following conditions:*

1. *All rewrite derivations are rule fair,*
2. *The lexicographic rule priority is applied.*

The first condition is needed for proving the termination of $\mathcal{R}''_f$ and the second one to ensure the application of the rule $\ell^{j}_{i,k}$ before applying an $\ell^{j'}_{i',k'}$ at position p, for each term in a derivation A when $(i,j,k) < (i',j',k')$.

We will see in section 5 how these rules are used in the implementation of concurrent conditional rewriting. Before showing the properties of this transformation, let us describe now the transformed rules. The first $\ell^1_{i,1}, \ell^2_{i,1}$ rules introduce the condition part as an extra argument of the symbol f_i and the substitution part as a second extra argument of f_i. The $\ell^3_{i,1}$ rule applies the original condition rule $\ell_{i,1}$ if the condition part is evaluated to $True$. When the condition associated to a term is evaluated to $False$, we distinguish two cases:

— In the case of $\ell^4_{i,1}$ where all subterms of an f_i symbol are irreducible, we increment the counter associated to the symbol f_i to enable the possible application of the next rule having f as a left-hand-side top symbol.

— In the dual case of the $\ell^5_{i,1}$ rule, when the irreducibility of all subterm of f_i have not been proved, we return to the initial term without changing the f decoration.

We distinguish these two cases to avoid the termination problem (see section 4) by using the irreducibility flag of the f_i subterms to decide when to set the irreducibility flag to 1 on f. This is done after testing the possible application of all the rules having f as top-left-symbol using the counter associated to f_i. This is illustrated in the example given below.

Assume $j \in \{0, \ldots, i_{\mathcal{F}} - i_{\mathcal{R}}\}$, for any symbol g with arity $n \geq 0$ which does not occur at the top of a left hand side in $\mathcal{R}$, we add the rules $\ell^1_{1,i_{\mathcal{R}}+j}$. The last rules $\ell^1_{1,i_{\mathcal{F}}+1}$ and $\ell^2_{1,i_{\mathcal{F}}+1}$ test the syntactic equality of the condition. Finally, for the unconditional rules of the initial system $\mathcal{R}$, we add a $True$ condition part to those rules to have a uniform transformation for both conditional and unconditional rewrite rules.

Remark: The additional symbol f_i and the second decoration (counter) of each symbol are unnecessary for unconditional rules. In fact, it is enough to transform an initial rule $u \rightarrow v$ to $\Phi_{intro}(u) \rightarrow \Phi_{intro}(v)$ and to add only the rewrite rules $\ell^6_{i_f,1}, \ell^7_{i_f,1}, \ell^8_{i_f,1}$ changing the decoration of function symbols.

The rewrite system described in Figure 3 enables us to sort a list of natural numbers. The rule $\ell_{1,1}$ of the example is transformed into the rules described in Figure 4.

Let us illustrate the use of the above transformed rules on the derivation A in $\mathcal{R}''ewSort$ issued from: $t = isort(c(s(s(s(0))), c(s(s(0), c(s(0), c(0, nil))))))$. We denote $t_i = \mathbf{d}^0_1(s(\mathbf{d}^0_1(\ldots \mathbf{d}^0_1(0))))$

$$\ell^1_{1,1} : \mathbf{d}^0_1(ins(\mathbf{d}^1_{y'_1}(x)(\mathbf{d}^1_{y'_2}(c(\mathbf{d}^1_{y'_3}(y), \mathbf{d}^1_{y'_4}(z)))))) \rightarrow$$
$$\mathbf{d}^0_1(ins_1(\mathbf{d}^1_{y'_1}(x)(\mathbf{d}^1_{y'_2}(c(\mathbf{d}^1_{y'_3}(y), \mathbf{d}^1_{y'_4}, \mathbf{d}^1_{y'_5}(z)))),$$
$$\mathbf{d}^0_1(\downarrow(\mathbf{d}^1_{y'_1}(x) <= \mathbf{d}^1_{y'_3}(y), \mathbf{d}^0_1(tt))), c(\mathbf{d}^1_{y'_1}(x), c(\mathbf{d}^1_{y'_3}(y), \mathbf{d}^1_{y'_5}(z)))))$$

$$\ell^2_{1,1} : \mathbf{d}^0_{\mathbf{x}}(ins(\mathbf{d}^{\mathbf{y_1}}_{y'_1}(x)(\mathbf{d}^{\mathbf{y_2}}_{y'_2}(c(\mathbf{d}^{\mathbf{y_3}}_{y'_3}(y), \mathbf{d}^{\mathbf{y_4}}_{y'_4}(z)))))) \rightarrow$$
$$\mathbf{d}^0_{\mathbf{x}}(ins_1(\mathbf{d}^{\mathbf{y_1}}_{y'_1}(x)(\mathbf{d}^{\mathbf{y_2}}_{y'_2}(c(\mathbf{d}^{\mathbf{y_3}}_{y'_3}(y), \mathbf{d}^{\mathbf{y_4}}_{y'_4}, \mathbf{d}^{\mathbf{y_5}}_{y'_5}(z)))),$$
$$\mathbf{d}^0_1(\downarrow(\mathbf{d}^{\mathbf{y_1}}_{y'_1}(x) <= \mathbf{d}^{\mathbf{y_3}}_{y'_3}(y), \mathbf{d}^0_1(tt))), c(\mathbf{d}^{\mathbf{y_1}}_{y'_1}(x), c(\mathbf{d}^{\mathbf{y_3}}_{y'_3}(y), \mathbf{d}^{\mathbf{y_5}}_{y'_5}(z)))))$$

$$\ell^3_{1,1} : \mathbf{d}^0_{\mathbf{x}}(ins_1(\mathbf{d}^{\mathbf{y_1}}_{y'_1}(x), True, y)) \rightarrow \Phi_{intro}(y)$$
$$\ell^4_{1,1} : \mathbf{d}^0_{\mathbf{x}}(ins_1(\mathbf{d}^1_{y'_1}(x), False, y)) \rightarrow \mathbf{d}^0_{\mathbf{x+1}}(ins(\mathbf{d}^1_{y'_1}(x)))$$
$$\ell^5_{1,1} : \mathbf{d}^0_{\mathbf{x}}(ins_1(\mathbf{d}^{\mathbf{y_1}}_{y'_1}(x), False, y)) \rightarrow \mathbf{d}^0_{\mathbf{x}}(ins(\mathbf{d}^{\mathbf{y_1}}_{y'_1}(x)))$$
$$\ell^6_{3,1} : \mathbf{d}^0_{\mathbf{x}}(ins(\mathbf{d}^1_{y'_1}(x))) \rightarrow \mathbf{d}^1_{\mathbf{x}}(ins(\mathbf{d}^1_{y'_1}(x)))$$
$$\ell^7_{3,1} : \Phi_{intro}(ins(x)) \rightarrow \mathbf{d}^0_1(ins(\Phi_{intro}(x)))$$
$$\ell^8_{3,1} : \Phi_{intro}(\mathbf{d}^y_{y'}(x)) \rightarrow \mathbf{d}^y_{y'}(x)$$
$$\ell^1_{1,10} : \quad \mathbf{d}^0_1(\downarrow(\mathbf{d}^1_z(x), \mathbf{d}^1_{z'}(x))) \rightarrow True$$

$$\ell^2_{1,10} : \quad \mathbf{d}^0_1(\downarrow(\mathbf{d}^1_z(x), \mathbf{d}^1_{z'}(y))) \rightarrow False$$

Figure 4: Part of the insertion sort unconditional rewriting system $\mathcal{R}''ewSort$

the natural number $i \geq 0$. Initially, the term t is mapped to :
$\Phi_{intro}(t) = \mathbf{d}^0_1(isort(\mathbf{d}^0_1(t_3, \mathbf{d}^0_1(c(t_2, \mathbf{d}^0_1(c(t_1, \mathbf{d}^0_1(c(t_0, \mathbf{d}^0_1(nil)))))))))$.
In $\mathcal{R}ewsort$ we have respectively $i_{\mathcal{F}} = 9$, $i_{\mathcal{R}} = 3$. It is easy to see that we can only apply the $\ell^2_{1,2}, \ell^2_{2,2}$ rewrite rules i.e the transformed rules of the original conditional rules $\ell_{1,2}, \ell_{2,2}$. Once we have applied these rules, the *isort* symbol is eliminated from the term t. We obtain the term $\mathbf{d}^0_1(ins(\mathbf{d}^0_1(t_3, \mathbf{d}^0_1(ins(t_2, \mathbf{d}^0_1(ins(t_1, \mathbf{d}^0_1(c(\ t_0, \mathbf{d}^0_1(nil)))))))))$. We describe only some steps of the derivation A detailed below. The different steps of the derivation are:

(1) In this step, we apply the rule $\ell^1_{1,1}$ whose condition part evaluates to $False$ and then we return to the initial term. We can remark here that this step can not be infinitely the same since the derivation has the rule fair property.

(2) We apply now the rules $\ell^6_{3,1}$ to set the normal flag for the subterms of ins.

(3) This step is identical to (1) but now with the subterms of ins in normal form.

(4) We apply the rule $\ell^1_{1,1}$ (failed applied rule) and we return to the initial term but the counter of ins is now two. This enables us to apply the next rule $\ell^2_{2,1}$ having *ins* as a top left symbol whose condition part is evaluated to $True$.

(5) We apply the rule $\ell^2_{2,1}$ to ins.

(6) We apply the rule $\ell^2_{2,1}$ twice to finally obtain the result.

Clearly, our transformation allows the parallel reduction of *ins* and *ins* (6) since the corresponding redexes are non-overlapping. Note that the different detailed steps above closely describe how the interpreter works (See section 5).

3.3 Termination

Except when explicitly specified, we assume in the following that the initial rewriting system satisfies:

Hypothesis $\mathcal{H}_{\mathcal{R}}$: *1- $\mathcal{R}$ is a decreasing conditional rewrite system.*
2- The lexicographic rule priority is applied.

The purpose of this section is to show that $\mathcal{R}''$ is terminating under hypothesis $\mathcal{H}_{\mathcal{R}}$ and $\mathcal{H}_{\mathcal{R}''}$.

$\mathbf{d}_1^0(ins(\mathbf{d}_1^0(t_3, \mathbf{d}_1^0(ins(t_2, \mathbf{d}_1^0(\underline{ins}(t_1, \mathbf{d}_1^0(c(t_0, \mathbf{d}_1^0(nil))))))))))$

$\downarrow * \ \ell_1^1$ *rule* application at $\underline{ins}$

$..\mathbf{d}_1^0(ins_1(t_1, \mathbf{d}_1^0(c(t_0, \mathbf{d}_1^0(nil)))), \mathbf{d}_1^0(\downarrow(\mathbf{d}_1^0(<(t_1, t_0), tt))), c(t_1, c(t_0, \mathbf{d}_1^0(nil)))))$

$\downarrow$ Reduction of condition

$..\mathbf{d}_1^0(ins_1(t_1, \mathbf{d}_1^0(c(t_0, \mathbf{d}_1^0(nil)))), False, c(t_1, c(t_0, \mathbf{d}_1^0(nil)))))$

$\downarrow$ Return to the initial term

$\mathbf{d}_1^0(ins(\mathbf{d}_1^0(t_3, \mathbf{d}_1^0(ins(t_2, \mathbf{d}_1^0(\underline{ins}(t_1, \mathbf{d}_1^0(c(t_0, \mathbf{d}_1^0(nil))))))))))$ } (1)

$\downarrow *$ Put the normal flag for subterms of $\underline{ins}$

$\ldots\ldots\ldots\ldots\ldots \mathbf{d}_1^0(\underline{ins}(\mathbf{d}_1^1(s(\mathbf{d}_1^1(0))), \mathbf{d}_1^1(c(\mathbf{d}_1^1(0), \mathbf{d}_1^1(nil)))))$ } (2)

$\downarrow *$ Same as (1) with subterms in normal form

$\ldots\ldots\ldots\ldots\ldots \mathbf{d}_2^0(\underline{ins}(\mathbf{d}_1^1(s(\mathbf{d}_1^1(0))), \mathbf{d}_1^1(c(\mathbf{d}_1^1(0), \mathbf{d}_1^1(nil)))))$ } (3)

$\downarrow *$ Reduction of $\underline{\text{ins}}$ $\ell_{2,1}^2$ *rule application*

$\ldots \mathbf{d}_2^0(\underline{ins_1}(\mathbf{d}_1^1(s(\mathbf{d}_1^1(0))), \mathbf{d}_1^1(c(\mathbf{d}_1^1(0), \mathbf{d}_1^1(nil)))), True, \mathbf{d}_1^0(c(...)))$ } (4)

$\downarrow *$ Reduction of $\underline{\text{ins}}$

$\ldots \mathbf{d}_1^0(c(\mathbf{d}_1^1(0)), \mathbf{d}_1^0(c(\mathbf{d}_1^1(s(\mathbf{d}_1^1(0)), \mathbf{d}_1^1(nil)))))$

$\downarrow$ Reduction of $\underline{\text{ins}}$

$\mathbf{d}_1^0(ins(\mathbf{d}_1^0(t_3, \mathbf{d}_1^0(\underline{ins}(t_2, \mathbf{d}_1^0(c(\mathbf{d}_1^1(0)), \mathbf{d}_1^0(c(\mathbf{d}_1^1(s(\mathbf{d}_1^1(0)), \mathbf{d}_1^1(nil)))))))))$ } (5)

$\downarrow$

$\mathbf{d}_1^0(\underline{\underline{ins}}(\mathbf{d}_1^0(t_3, \mathbf{d}_1^0(c(\mathbf{d}_1^1(0), \mathbf{d}_1^0(\underline{ins}(s(\mathbf{d}_1^1(s(\mathbf{d}_1^1(0)), \mathbf{d}_1^0(c(\mathbf{d}_1^1(s(\mathbf{d}_1^1(0)), \mathbf{d}_1^1(nil)))...))$ } (6)

$\downarrow *$

$\mathbf{d}_1^1(c(\mathbf{d}_1^1(0), c(\mathbf{d}_1^1(s(\mathbf{d}_1^1(0))), \mathbf{d}_1^1(c(\mathbf{d}_1^1(s(\mathbf{d}_1^1(s(\mathbf{d}_1^1(0))))), \mathbf{d}_1^1(nil))))))$

Figure 5: An example of derivation in $\mathcal{R}''ewsort$

Lemma 3.2 *If we do not assume $\mathcal{H}_{\mathcal{R}''}$, all infinite $\mathcal{R}''$-derivations are cyclic derivations: i.e. for all infinite $\mathcal{R}''$-derivation $t_1 \to \ldots \to t_k \ldots$ there exists $i, j \geq 1$ such that $i \neq j$ and $t_j = t_i$ where $t_i, t_j \in \mathcal{T}(\mathcal{F}')$.*

Proof: Given a derivation A issued from t_1 where $A : t_1 \to \ldots \to t_k$, $\mathcal{F}(A) = t_k$ and all t_j, t_i, $1 \leq j, i \leq k$ are different. We prove by construction and case analysis that all derivations B issued from t_k can be finite or infinite and in the last case the derivation B is necessarily cyclic. We recall that $\mathcal{R}''$ has the following properties:

–(1) All left hand sides of $\mathcal{R}''$ rules are by definition terms in $\mathcal{T}(\mathcal{F} \cup \mathcal{F}^{add}, \mathcal{X}) \cup \mathcal{T}(\mathcal{F}^{top} \cup \mathcal{F}^{add}, \mathcal{X})$

–(2) If the rules $\ell_{i_f,1}^6$ or $\ell_{j,1}^1$ are applied (rules that set the normal flag of the decoration $\mathbf{d}$ associated to f is 1). Further we can not apply rules (there is no match with any left hand sides of rules).

We recall that given a position p in a term, we introduce an f_i symbol which disappears only by the application of rules $\ell_{i,1}^3, \ell_{i,1}^4$ and $\ell_{i,1}^5$ since we have (1). Furthermore conditions are evaluated recursively by the same rewrite relation and then we proceed in the same way.

We study the case where the rules $\ell_{i,1}^4, \ell_{i,1}^5$ are applied in the derivation B from t_k. Assuming $t_k = \mathbf{d}_y^x(f(\mathbf{d}_{y_1'}^{y_1}(u_1), \ldots, \mathbf{d}_{y_n'}^{y_n}(u_n)))$ which is $\in \mathcal{T}(\mathcal{F} \cup \mathcal{F}^{deco})$ (the proof is similar if $t_k \in \mathcal{T}(\mathcal{F}')$). We analyse now the derivation B in the case of $\ell_{i,1}^4, \ell_{i,1}^5$ application. First, we recall that the condition associated to f_i is introduced with the rules $\ell_{i,1}^1$ or $\ell_{i,1}^2$. Four cases are discussed:

– (1) If the rule $\ell^3_{i,1}$ is applied then $t_j < t_k$, since $\mathcal{R}$ is decreasing.

– (2) If the rule $\ell^4_{i,1}$ is applied, the second decoration of $\mathbf{d}$ (the counter) is incremented and then the derivation is finite since at each failed application rule we increment the counter and then the final term does not match any left hand-side of a rule $(\mathbf{d^0_{if+1}}(f(t_k)))$ and finally we apply the $\ell^6_{i_f,1}$ rule to set the flag of irreducibility on f.

– (3) In the case of the application of $\ell^5_{i,1}$ and no rules are applied in subterms of t_k before the application of $\ell^5_{i,1}$, we obtain the same term t_j where $t_j = t_k$ and $i \neq j$, this is the only case where we get an infinite derivation s.t. B is a cyclic derivation.

– (4) In the case of the application of $\ell^5_{i,1}$ and a rule is applied in one subterms u_k of t_k. If the condition associated to the term t_k is $True$ then we have the case (2) otherwise we have the case (3). □

We state now the main lemma proving the termination of $\mathcal{R}''$

Lemma 3.3 *Under the hypothesis $\mathcal{H}_{\mathcal{R}''}$ and $\mathcal{H}_{\mathcal{R}}$, all rewrite derivations in $\mathcal{R}''$ are finite.*

Proof: Let us assume that there exists an infinite derivation A. By the previous lemma, A is cyclic and thus there exists i s.t. $A : t_i \to \ldots t_i \to \ldots$ (to simplify the notation, we omit to mention the $\mathbf{d}$s in the term structure). Without lost of generality, let us assume that a rule is applied on the top of the term t_i and that the top symbol of t_i is f. Since A is a cyclic derivation, the first extra argument of f_i (the condition c) is $False$ (necessarily the rule $\ell^5_{i,1}$ is applied). The cyclic derivation is then: $A : f(u_1, \ldots, u_n) \to f_i(u_1, \ldots, u_n, c, r_i) \ldots \to f(u_1, \ldots, u_n)$ and at least one subterm of t is not in normal form (otherwise if all u_i are in normal form, the derivation A is finite). The derivation $t_i \xrightarrow{*}_{\mathcal{R}''} t_i$ is cyclic but it is not fair since a rule that can be applied has been infinitely delayed. □

To conclude, we have shown the termination of $\mathcal{R}''$ under the hypothesis $\mathcal{H}_{\mathcal{R}''}$ and $\mathcal{H}_{\mathcal{R}}$:

Corollary 3.4 *Under the hypothesis $\mathcal{H}_{\mathcal{R}''}$ and $\mathcal{H}_{\mathcal{R}}$, $\mathcal{R}''$ is terminating.*

4 Correspondence of derivations

After the presentation of the transformation in the last section, we detail now a correspondence between the derivations in the original conditional rewriting system $\mathcal{R}$ and the derivations in the transformed one $\mathcal{R}''$. The general idea is to map each conditional rewriting step in $\mathcal{R}$ to a derivation in $\mathcal{R}''$ that introduces the evaluation of the condition, evaluates the condition and finally applies the conditional rule (when the condition is evaluated to $True$). This first correspondence, which is the simplest one, is given by the completeness result (theorem 4.10). On the other hand, given a derivation in $\mathcal{R}''$, it is not obvious to have an equivalent one in $\mathcal{R}$ since the evaluation of a condition can be done concurrently with other rewrite applications. Theorem 4.8 establishes a correspondence from $\mathcal{R}''$ derivations to equivalent ones in $\mathcal{R}$. Thus this section shows that the proposed transformation provides a rewrite system that simulates in a correct and complete way the application of the initial conditional rules (Figure 6).

Exemple 4.1 *Consider the CTRS defined in the example 3.1 and the following derivation A in $\mathcal{R}''$.*

$$f(g(f(g(a)) \to f_1(g(\underline{f(g(a))}), \downarrow(c(f(g(a)), tt)), \overline{p}(a)) \to$$
$$f_1(g(\underline{f_1(g(a), \downarrow(c(a), tt), \overline{p}(a))})), \downarrow(c(f(g(a)), tt)), \overline{p}(a))$$

It is clear that we can introduce a condition evaluation corresponding to the term $\underline{f(g(a))}$ before ending the evaluation of the first condition $\downarrow(c(f(g(a)), tt))$. This is why the correctness of the derivation correspondence is not obvious.

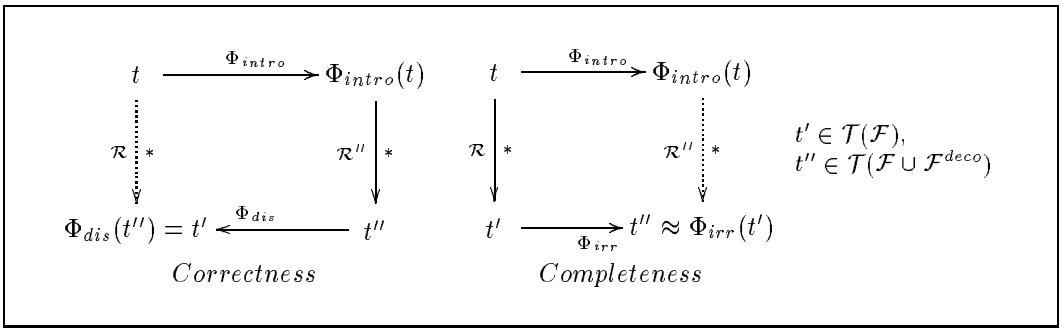

Figure 6: The derivation correspondence (correctness and completeness)

During the correctness proof, the goal is to "collect" rewriting steps of the derivation evaluating the condition. The general idea is to transform a derivation in $\mathcal{R}''$ into an equivalent one in $\mathcal{R}''$ consisting of rewriting subsequences. Each rewriting subsequence simulates an application of a conditional rule in $\mathcal{R}$.

To simplify the notation, we do not explicitly represent the decoration $\mathbf{d}$ at each term position. This does not change the proofs since the decorations are introduced only to help us to deal with the termination of the transformed rewriting system $\mathcal{R}''$ and does not interact with the derivation correspondence. We also use the fact that the rules in $\mathcal{R}''$ could be assumed to be left linear since all non-left linear rules could be transformed into left linear conditional rules by keeping the equality constraint of non-left linear variables in the condition. We distinguish between four types of elementary derivations in $\mathcal{R}''$: I^p for the $\ell^1_{i,1}, \ell^2_{i,1}$ rule application, T^p for the $\ell^3_{i,1}$ rules application, F^p for the $\ell^4_{i,1}, \ell^5_{i,1}$ rules application and D^p for the $\ell^6_{i_f,1}, \ell^7_{i_f,1}, \ell^8_{i_f,1}, \ell^1_{1,i_{\mathcal{R}}+j}$ rules application at position p for all $1 \leq i \leq i_f$ and $1 \leq j \leq i_{\mathcal{F}-} i_{\mathcal{R}}$.

Remember that the right-hand side $\overline{r}$ of rules is introduced here only as a mean to store the values gathered by the matching substitution that will be used when applying the rule. In the following we denote by $\overline{r}$ the terms where rewriting is forbidden in non-variable positions of r. We use the following notation for specific derivations:

$U_i : f_i(u_1, \ldots, u_i^1, \ldots, u_n, \downarrow (v, v'), \overline{r}) \xrightarrow{p_1} \ldots \xrightarrow{p_k} f_i(u_1, \ldots, u_i^k, \ldots, u_n, \downarrow (v, v'), \overline{r})$
with $p_j \in \cup_{1 \leq j \leq k} \mathcal{POS}(u_i^j)$,
$U_i = U_i^{p_1} \ldots U_i^{p_j}$ and where $U_i^{p_1}, \ldots, U_i^{p_j}$ are elementary derivations.
U denotes the derivation $U_1 \ldots U_n$.

$R : f_i(u_1, \ldots, u_i, \ldots, u_n, \downarrow (v, v'), \overline{r_1}) \xrightarrow{p_1} \ldots \xrightarrow{p_k} f_i(u_1, \ldots, u_i, \ldots, u_n, \downarrow (v, v'), \overline{r_k})$
with $p_j \in \cup_{1 \leq j \leq k} \mathcal{POS}(\overline{r_j})$.

$C : f_i(u_1, .., u_n, v_1, \overline{r}) \xrightarrow{p_1} \ldots \xrightarrow{p_j} \ldots \xrightarrow{p_k} f_i(u_1, ..., u_n, c_k, \overline{r})$ with $p_j \in \cup_{1 \leq j \leq k} \mathcal{POS}(c_j)$.

A denotes an U, C or R derivation. D^* denotes a derivation composed by D elementary derivations. $A^{>p}$ (respectively $A^{<p}$) denotes a derivation where rewriting occurs in position q such that $p > q$ (respectively $p < q$). From now on, we assume that all the derivations are $\mathcal{R}''$ derivations. If we define U, C, R and A as above, we obtain the following result.

Lemma 4.2 *Each rewrite derivation $A = I^p A' T^p$ with:*

$$A' : t[f_i(u_1, \ldots, u_n, \downarrow (v, v'), \overline{r}]_p \xrightarrow{*} t[f_i(u'_1, \ldots, u'_n, True, \overline{r'}]_p.$$

is equivalent to the derivation $B = I^p U C R T^p$.

Proof: The derivation A' can be transformed into $A'' = UCR$ since all derivations in U, C and R occur at disjoint positions. □

A similar lemma holds symmetrically if we replace T^p by F^p and $True$ by $False$.

4.1 Correctness of the transformation

Let us now show that for each rewrite derivation in $\mathcal{R}''$ there exists an equivalent rewrite derivation in $\mathcal{R}$ as described in Figure 6 under:

Hypothesis $\mathcal{H}$:

- *$\mathcal{R}$ satisfies $\mathcal{H}_{\mathcal{R}}$ and is confluent,*
- *$\mathcal{R}''$ satisfies $\mathcal{H}_{\mathcal{R}''}$.*

4.2 Correspondence of derivations in $\mathcal{R}''$

The next lemmas shows how to transform a derivation in $\mathcal{R}''$.

Dfinition 4.3 *If a term $t[v]_p$ rewrites at occurrence p with the rule $u \to u'$ into $t' = t[\sigma u']_p$ with $\sigma u = v$, q is called a* variable substitution position *of t' if $q = p.q'$ where q' is a variable position of u'.*

From now on, we assume $I^p : t[f(u_1, \ldots, u_n)]_p \to t[f_i(u_1, \ldots, u_n, c_i, \overline{r_i})]_p$

Lemma 4.4 *The derivation $A = I^p C^{p.(n+1)} U^{>p} R^{p.q} U'^{>p} C^{p.(n+1)} T^p$ is equivalent to $B = I^p C^{p.(n+1)} C^{p.(n+1)} T^p R^{p.q}$ where q is a variable substitution position of $\overline{r_i}$ and $A, B \in \mathcal{D}(t[f(u_1, \ldots, u_n)]_p)$.*

Proof: The elementary derivation $I^p : t[u]_p \to t[f_i(u_1, \ldots, u_n, c_i, \overline{r_i})]_p$ introduced the condition evaluation at position p. Since the condition c_i is evaluated to $True$ (the last rule applied in the derivation A is $\ell^3_{i,1}$), the derivations $U^{>p}, U'^{>p}$ are unnecessary derivations. As a consequence the derivation A is equivalent to $A_1 = I^p C^{p.(n+1)} R^{p.q} C^{p.(n+1)} T^p$. As q is a variable substitution position of $\overline{r_i}$, we can apply the permutation lemma to the derivation $R^{p.q} C^{p.(n+1)} T^p$ which is equivalent to $C^{p.(n+1)} T^p R^{p.q}$. As a consequence $A \sim B$ where $B = I^p C^{p.(n+1)} C^{p.(n+1)} T^p R^{p.q}$ □

Lemma 4.5 *The derivation $A = I^p C^{p.(n+1)} U^{>p} R^{>p} U'^{>p} C'^{p.(n+1)} T^{p.(n+1)} I^q$ is equivalent to $B = I^p C^{p.(n+1)} R^{>p} I^q (C'^{p.(n+1)} \backslash I^q)(T^{p.(n+1)} \backslash I^q)$ where:*

- *p, q are disjoint positions,*
- *$A, B \in \mathcal{D}(t[f(u_1, \ldots, u_n)]_p)$*
- *$p > q$.*

Proof: All left hand sides of $\mathcal{R}''$ rules are by definition terms in:

$$\mathcal{T}(\mathcal{F} \cup \mathcal{F}^{add}, \mathcal{X}) \cup \mathcal{T}(\mathcal{F}^{top} \cup \mathcal{F}^{add}, \mathcal{X}) \tag{1}$$

As a consequence the rules associated to the derivations I^p and I^q do not overlap. We apply the permutation lemma to get the result. □

In order to transform any $\mathcal{R}''$-derivations, the idea behind the following lemma is to "pull back" all redex residuals introduces by I^p to allow the application of the parallel move lemma.

Lemma 4.6 *Given a derivation $A = I^p C^{p.(n+1)} U^{>p} R^{>p} I^q Q$, then there exists derivations C_1,T_1,Q' and Q'' such that:*

- $A \sim B$,
- $B = I^p C^{p.(n+1)} R^{>p} I^q Q'(C_1 \backslash (I^q Q'))(T_1 \backslash (I^q Q'))Q''$,
- $p > q$,
- $Q \sim Q'(C_1 \backslash (I^q Q'))(T_1 \backslash (I^q Q'))Q''$,

where the condition introduced by I^p evaluates to True using the derivation $C' = C^{p.(n+1)} C_1$ and $\mathcal{F}(A) \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}^{deco})$.

Proof: Recall that all left hand sides of rules of $\mathcal{R}''$ rules are in the set $\mathcal{T}(\mathcal{F} \cup \mathcal{F}^{add}) \cup \mathcal{T}(\mathcal{F}^{top} \cup \mathcal{F}^{add})$. Since in position p we have a symbol f_i and $\mathcal{F}(A) \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}^{deco})$, the redex denoted R_p in position p must be reduced. Since condition associated to the f_i is True. It remains True for redex residuals from f_i which are reduces to the same term (since $\mathcal{R}$ is confluent by hypothesis) We examine now the derivation $Q = Q_1 ..... Q_n$. Q is decomposed by construction as follows: Assuming Q_i the elementary derivation that introduces the last redex residual issued from R_p.

$Q = Q_1 ... Q_i ... Q_n$

which is equivalent to $Q \sim Q'(C_1 \backslash (I^q Q'))(T_1 \backslash (I^q Q'))Q''$.

$C_1 \backslash (I^q Q') \sim C^{p_1.(n+1)} \ldots C^{p_l.(n+1)}$ and

$T_1 \backslash (I^q Q') \sim T^{p_1.(n+1)} \ldots T^{p_l.(n+1)}$

where $p_i \in p \backslash Q'$

Finally, we ignore $U^{>p_i}$ derivations that occurs in the first n subterms of f_i since the condition associated to f_i is True. The derivation A is equivalent to:

$B = I^p C^{p.(n+1)} R^{>p} I^q Q'(C_1 \backslash (I^q Q''))(T_1 \backslash (I^q Q'))Q'$ with the hypothesis defined in the lemma statement. □

A pictorial explanation of the previous lemma is given in Figure 7. A similar lemma holds when the condition introduced by I^p is evaluated to $False$.

Lemma 4.7 *Given a derivation $A = I^p C^{p.(n+1)} U^{>p} R^{>p} I^q Q$, there exist derivations C_1,T_1,Q' and Q'' such that:*

- $A \sim B$
- $B = I^p C^{p.(n+1)} U^{>p} R^{>p} I^q Q'(C_1 \backslash (I^q Q'))(F_1 \backslash (I^q Q'))Q''$,
- $p > q$,
- $Q \sim Q'(C_1 \backslash (I^q Q'))(F_1 \backslash (I^q Q'))Q''$,

where the condition introduced by I^p evaluates to $False$ using the derivation $C' = C^{p.(n+1)} C_1$ and $\mathcal{F}(A) \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}^{deco})$.

Proof: Same proof as lemma 4.5. □

We denote by e the transformation, described in the previous lemmas, from the derivation A to the derivation B. Given a derivation A in $\mathcal{R}''$, where $\mathcal{D}(A)$ and $\mathcal{F}(A) \in T(\mathcal{F} \cup \mathcal{F}^{deco})$ the transformation $\mathcal{T}$ is defined by:

$$\begin{cases} \mathcal{T}(A) & = \mathcal{T}(e(A)) \\ \mathcal{T}(I^p C^{p.(n+1)} T^p B) & = I^p \mathcal{T}(C^{p.(n+1)}) T^p \mathcal{T}(B) \\ \mathcal{T}(I^p C^{p.(n+1)} U^{>p} R^{>p} F^p B) & = \mathcal{T}(U^{>p} B) \\ \mathcal{T}(DA) & = D\mathcal{T}(A) \end{cases}$$

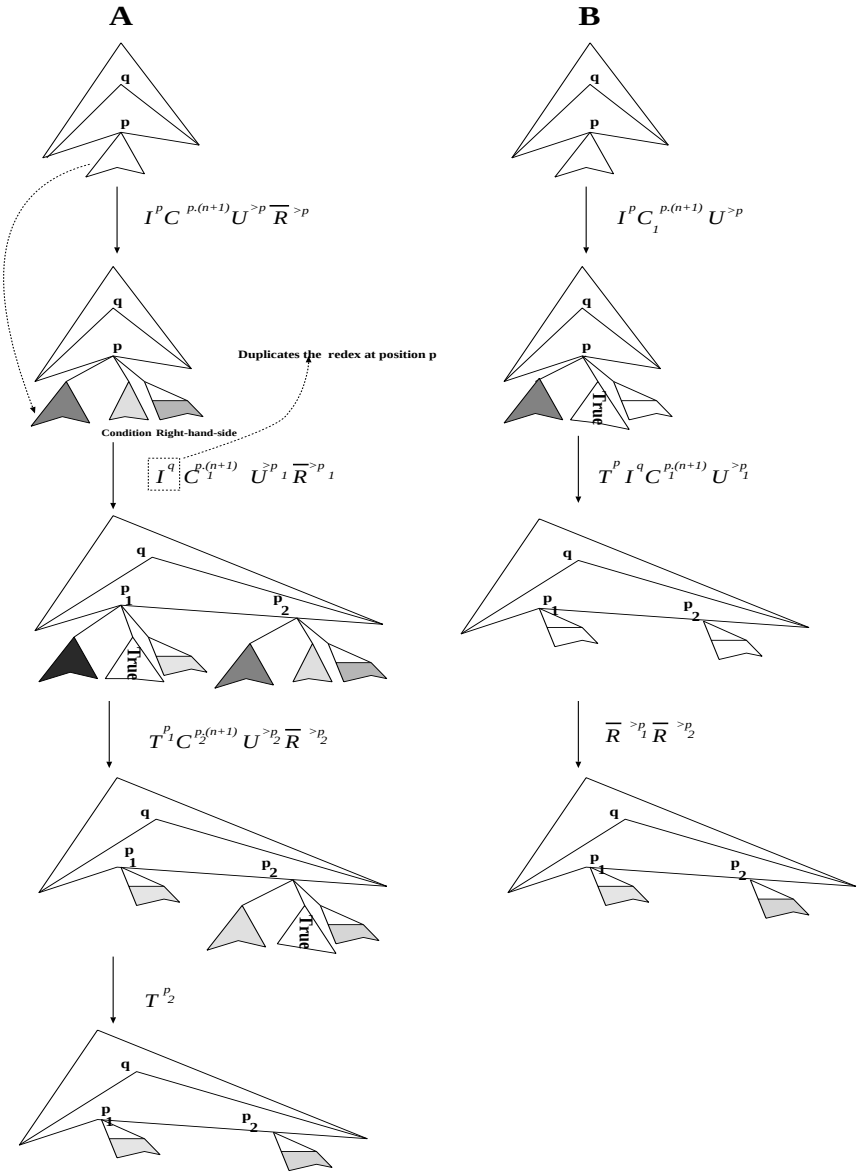

Figure 7: Pictorial illustration of Lemma 4.6

Theorem 4.8 *Each rewrite derivation A in $\mathcal{R}''$ is equivalent to a derivation B having the following structure, $B = D_1^* I_1 C_1 T_1 \ldots D_j^* I_j C_j T_j D_{j+1}^*$ where j is obtained after applying transformation $\mathcal{T}$ to A.*

Proof: The proof is based on the application of the three previous lemmas. Since A is finite the transformation terminates (We have a finite number of I derivations in A) Given a rewrite derivation:

$A : \Phi_{intro}(t) \xrightarrow{*}_{\mathcal{R}''} t''$, such that t'' is in $\mathcal{R}''$ $t'' \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}^{deco})$

We apply the transformation $\mathcal{T}$ to the derivation A to obtain the final result. □

Corollary 4.9 *Assuming $\mathcal{H}$, for each rewrite derivation $A : \Phi_{intro}(t) \xrightarrow{*}_{\mathcal{R}''} t''$ such that t'' is in $\mathcal{T}(\mathcal{F} \cup \mathcal{F}^{deco})$ there exists a rewrite derivation $B : t \xrightarrow{*}_{\mathcal{R}} t'$ such that t' is in $\mathcal{R}$ and $t' = \Phi_{dis}(t'')$.*

This means that each rewrite derivation in $\mathcal{R}''$ is simulated by a rewrite derivation in $\mathcal{R}$, where the two derivations lead to the same term using the mapping Φ_{dis}.

4.3 Completeness of the transformation

Now, we show that each derivation on the conditional rewriting system is easily mapped into a rewrite derivation in $\mathcal{R}''$. We recall that Φ_{irr} a mapping that given a term t in $\mathcal{T}(\mathcal{F})$ adds to t the necessaries normal flag decoration information to obtain a term in $\mathcal{T}(\mathcal{F} \cup \mathcal{F}^{deco})$ i.e. each term in $\mathcal{R}$-normal form is transformed into a term in $\mathcal{R}''$ normal form with the mapping Φ_{irr}. We write $t \approx t'$ if $t, t' \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}^{deco})$ and t, t' are syntactically equal except on the third argument of $\mathbf{d}$. For example, $\mathbf{d_0^1}(f(t)) \approx \mathbf{d_2^1}(f(t))$.

Theorem 4.10 *Assuming $\mathcal{H}$, for each rewrite derivation $t \xrightarrow{*}_{\mathcal{R}} t'$ there exists a rewrite derivation $\Phi_{intro}(t) \xrightarrow{*}_{\mathcal{R}''} t''$ where $t'' \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}^{deco})$ and $t'' \approx \Phi_{irr}(t')$.*

Proof: It is clear that each application of a conditional rule is mapped into rewriting steps in $\mathcal{R}''$ introducing the evaluation of the condition then evaluating the condition. Finally we apply the D rewrite rules at each occurrence of the term to add the necessary decorations into the structure of term. □

5 Our implementation of conditional concurrent rewriting

We now describe our implementation of conditional concurrent rewriting on distributed memory machines. Following [GKM87] an implementation based on term hypergraph rewriting has been defined [KV92, Alo93, Alo95] on MIMD machines using both a message passing mechanism between processors and a distributed matching algorithm [KV92]. The implementation of these ideas, RECO has been built using a fine-grained parallelism on terms.

The concurrent matching algorithm uses a bottom up approach. Its message-passing implementation avoids all the locking and synchronisation problems. We briefly describe how it works. Given a rewriting system, an automaton is precomputed from the rewriting system. Each node of the term has a matching state. Using this automaton, according to the matching state of its children we lookup to the automaton giving us the new matching state. Finally, having a matching state, we lookup to a table giving us a list of rewrite rules that can be applied. We discuss now the different ways to implement the detailed transformation.

5.1 Direct implementation of the transformation

The first method to implement the conditional concurrent rewriting is to use our implementation of concurrent rewriting directly since the transformed rewriting system is unconditional. By directly we mean by first generating the transformed system and then load and run the transformed system on the implementation. But, we have to verify the lexicographic rule priority and the rule fairness assumptions since the transformation is correct under the previous assumptions. We ensure the rule fairness assumption by treating nodes of the term in a round robin fashion i.e. each set of nodes in a processor is treated in a circular way. The lexicographic rule priority assumption is verified since at each term position the precompiled matching algorithm gives a list of possible rewrite rules that can be applied. The order in the list is the same order as they appear in the labelled rewriting system.

5.2 Indirect implementation of the transformation

The second approach, and indeed the one we have implemented, consists of loading the CdCcRw directly and to modifying slightly the unconditional concurrent rewriting implementation in order to simulate the transformation described in 3. In this way these is no need to explicitly add in the *term* structure the **d** symbol with its two extra arguments. We choose this method because the irreducibility flag is information which is already defined in the node structure i.e. the representation of this flag in the transformed rewriting system is only needed to prove the correctness and completeness of the transformation. Moreover, all left-hand sides of rules are in $\mathcal{T}(\mathcal{F} \cup \mathcal{F}^{add}) \cup \mathcal{T}(\mathcal{F}^{top} \cup \mathcal{F}^{add})$ so there is no pattern that match terms built on $\mathcal{T}(\mathcal{F} \cup \mathcal{F}^{top})$ which allows to use the automaton built on the initial signature.

We briefly present now an outline of the basic operations needed to implement conditional concurrent graph rewriting on a simple example. Two basic operations are implemented, matching and replacement.

The term to be reduced is represented as a jungle (a special acyclic directed hyper-graph) which is distributed over the processors. Jungles are based on objects (nodes and edges) and pointers (local and distant pointers). The basic operations needed for performing a replacement are implemented using message passing. To solve the lack of global state, each node stores a data structure called photo which is a part of the graph below the node. The photo, denoted in prefix notation (we assume that we memorise the arity of each symbol of the rewriting system in a global table), is used to calculate addresses of variables substitutions and then rewriting is done with those photos which remain coherent with the matching information.

The two main characteristics of the implemented model are:

(1) No explicit parallelisation directives are to be given, the same program may run on a sequential or a parallel machine.

(2) No explicit synchronisation is needed.

RECO offers two mechanisms for right-hand side rewrite rule creation. The first, called local rewriting, creates the right side locally in a processor. Figure 8 shows an application of the rule $< F(x) + F(y) \rightarrow F(x+y)$ if $F(x) \downarrow_R^! a >$ when a term is distributed over three processors P1,P2 and P3. Conditional replacement (Fig 8) is done as follows:

- Phase 1 creates the condition part of the rule and stores in node N_1 the N_1 photo denoted P_{N_1}.
- Phase 2 computes addresses of variables substitution of the condition (using the N_1 photo which is $N_1N_2N_3N_4N_5$) by sending a connect messages to nodes N_3 and N_4.

- In phase 3, each node receiving a connect message (for example node N_3) creates the ascendant pointer and sends match messages which contain its photo and its matching information.
- In phase 4, when N_7 is in normal form, N_7 sends a message *condition* to N_1 telling the result of the condition evaluation.
- In phase 5, if the condition is evaluated to $True$, a message condition is sent to node N_1 giving the result of the condition evaluation ($True$ or $False$) to N_1. Then replacement and instantiation of the right hand side (phase 6 and 7) is done in the same way as phase 2 and 3 using the photo P_{N_1} . If the condition is evaluated to $False$ no creation of right hand sides is needed. We note here, that in both previous cases node N_7 becomes garbage and node N_1 is reactivated only after the condition evaluation.

Remember that in the CcRw model, the normalisation flag which is contained in each node structure, is used to detect the termination of the concurrent reduction process. This normalisation information is easily updated by a bottom-up message mechanism. In the CdCcRw model, the normalisation flag is updated as described in the transformation detailed in section 3. Given a node N whose subterms are all in normal form, we set the normalisation flag in N if no unconditional rule can be applied at N or if all conditions of the conditional rules tried to N are evaluated to $False$.

A second mechanism, not detailed here, offers the possibility to create right-hand sides of rewrite rules on a distant processor. Thus, load balancing is possible when creating nodes and then can be done at the same time when applying rules. This is an important feature of the model since we do not have to move subgraphs from one processor to another one.

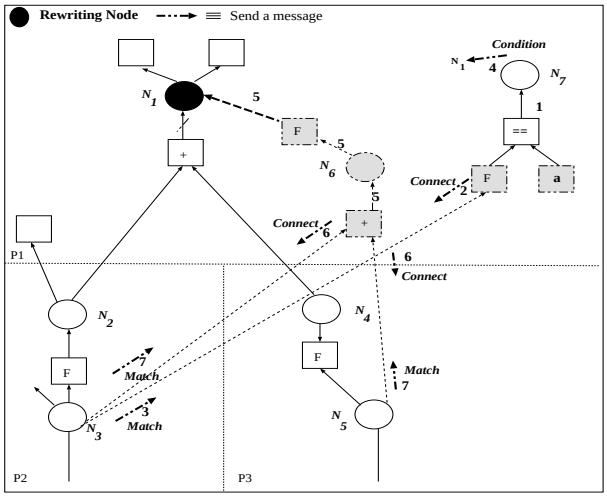

Figure 8: A conditional concurrent rewriting step

5.3 Some experimental results

The implementation RECO (conditional COncurrent REwriting) together with its garbage collection algorithm [Alo95], runs on various hardware platforms. From a clusters of machines (Sparc stations, HP stations) to different parallel distributed memory machines architectures(IBM SP/1, Power Challenge Array). It is written in C using the library PVM (parallel virtual machine) [BDGG94] for messages passing primitives. The implementation

of conditional concurrent rewriting is totally asynchronous. All primitives for receiving messages are non-blocked calls. Processes running on each processor never wait for messages. The concurrent normalisation process is non-deterministic, this leads to different possible concurrent executions for computing a normal form of a term. The benchmarks presented below are based on the average values of several normalisations of a given term on a cluster of Solaris Sun4 WorkStations, and on a Power Challenge Array (PCA) with 8 R8000 processors.

We present benchmarks based on using the system $\mathcal{R}ewSort$ sorting a list of natural numbers defined in section 3 and the rewrite system $\mathcal{R}_1$ which presents a quite unfavourable case where the structure of terms used in the tests have a large amount of overlapping redexes. Table 1 summarises, for each concurrent normalisation process, the rewriting system we use, the size of terms to be normalised, the platform and the number of processors used, the number of rewriting steps to obtain the normal form, the real time of execution given in second, and the different speed-ups.

$$\mathcal{R}_1 = \begin{cases} \ell_1 : F(x) + F(y) \to F(x+y) \text{ if } g(x) \downarrow^! tt & \ell_6 : x + 0 \to x \\ \ell_2 : F(x) + F(y) \to F(y+x) \text{ if } g(y) \downarrow^! ff & \ell_7 : g(0) \to tt \\ \ell_3 : x + (y+z) \to (x+y) + z & \ell_8 : g(s(x)) \to tt \\ \ell_4 : F(0) \to 0 & \ell_9 : g(f(x)) \to ff \\ \ell_5 : 0 + x \to x & \ell_{10} : g(x+y) \to tt \end{cases}$$

Example	#Term	Platform	#Rew.					Real time					Speed-ups				
		# Proc	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
$\mathcal{R}_1$	767	Sun4	575	612	629	689	704	6.46 s	4.81 s	4.61 s	4.63 s	4.96 s	1	1.34	1.9	2.4	3.32
$\mathcal{R}_1$	1535	Sun4	1279	1374	1386	1437	1583	13.30 s	9.5 s	6.33 s	4.29 s	3.39 s	1	1.4	2.1	3.1	3.92
$\mathcal{R}_1$	1535	PCA	1279	1389	1410	1454	1572	4.1 s	2.84 s	1.86 s	1.5 s	1.28 s	1	1.44	2.2	2.73	3.2
$\mathcal{R}ewSort$	269	Sun4	4744	6329	6487	7654	8324	10.88 s	7.88 s	5.97 s	5.13 s	3.2 s	1	1.38	1.82	2.12	3.4
$\mathcal{R}ewSort$	269	PCA	4744	6329	6487	7654	8324	3.03 s	2.86 s	2.24s	2.1 s	2.0 s	1	1.05	1.35	1.26	1.5
$\mathcal{R}ewSort$	520	PCA	13631	15439	16543	17656	17704	6.46 s	4.81 s	4.61 s	4.63 s	4.96 s	1	1.34	1.9	2.4	3.32

Table 1

These experimental results shows that the total number of applied rules in a concurrent execution is approximately equal to those applied in a sequential execution. It is clear, that big terms have greater speed-ups (up to 3.1 for terms having 1535 nodes distributed on four processors) than smaller ones. Although, increasing the number of processors does not lead to a better speed-up if the term size is not increasing too. The experimental results are showing the importance of the interprocess communication in our CdCcRw implementation. Moreover, in the case of SUN4 experiments, the network used is based on a sequential bus (Ethernet). A higher bandwidth network is certainly needed to have better experimental results. Furthermore, we are now sharpening the current implementation.

6 Conclusions and further work

We have shown that a special transformation from conditional term rewriting to unconditional term rewriting can be successfully used to implement efficiently the conditional *concurrent* rewriting model on *distributed* memory machines. This transformation provides a terminating rewrite system that simulates in a correct and complete way the application of the initial conditional rewrite rules. Its main originality and usefulness is to preserve as much as possible the inherent concurrency present in the initial system. From a practical point of view, this work allows us to extend existing concurrent rewriting implementation to the conditional case. We have described our implementation of conditional concurrent rewriting on distributed memory machines. Finally, we believe that the techniques and ideas introduced in this paper are a contribution towards solving the problem of designing efficient normalisation procedures of terms in concurrent deduction. This will certainly help to develop practical parallel implementations of languages like MAUDE [Mes93] and ELAN [KKV95] based on conditional rewriting with strategies.

Acknowledgements This work is partially supported by the Esprit Basic Research working group 6028, CCL. We would like to thank Patrick Viry, Christopher Lynch and Christelle Scharff for their comments.

References

[AGM90] H. Aida, G. Goguen, and J. Meseguer. 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 *Lecture Notes in Computer Science*, pages 320–332. Springer-Verlag, June 1990.

[Alo93] Ilies Alouini. Réécriture concurrente: Etude et implantation. Rapport de DEA, Université de Nancy 1, September 1993.

[Alo95] Iliès Alouini. Concurrent garbage collection for concurrent rewriting. In J. Hsiang, editor, *Proceedings 6th Conference on Rewriting Techniques and Applications, Kaiserslautern (Germany)*, volume 914 of *Lecture Notes in Computer Science*, pages 132–146. Springer-Verlag, 1995.

[BDGG94] A. Beguelin, J.J Dongarra, and A Geist G. Parallel virtual machine user's guide and reference manual. Technical report, Oak Ridge National Laboratory, May 1994.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 6, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990.

[DL95] Nachum Dershowitz and Naomi Lindenstrauss. Abstract and-parallel machines. Technical report, University of Illinois, September 1995.

[DO90] N. Dershowitz and M. Okada. A rationale for conditional equational programming. *Theoretical Computer Science*, 75:111–138, 1990.

[GKL+86] J. A. Goguen, Claude Kirchner, S. Leinwand, J. Meseguer, and T. Winkler. Progress report on the rewrite rule machine. *IEEE Computer Architecture Technical Commitee Newsletter*, pages 7–21, March 1986.

[GKM87] J. A. Goguen, Claude Kirchner, and J. Meseguer. Concurrent term rewriting as a model of computation. In R. Keller and J. Fasel, editors, *Proceedings of Graph Reduction Workshop*, volume 279 of *Lecture Notes in Computer Science*, pages 53–93, Santa Fe (NM, USA), 1987. Springer-Verlag.

[Hin94] C. Hintermeier. How to transform canonical decreasing ctrss into equivalent canonical trss. In *Proceedings of the 4th International Workshop on Conditional Term Rewriting Systems, Jerusalem (Israel)*, June 1994.

[HL91] G. Huet and J.-J. Lévy. Computations in orthogonal rewriting systems, I. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic*, chapter 11, pages 395–414. The MIT press, 1991.

[HP88] B. Hoffmann and D. Plump. Jungle evaluation for efficient term rewriting. In J. Grabowski, P. Lescanne, and W. Wechler, editors, *Proceedings 1st International Workshop on Algebraic and Logic Programming*, number 343 in Lecture Notes in Computer Science. Springer-Verlag, 1988.

[KKV95] Claude Kirchner, Hélène Kirchner, and M. Vittek. Designing constraint logic programming languages using computational systems. In P. Van Hentenryck and V. Saraswat, editors, *Principles and Practice of Constraint Programming. The Newport Papers.*, pages 131–158. The MIT press, 1995.

[KM91] J. W. Klop and A. Middeldorp. Sequentiality in orthogonal term rewriting systems. *Journal of Symbolic Computation*, 12(2):161–196, August 1991.

[KV92] Claude Kirchner and P. Viry. 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, 1992.

[Mes92] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[Mes93] J. Meseguer. A logical theory of concurrent objects and its realisation in the Maude language. In Agha, Wegner, and Yonezawa, editors, *Research Directions in Object-Based Concurrency*. The MIT press, 1993.

[MOM93] N. Martì-Oliet and J. Meseguer. Rewriting logic as a logical and semantical framework. Technical report, SRI International, May 1993.

[PF94] Sara Porat and Nissim Francez. Fairness in term rewriting systems. *Methods of logic in computer science*, pages 141–181, 1994.

[Vir92] Patrick Viry. *La réécriture concurrente*. Thèse de Doctorat d'Université, Université de Nancy 1, October 1992.

[Vir94] Patrick Viry. Rewriting : An effective model of concurrency. In *Proceedings of PARLE'94*, Lecture Notes in Computer Science. Springer-Verlag, 1994.

Unit´e de recherche INRIA Lorraine, Technop ˆole de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rh ˆone-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399