RTA'06, LNCS 4098, pp 19-34. 2006

Computationally Equivalent Elimination of Conditions
- extended abstract -

Traian Florin Şerbănuţă and Grigore Roşu*

Department of Computer Science,
University of Illinois at Urbana-Champaign.

Abstract. An automatic and easy to implement transformation of conditional term rewrite systems into computationally equivalent unconditional term rewrite systems is presented. No special support is needed from the underlying unconditional rewrite engine. Since unconditional rewriting is more amenable to parallelization, our transformation is expected to lead to efficient concurrent implementations of rewriting.

1 Introduction

Conditional rewriting is a crucial paradigm in algebraic specification, since it provides a natural means for executing equational specifications. Many specification languages, including CafeOBJ [8], ELAN [4], Maude [6], OBJ [10], ASF/SDF [21], provide conditional rewrite engines to execute and reason about specifications. It also plays a foundational role in functional logic programming [11]. Conditional rewriting is, however, rather inconvenient to implement directly. To reduce a term, a rewrite engine needs to maintain a *control context* for each conditional rule that is tried. Due to the potential nesting of rule applications, such a control context may grow arbitrarily. The technique presented in this paper automatically translates conditional rewrite rules into unconditional rules, by *encoding the necessary control context into data context*. The obtained rules can be then executed on any unconditional rewrite engine, whose single task is to *match-and-apply* unconditional rules. Such a simplified engine can be seen as a *rewrite virtual machine*, which can be even implemented in hardware, and our transformation technique can be seen as a compiler. One can also simulate the proposed transformation as part of the implementation of a conditional engine.

Experiments performed on three fast rewrite engines, Elan[4], Maude[6] and ASF/SDF [21], show that performance increases can be obtained on current engines if one uses the proposed transformation as a front-end. However, since these rewrite engines may be optimized for conditional rewriting, we expect significant further increases in performance if one just focuses on the much simpler problem of developing optimized *unconditional* rewrite engines and use our technique. Moreover, one can focus on developing *parallel rewrite machines* without worrying about conditions, which obstruct the potential for high parallelism.

* e-mail: {tserban2,grosu}@cs.uiuc.edu

On computational equivalence. Let us formalize the informal notion of "computationally equivalent elimination of conditions". Consider a conditional term rewriting system (CTRS) $\mathcal{R}$ over signature Σ and an (unconditional) term rewriting system (TRS) $\mathcal{R}'$ over signature Σ'. Also, assume some mapping φ from Σ-terms to Σ'-terms and some partial mapping ψ from Σ'-terms to Σ-terms that is an inverse to φ (i.e., $\psi(\varphi(s)) = s$ for any Σ-term s). Σ'-terms $\varphi(s)$ are called *initial*, while terms t' with $\varphi(s) \to^*_{\mathcal{R}'} t'$ are called *reachable* in $\mathcal{R}'$. The (partial) mapping ψ only needs to translate reachable Σ'-terms into Σ-terms. φ can be thought of as translating input terms for $\mathcal{R}$ into input terms for $\mathcal{R}'$, while ψ as taking results of rewritings in $\mathcal{R}'$ into corresponding results for $\mathcal{R}$. In other words, φ and ψ can wrap a Σ' rewrite engine into a Σ rewrite engine. Typically, φ and ψ are straightforward linear translators of syntax.

$\mathcal{R}'$ is *complete* for $\mathcal{R}$ iff any reduction in $\mathcal{R}$ has some corresponding reduction in $\mathcal{R}'$: $s \to^*_{\mathcal{R}} t$ implies $\varphi(s) \to^*_{\mathcal{R}'} \varphi(t)$. Completeness is typically easy to prove but, unfortunately, has a very limited practical use: it only allows to disprove reachability tasks in $\mathcal{R}$ by disproving corresponding tasks in $\mathcal{R}'$. $\mathcal{R}'$ is *sound* for $\mathcal{R}$ iff any reduction in $\mathcal{R}'$ of an initial term corresponds to some reduction in $\mathcal{R}$: $\varphi(s) \to^*_{\mathcal{R}'} t'$ implies $s \to^*_{\mathcal{R}} \psi(t')$. The soundness of $\mathcal{R}'$ allows to compute partial reachability sets in $\mathcal{R}$: applying ψ to all t' reached from $\varphi(s)$ in $\mathcal{R}'$, we get Σ-terms (not necessarily all) reachable from s in $\mathcal{R}$. The soundness and completeness of $\mathcal{R}'$ gives a procedure to *test* reachability in the CTRS $\mathcal{R}$ using any reachability analysis procedure for the TRS $\mathcal{R}'$: $s \to^*_{\mathcal{R}} t$ iff $\varphi(s) \to^*_{\mathcal{R}'} \varphi(t)$.

Soundness and completeness may seem the ideal properties of a transformation. Unfortunately, they do not yield the computational equivalence of the original CTRS to (the wrapping of) the resulting TRS. By *computational equivalence of $\mathcal{R}'$ to $\mathcal{R}$* we mean the following: if $\mathcal{R}$ terminates on a given term s admitting a unique normal form t, then $\mathcal{R}'$ also terminates on $\varphi(s)$ and for any of its normal forms t', we have that $\psi(t') = t$. In other words, the unconditional $\mathcal{R}'$ *can be used transparently* to perform computations for $\mathcal{R}$. Example 3 shows that the soundness and completeness of a transformation do *not* imply computational equivalence, even if the original CTRS is confluent and terminates! Note that termination of $\mathcal{R}$ is *not* required. Indeed, termination of the CTRS may be too restrictive in certain applications, e.g., in functional logic programming [2].

On Termination. Rewriting of a given term in a CTRS may not terminate for two reasons [19]: the reduction of the condition of a rule does not terminate, or there are some rules that can be applied infinitely often on the given term. In rewrite engines, the effect in both situations is the same: the system loops forever or crashes running out of memory. For this reason, we do not make any distinction between the two cases, and simply call a Σ-CTRS *operationally terminating* [13] on Σ-term s iff it always reduces s to a normal form regardless of the order rules apply. Note that this notion is different from *effective termination* [14]; Example 6 shows a system that is confluent and effectively terminating but *not* operationally terminating. Operational termination is based on the assumption that, in general, one cannot expect a rewrite engine to be "smart" enough to pick the right rewrite sequence to satisfy a condition. Formally, a CTRS $\mathcal{R}$ is

RTA'06, LNCS 4098, pp 19-34. 2006

operationally terminating on s if for any t, any proof tree attempting to prove that $s \to^*_{\mathcal{R}} t$ is finite. Operational termination is equivalent to decreasingness for normal CTRSs (and with quasi-decreasingness for deterministic CTRSs) [13].

We give an automatic transformation technique of CTRSs into TRSs, taking ground confluent normal CTRSs $\mathcal{R}$ into computationally equivalent TRSs $\mathcal{R}'$. This technique can be extended to more general CTRSs including ones with extra variables in conditions (see [20]). Experiments show that the resulting TRSs yield performant computational engines for the original CTRSs. On the theoretical side, our main new result is that if $\mathcal{R}$ is finite, ground confluent and operationally terminating on a term s then $\mathcal{R}'$ is ground confluent on reachable terms and terminating on $\varphi(s)$ (Theorem 6). This effectively gives computational equivalence of our transformation for (ground) confluent (finite) systems. To achieve this main result, we prove several other properties: the completeness of our transformation (Theorem 1); ground confluence (Theorem 2) *or* left linearity (Theorem 3) of $\mathcal{R}$ implies the soundness of $\mathcal{R}'$; if $\mathcal{R}$ is left linear, then ground confluence of $\mathcal{R}$ implies ground confluence of $\mathcal{R}'$ on reachable terms (Theorem 4) and operational termination of $\mathcal{R}$ on s implies termination of $\mathcal{R}'$ on $\varphi(s)$ (Theorem 5). Part of these properties recover the power of previous transformations; note however that they are not simple instances of those, due to the particularities of our transformation. Additionally, we show that left linearity and ground confluence of $\mathcal{R}'$ on reachable terms implies ground confluence of $\mathcal{R}$ (Proposition 2), and termination of $\mathcal{R}'$ on reachable terms implies operational termination of $\mathcal{R}$ (Proposition 4); these results potentially enable one to use confluence and/or termination techniques on unconditional TRSs to show confluence and/or operational termination of the original CTRS, but this was not our purpose and consequently have not experimented with this approach.

Section 2 discusses previous transformations of CTRSs into TRSs. We only focus on ones intended to be computationally equivalent and discuss their limitations. Section 3 presents our transformation. Section 4 shows it at work on several examples; some of these examples have been experimented with on the rewrite engines Elan [4], Maude [6] and ASF/SDF [21], with promising performance results. Section 5 lists theoretical results. All proofs can be found in the companion report [20], which will be published elsewhere soon. Section 6 concludes the paper.

2 Previous Transformations

Stimulated by the benefits of transforming CTRSs into equivalent TRSs, there has been much research on this topic. Despite the apparent simplicity of most transformations, they typically work for restricted CTRSs and their correctness, when true, is quite involved. We focus on transformations that generate TRSs intended to be *transparently* used to reduce terms or test reachability in the original CTRSs. Significant efforts have been dedicated to transformations preserving only certain properties, e.g., termination or confluence [18]; we do not

discuss these here. We use the following two examples to illustrate the different transformations and to analyze their limitations.

Example 1. [14, 18]. The CTRS $\mathcal{R}_s$ will be used to test if a transformation is sound and $\mathcal{R}_t$ to test if it preserves termination. Let $\mathcal{R}_s$ be the CTRS

$$\begin{array}{llllll} A \to h(f(a), f(b)) & g(d,x,x) \to B & a \to c & b \to c & c \to e & d \to m \\ h(x,x) \to g(x,x,f(k)) & f(x) \to x \ \mathit{if}\ x \to e & a \to d & b \to d & c \to l & k \to l \\ & & & & & k \to m \end{array}$$

Let $\mathcal{R}_t$ be $\mathcal{R}_s \cup \{B \to A\}$; then $A \not\to^*_{\mathcal{R}_t} B$ and $\mathcal{R}_t$ operationally terminates. □

Example 2. [2]. The two-rule canonical CTRS $\{f(g(x)) \to x \ \mathit{if}\ x \to 0, g(g(x)) \to g(x)\}$ will be used to test whether a transformation preserves confluence. □

Bergstra&Klop. The first CTRS-to-TRS transformation appeared in [3]: start with a rule $Ix \to x$ and to each rule $\rho_i : l \to r \ \mathit{if}\ cl \to cr$ associate rules $\rho'_i : l \to \sigma_i(cl)r$ and $\rho''_i : \sigma_i(cr) \to I$. The transformation is proved to be complete in [3] and claimed to also be sound. Let us apply this transformation on $\mathcal{R}_s$ in Example 1. Rule $f(x) \to x \ \mathit{if}\ x \to e$ is replaced by $f(x) \to \sigma_1(x)x$ and $\sigma_1(e) \to I$, and rule $Ix \to x$ is added. Then:

$$\begin{array}{l} A \to h(f(a), f(b)) \to h(\sigma_1(a)a, f(b)) \to h(\sigma_1(a)d, f(b)) \to h(\sigma_1(c)d, f(b)) \\ \to h(\sigma_1(c)d, \sigma_1(b)b) \to h(\sigma_1(c)d, \sigma_1(b)d) \to h(\sigma_1(c)d, \sigma_1(c)d) \\ \to g(\sigma_1(c)d, \sigma_1(c)d, f(k)) \to g(\sigma_1(e)d, \sigma_1(c)d, f(k)) \to g(Id, \sigma_1(c)d, f(k)) \\ \to g(d, \sigma_1(c)d, f(k)) \to g(d, \sigma_1(l)d, f(k)) \to g(d, \sigma_1(l)m, f(k)) \\ \to g(d, \sigma_1(l)m, \sigma_1(k)k) \to g(d, \sigma_1(l)m, \sigma_1(l)k) \to g(d, \sigma_1(l)m, \sigma_1(l)m) \to B \end{array}$$

So this transformation is *not sound*. Transforming $\mathcal{R}_t$, we can see that this transformation does *not preserve termination*, because $A \to^+ A$. For the system in Example 2, $f(g(x)) \to x \ \mathit{if}\ x \to 0$ is replaced by $f(g(x)) \to \sigma_1(x)x$ and $\sigma_1(0) \to I$, so $f(g(g(0))) \to f(g(0)) \to \sigma_1(0)0 \to I0 \to 0$ and $f(g(g(0))) \to \sigma_1(g(0))g(0)$, both of them normal forms. Thus the resulting TRS is *not confluent*. Consequently, this transformation does not produce computationally equivalent TRSs.

Giovanetti&Moiso. The transformation in [9] (suggested in [7]) replaces each rule $\rho_i : l \to r \ \mathit{if}\ cl \to cr$ by $l \to \mathit{if}_i(\mathit{Var}(l), cl)$ and $\mathit{if}_i(\mathit{Var}(l), cr) \to r$. However, this transformation is complete and computationally equivalent only when the original CTRS is *safely transformable* [9], that is, has no superposition, is simply terminating, and is non-overlapping on conditions. The "safely transformable" CTRSs are too restrictive; our transformation yields computationally equivalent TRSs imposing only ground confluence (safely transformable CTRSs are ground confluent) on the original CTRS.

Hintermeier [12] proposes a technique where an "interpreter" for a CTRS is specified using unconditional rewrite rules, defining the detailed steps of the application of a conditional rewrite rule including rewrite-based implementations of matching and substitution application. Although this is an interesting result, it has little practical relevance (the "meta" stepwise simulation of a rewrite system using another rewrite system leads to dramatic performance loss).

Marchiori's Unravellings. An abstract notion of transformation, called *unravelling*, and several concrete instances of it, were introduced by Marchiori in [14];

these were further studied in [15, 17, 18, 16]. An *unraveling* is a computable map U from CTRSs to TRSs over the same signature, except a special operation U_ρ for each rule ρ, such that $\downarrow_R \subseteq \downarrow_{U(R)}$ ($\downarrow$ stands for "join", i.e., $\rightarrow;\leftarrow$) and $U(T \cup R) = T \cup U(R)$ if T is a TRS. The concrete instance transformations are similar to that in [9]: each conditional rule $\rho : l \rightarrow r$ *if* $cl \rightarrow cr$ is replaced by its unravelling, rules $l \rightarrow U_\rho(cl, \mathit{Var}(r))$ and $U_\rho(cr, \mathit{Var}(r)) \rightarrow r$. Completeness holds, but soundness does not hold without auxiliary hypotheses [14] (see Example 1) such as left linearity [14, 18]. Also, (quasi) decreasingness and left linearity of the CTRS imply termination of the corresponding TRS.

Example 3. A sound and complete transformation does not necessarily yield computational equivalence even if the original CTRS is canonical. The unravelling of the system in Example 2 is $\{f(g(x)) \rightarrow U_1(x,x), U_1(0,x) \rightarrow x, g(g(x)) \rightarrow g(x)\}$. The original CTRS is left linear, so the unravelling is sound and complete, but is *not* computationally equivalent: $f(g(g(0)))$ reduces to $U_1(g(0), g(0))$, a normal form with no correspondent normal form in the original CTRS. □

Unfortunately, no unravelling preserves confluence or termination [14] (i.e., for any unravelling U, there are confluent and/or terminating CTRSs R such that $U(R)$ is *not* a confluent and/or terminating TRS), thus they do not yield computationally equivalent TRSs. Therefore, it is not surprising that the more recent transformations discussed next that aim at computational equivalence, including ours, are *not* unravellings (they modify the original signature).

Viry. The transformation in [22] (inspired from [1]) inspired all subsequent approaches. It modifies the signature by adding to each operation as many arguments as conditional rules having it at the top of their lhs. Two unconditional rules replace each conditional rule, one for initializing the auxiliary arguments and the other for the actual rewrite step. Formally: let $\rho_{\sigma,i}$ denote the ith rule whose lhs is topped in σ; add as many arguments to σ as the number of rules $\rho_{\sigma,i}$; let $c_{\sigma,i}$ be the number $\mathit{arity}(\sigma) + i$, corresponding to the i^{th} auxiliary argument added to σ; transform each rule $\rho_{\sigma,i} : l \rightarrow r$ *if* $cl \rightarrow cr$ into

$$\rho'_{\sigma,i} : \widetilde{l}[c_{\sigma,i} \leftarrow \bot] \rightarrow \widetilde{l}[c_{\sigma,i} \leftarrow [\overline{cl}, \mathit{Var}(l)]] \text{ and } \rho''_{\sigma,i} : l^*[c_{\sigma,i} \leftarrow [cr, \mathit{Var}(l)]] \rightarrow \overline{r},$$

where "$\bot$" is a special constant stating that the corresponding conditional rule has not been tried yet on the current position, $\overline{s}$ lifts a term by setting all new arguments to $\bot$, $\widetilde{s}$ lifts a term with fresh variables on the new arguments, and $s^* = s$ replaces all variables in $\widetilde{s}$ with fresh variables. Structures $[u, \overrightarrow{s}]$ comprise the reduction status of conditions (u) together with corresponding substitutions ($\overrightarrow{s}$) when they were started. The substitution is used to correctly initiate the reduction of the rhs of the original conditional rule. Viry proved in [22] his transformation sound and complete and that it preserves termination. We believe the completeness indeed holds, but have counterexamples for the other properties. Let us transform the CTRS $\mathcal{R}_s$ from Example 1. First, rules $h(x,x) \rightarrow g(x,x,f(k))$ and $g(d,x,x) \rightarrow B$ are replaced by $h(x,y) \rightarrow g(x,x,f(k))$ *if* $\mathit{eq}(x,y) \rightarrow \mathit{true}$ and $g(d,x,y) \rightarrow B$ *if* $\mathit{eq}(x,y) \rightarrow \mathit{true}$ to resolve non-left linearity, where $\mathit{eq}(x,x) \rightarrow \mathit{true}$ is the only non-left linear rule allowed [22]. Then these conditional rules and $f(x) \rightarrow x$ *if* $x \rightarrow e$ and $A \rightarrow h(f(a), f(b))$ are transformed into:

$$\begin{array}{ll} f(x,\bot) \to f(x,[x,x]) & h(x,y,\bot) \to h(x,y,[eq(x,y),x,y] \\ f(y,[e,x]) \to x & h(x',y',[true,x,y]) \to g(x,x,f(k,\bot),\bot) \\ g(d,x',y',[true,x,y]) \to B & g(d,x,y,\bot) \to g(d,x,y,[eq(x,y),x,y]) \end{array}$$
$$A \to h(f(a,\bot),f(b,\bot),\bot)$$

The following is then a valid sequence in the generated unconditional TRS:

$A \to h(f(a,\bot),f(b,\bot),\bot) \to h(f(a,[a,a]),f(b,\bot),\bot) \to h(f(d,[a,a]),$
$f(b,\bot),\bot) \to h(f(d,[c,a]),f(b,\bot),\bot) \to h(f(d,[c,c]),f(b,\bot),\bot) \to h(f(d,[c,c]),$
$f(b,[b,b]),\bot) \to h(f(d,[c,c]),f(d,[b,b]),\bot) \to h(f(d,[c,c]),f(d,[c,b]),\bot) \to$
$h(f(d,[c,c]),f(d,[c,c]),\bot) \to h(f(d,[c,c]),f(d,[c,c]),[eq(f(d,[c,c]),f(d,[c,c])),$
$f(d,[c,c]),f(d,[c,c])]) \to h(f(d,[c,c]),f(d,[c,c]),[true,f(d,[c,c]),f(d,[c,c])]) \to$
$g(f(d,[c,c]),f(d,[c,c]),f(k,\bot),\bot) \to g(f(d,[e,c]),f(d,[c,c]),f(k,\bot),\bot) \to$
$g(f(d,[e,e]),f(d,[c,c]),f(k,\bot),\bot) \to g(d,f(d,[c,c]),f(k,\bot),\bot) \to$
$g(d,f(m,[c,c]),f(k,\bot),\bot) \to g(d,f(m,[l,c]),f(k,\bot),\bot) \to g(d,f(m,[l,l]),$
$f(k,\bot),\bot) \to g(d,f(m,[l,l]),f(k,[k,k]),\bot) \to g(d,f(m,[l,l]),f(m,[k,k]),\bot)$
$\to g(d,f(m,[l,l]),f(m,[l,k]),\bot) \to g(d,f(m,[l,l]),f(m,[l,l]),\bot) \to$
$g(d,f(m,[l,l]),f(m,[l,l]),[eq(f(m,[l,l]),f(m,[l,l])),f(m,[l,l]),f(m,[l,l])]) \to$
$g(d,f(m,[l,l]),f(m,[l,l]),[true,f(m,[l,l]),f(m,[l,l])]) \to B$

Hence, Viry's transformation is *not sound*. Using $\mathcal{R}_t$ instead of $\mathcal{R}_s$, whose corresponding TRS just adds rule $B \to A$ to that of $\mathcal{R}_s$, we can notice that it *does not preserve termination* either. Let us now transform the CTRS in Example 2 to $\{f(g(x),\bot) \to f(g(x),[x,x]), f(x,[0,y]) \to y, g(g(x)) \to g(x)\}$; note that $\mathcal{R}'$ is *not confluent* [2] (with or without Viry's conditional eagerness [22]): $f(g(g(0)),\bot)$ can be reduced to both 0 and $f(g(0),[g(0),(g(0))])$. Therefore, this transformation does not fulfill the requirements of computational equivalence.

Antoy,Brassel&Hanus proposed in [2] a simple fix to Viry's technique, namely to restrict the input CTRSs to *constructor-based* (i.e., the lhs of each rule is a term of the form $f(t_1,\ldots,t_n)$, where f is *defined* and $t_1,\ldots,t_n$ are all *constructor* terms) and *left linear* ones. Under these restrictions, they also show that the substitution needed by Viry's transformation is not necessary anymore, so they drop it and prove that the new transformation is sound and complete; moreover, if the original CTRS is additionally weakly orthogonal, then the resulting TRS is confluent on reachable terms. It is suggested in [2] that what Viry's transformation (or their optimized version of it) needs to generate computationally equivalent TRSs is to reduce its applicability to only constructor-based, weakly orthogonal and left linear CTRSs. While constructor-baseness and left linearity are common to functional logic programming and are easy to check automatically, we believe that they are, in general, an unnecessarily strong restriction on the input CTRS, which may make the translation unusable in many situations of practical interest (see, e.g., the bubble-sort algorithm in Section 4).

Roşu. The transformation in [19] is defined for join CTRSs and requires the rewrite engine to support some simple *contextual rewriting strategies*, namely an *if*(_,_,_) eager on the condition and an *equal?* eager on both arguments. As in Viry's transformation, additional arguments are added to each operation σ for each conditional rule $\rho_{\sigma,i}$, but they only need to keep truth values. The distinctive feature of this transformation is the introduction of the

$\{\_\}$ operation, which allows the rewriting process to continue after a condition got stuck provided changes occur in subterms. Each conditional rule $\rho_{\sigma,i}$: $l \to r$ *if* $cl \downarrow cr$ is encoded by one unconditional rule $\overline{\rho}_{\sigma,i}$: $\widetilde{l}[c_{\sigma,i} \leftarrow \mathit{true}] \to \mathit{if}(\mathit{equal?}(\{\overline{cl}\}, \{\overline{cr}\}), \{\overline{r}\}, \widetilde{l}[c_{\sigma,i} \leftarrow \mathit{false}])$. The bracket clears the failed conditions on the path to the top: $\sigma(x_1, .., \{x_i\}, .., x_{\mathit{arity}(\sigma)}, y_1, .., y_m) \to \{\sigma(x_1, .., x_i, .., x_{\mathit{arity}(\sigma)}, \mathit{true}, .., \mathit{true})\}$. It is shown in [19] that this transformation is sound and that operational termination is preserved and implies completeness and preservation of ground confluence, that is, computational equivalence. Left linearity needs *not* be assumed. Although most modern rewrite systems support the rewrite strategies required by the transformation in [19], we argue that imposing restrictions on the order of evaluation makes a rewrite engine less friendly w.r.t parallelism and more complex; in some sense, contextual strategies can be seen as some sort of conditional rules: apply the rule *if* the context permits.

Our transformation basically integrates Rosu's $\{\_\}$ operation within Viry's transformation, which allows us to also eliminate the need to carry a substitution. We recently found out[1] that a related approach was followed by Brassel in his master thesis [5], but we can't relate our results since we were unable to obtain an English translation of his results.

3 Our Transformation

Like in the last three transformations above, auxiliary arguments are added to some operators to maintain the control context information. For simplicity, we here discuss only the transformation of *normal* CTRSs, that is, ones whose conditional rules have the form $l \to r$ *if* $cl \to cr$, cr is a constant in normal form and all variables from cl and r also occur in l (and l is not a variable). In [20] we discuss extensions of our technique to more complex cases, including ones with extra variables and matching in conditions. Let $\mathcal{R}$ be any Σ-CTRS. A *σ-conditional rule* [22] is a conditional rule with σ at the top of its lhs, i.e., one of the form $\sigma(t_1, \ldots, t_n) \to r$ *if* $cl \to cr$. Let k_σ be the number of σ-conditional rules and let $\rho_{\sigma,i}$ denote the i^{th} σ-conditional rule in $\mathcal{R}$.

The signature transformation. Let $\overline{\Sigma}$ be the signature containing: a fresh constant $\bot$; a fresh unary operator $\{\_\}$; for any $\sigma \in \Sigma_n$ (i.e., $\sigma \in \Sigma$ has n arguments), an operation $\overline{\sigma} \in \overline{\Sigma}_{n+k_\sigma}$ (the additional k_σ arguments of $\overline{\sigma}$ are written to the right of the other n arguments). An important step in our transformation is to replace Σ-terms by corresponding $\overline{\Sigma}$-terms. The reason for the additional arguments is to pass the control context (due to conditional rules) into data context: the additional i-th argument of $\overline{\sigma}$ at some position in a term maintains the status of appliance of $\rho_{\sigma,i}$; if $\bot$ then that rule was not tried, otherwise the condition is being under evaluation or is already evaluated. Thus, the corresponding $\overline{\Sigma}$-term of a Σ-term is obtained by replacing each operator σ by $\overline{\sigma}$ with the k_σ additional arguments all $\bot$. Formally, let $\mathcal{X}$ be an infinite set of *variables* and let $\bar{\cdot} : T_\Sigma(\mathcal{X}) \to T_{\overline{\Sigma}}(\mathcal{X})$ be defined inductively as: $\overline{x} = x$ for

[1] From a private communication with Bernd Brassel.

any $x \in \mathcal{X}$ and $\overline{\sigma(t_1,\ldots,t_n)} = \overline{\sigma}(\overline{t_1},\ldots,\overline{t_n},\bot,\ldots,\bot)$ for any $\sigma \in \Sigma_n$ and any $t_1,\ldots,t_n \in T_\Sigma(\mathcal{X})$. Let us define another map, $\widetilde{\cdot}^X : T_\Sigma(X) \to T_{\overline{\Sigma}}(\mathcal{X})$, this time indexed by a *finite* set of variables $X \subseteq \mathcal{X}$, as $\widetilde{x}^X = x$ for any $x \in X$, and as $\widetilde{\sigma(t_1,\ldots,t_n)}^X = \overline{\sigma}(\widetilde{t_1}^X,\ldots,\widetilde{t_n}^X,z_1,\ldots,z_{k_\sigma})$ for any $\sigma \in \Sigma_n$ and $t_1,\ldots,t_n \in T_\Sigma(X)$, where $z_1,\ldots,z_{k_\sigma} \in \mathcal{X} - X$ are some arbitrary but fixed different fresh variables that do not occur in X or in $\widetilde{t_1}^X,\ldots,\widetilde{t_n}^X$. Therefore, $\widetilde{t}^X$ transforms the Σ-term t into a $\overline{\Sigma}$-term by replacing each operation $\sigma \in \Sigma$ by $\overline{\sigma} \in \overline{\Sigma}$ and adding some distinct fresh variables for the additional arguments, chosen arbitrarily but deterministically.

The rewrite rules transformation. Given a Σ-CTRS $\mathcal{R}$, let $\overline{\mathcal{R}}$ be the $\overline{\Sigma}$-TRS obtained as follows. For each σ-conditional rule $\rho_{\sigma,i}$: $l \to r$ *if* $cl \to cr$ over variables X in $\mathcal{R}$, add to $\overline{\mathcal{R}}$ two rules, namely $\overline{\rho}_{\sigma,i} : \widetilde{l}^X[c_{\sigma,i} \leftarrow \bot] \to \widetilde{l}^X[c_{\sigma,i} \leftarrow \{\overline{cl}\}]$ and $\overline{\rho}'_{\sigma,i} : \widetilde{l}^X[c_{\sigma,i} \leftarrow \{cr\}] \to \{\overline{r}\}$, where $c_{\sigma,i}$ is the number $\mathit{arity}(\sigma) + i$ corresponding to the i^{th} conditional argument of σ. For each unconditional rule $l \to r$ in $\mathcal{R}$, add rule $\widetilde{l}^X \to \{\overline{r}\}$ to $\overline{\mathcal{R}}$. For each $\sigma \in \Sigma_n$ and each $1 \le i \le n$, add to $\overline{\mathcal{R}}$ a rule $\overline{\sigma}(x_1,..,x_{i-1},\{x_i\},x_{i+1},..,x_n,z_1,..,z_{k_\sigma}) \to \{\overline{\sigma}(x_1,..,x_{i-1},x_i,x_{i+1},..,x_n,\bot,..,\bot)\}$, intuitively stating that a condition tried and potentially failed in the past at some position may hold once an immediate subterm changes; the operation $\{\_\}$, symbolizing the change, also needs to be propagated bottom-up, reseting the other started conditions to $\bot$. The applicability information of an operation can be updated from several of its subterms; to keep this operation idempotent, we add $\{\{x\}\} \to \{x\}$ to $\overline{\mathcal{R}}$. The size of $\overline{\mathcal{R}}$ is $1 + u + 2 \times c + \sum_{n \ge 0} n \times |\Sigma_n|$, where u is the number of unconditional rewrite rules and c is the number of conditional rewrite rules in $\mathcal{R}$.

4 Examples and Experiments

We next illustrate our transformation on several examples.
Confluence is preserved. Let us transform the CTRS in Example 2:

$$\begin{array}{lll} \overline{f}(\overline{g}(x),\bot) \to \overline{f}(\overline{g}(x),\{x\}) & \overline{f}(\overline{g}(x),\{\overline{0}\}) \to \{x\} & \overline{g}(\overline{g}(x)) \to \{\overline{g}(x)\} \\ \overline{g}(\{x\}) \to \{\overline{g}(x)\} & \overline{f}(\{x\},b) \to \{\overline{f}(x,\bot)\} & \{\{x\}\} \to \{x\} \end{array}$$

The problem that appeared in Viry's transformation is avoided in our transformation by the rules of $\{\cdot\}$, which allow the evaluation of a condition to be restarted at the top of a term once a modification occurs in a subterm. Thus, given the $\overline{\Sigma}$-term $\{\overline{f}(\overline{g}(\overline{g}(\overline{0})),\bot)\}$, even if a rewrite engine first tries to evaluate the condition at the top, a "correct" rewriting sequence is eventually obtained: $\{\overline{f}(\overline{g}(\overline{g}(\overline{0})),\bot)\} \to_{\overline{\mathcal{R}}} \{\overline{f}(\overline{g}(\overline{g}(\overline{0})),\{\overline{g}(\overline{0})\})\} \to_{\overline{\mathcal{R}}} \{\overline{f}(\{\overline{g}(\overline{0})\},\{\overline{g}(\overline{0})\})\} \to_{\overline{\mathcal{R}}} \{\{\overline{f}(\overline{g}(\overline{0}),\bot)\}\}$, and now the condition can be tried again and this time will succeed.

Odd/Even [19]. Let us consider natural numbers with 0 and successor s, constants *true* and *false* and the following on purpose inefficient conditional rules defining *odd* and *even* operators on natural numbers (here denoted as o and e):

$$\begin{array}{lll} o(0) \to \mathit{false} & o(s(x)) \to \mathit{true}\ \mathit{if}\ e(x) \to \mathit{true} & o(s(x)) \to \mathit{false}\ \mathit{if}\ e(x) \to \mathit{false} \\ e(0) \to \mathit{true} & e(s(x)) \to \mathit{true}\ \mathit{if}\ o(x) \to \mathit{true} & e(s(x)) \to \mathit{false}\ \mathit{if}\ o(x) \to \mathit{false} \end{array}$$

In order to check whether a natural number n, i.e., a term consisting of n successor operations applied to 0, is odd, a conditional rewrite engine may need $\mathcal{O}(2^n)$ rewrites in the worst case. Indeed, if $n > 0$ then either the second or the third rule of *odd* can be applied at the first step; however, in order to apply any of those rules one needs to reduce the even of the predecessor of n, twice. Iteratively, the evaluation of each even involves the reduction of two odds, and so on. Moreover, the rewrite engine needs to maintain a control context data-structure, storing the status of the application of each (nested) rule that is being tried in a reduction. It is the information stored in this control context that allows the rewriting engine to backtrack and find an appropriate rewriting sequence. As shown at the end of this section, some rewrite engines perform quite poorly on this system. Let us apply it our transformation. Since there are two *odd*-conditional rules and two *even*-conditional rules, each of these operators will be enriched with two arguments. The new TRS is (for aesthetic reasons we overline only those operations that change; z_1 and z_2 are variables):

$$\begin{array}{ll}
\overline{o}(0, z_1, z_2) \to \{\mathit{false}\} & \overline{e}(0, z_1, z_2) \to \{\mathit{true}\} \\
\overline{o}(s(x), \{\mathit{false}\}, z_2) \to \{\mathit{false}\} & \overline{e}(s(x), \{\mathit{false}\}, z_2) \to \{\mathit{false}\} \\
\overline{o}(s(x), z_1, \{\mathit{true}\}) \to \{\mathit{true}\} & \overline{e}(s(x), z_1, \{\mathit{true}\}) \to \{\mathit{true}\} \\
\overline{o}(s(x), \bot, z_2) \to \overline{o}(s(x), \{\overline{e}(x, \bot, \bot)\}, z_2) & \overline{e}(s(x), \bot, z_2) \to \overline{e}(s(x), \{\overline{o}(x, \bot, \bot)\}, z_2) \\
\overline{o}(s(x), z_1, \bot) \to \overline{o}(s(x), z_1, \{\overline{e}(x, \bot, \bot)\}) & \overline{e}(s(x), z_1, \bot) \to \overline{e}(s(x), z_1, \{\overline{o}(x, \bot, \bot)\}) \\
s(\{x\}) \to \{s(x)\} & \overline{o}(\{x\}, z_1, z_2) \to \{\overline{o}(x, z_1, z_2)\} \\
\{\{x\}\} \to \{x\} & \overline{e}(\{x\}, z_1, z_2) \to \{\overline{e}(x, z_1, z_2)\}
\end{array}$$

If one wants to test whether a number n, i.e., n consecutive applications of successor on 0, is odd, one should reduce the term $\{o(n, \bot, \bot)\}$.

Bubble sort. The following one-rule CTRS sorts lists of numbers (we assume appropriate rules for numbers) implementing the bubble sort algorithm:

$$\cdot(x, \cdot(y, l)) \to \cdot(y, \cdot(x, l)) \ \mathit{if}\ x < y \to \mathit{true}$$

This CTRS is ground confluent but *not* constructor-based. Its translation is:

$$\begin{array}{ll}
\cdot(x, \cdot(y, l, c), \bot) \to \cdot(x, \cdot(y, l, c), \{x > y\}) & \{\{l\}\} \to \{l\} \\
\cdot(x, \cdot(y, l, c), \{\mathit{true}\}) \to \{\cdot(y, \cdot(x, l, \bot), \bot)\} & \cdot(x, \{l\}, c) \to \{\cdot(x, l, \bot)\}
\end{array}$$

Experiments. Our major motivation to translate a CTRS into a computationally equivalent TRS that can run on any unrestricted unconditional rewrite engine was the potential to devise highly parallelizable rewrite engines. It was therefore an unexpected and a pleasant surprise to note that our transformation can actually bring immediate benefits if implemented as a front-end to *existing*, non-parallel rewrite engines. Note, however, that current rewrite engines are optimized for *both* conditional and unconditional rewriting; an engine optimized for just unconditional rewriting could probably be even more efficient.

We next give some numbers regarding the speed of the generated TRS. We used Elan and Maude as interpreters and ASF/SDF as a compiler - our goal was *not* to compare rewrite engines (that's why we did not use same input data for all engines) but rather to compare how our transformation performs on each of them. Besides the two examples presented above (Odd/Even and Bubble sort), we tested our transformation on two other CTRSs: Quotient/Reminder (inspired from [18]) and the evaluation of a program generating all permutations of n elements and counting them written for a rewriting based interpreter of a

simple programming language with arrays (using both matching in conditions and rewriting modulo axioms - see [20] for a discussion of when our transformation is sound for rewriting modulo axioms). We have tested how long it took for a term to be rewritten to a normal form. In the table below, Cond shows the results using the original system, Ucond those using the presented transformation and Ucond* the transformation enhanced with some simple but practical optimizations described below. Times were obtained on a machine with 2 GHz Pentium 4 CPU and 1GB RAM.

Odd/Even								
Elan - odd(18)			Maude - odd(24)			ASF/SDF - odd(25)		
Cond	Uncond	Uncond*	Cond	Uncond	Uncond*	Cond	Uncond	Uncond*
85.79s	5.55s	∼0s	84.97s	17.05s	∼0s	0.02s	7.46s	0.01s

Bubble Sort					
Elan 100		Maude 5000		ASF/SDF 5000	
Cond	Uncond(*)	Cond	Uncond(*)	Cond	Uncond(*)
28.19s	3.46s	72.34s	43.53s	81.64s	85.71s

Quotient(Reminder)								
Elan $10^5/6$			Maude $10^7/2$			ASF/SDF $10^6/2$		
Cond	Uncond	Uncond*	Cond	Uncond	Uncond*	Cond	Uncond	Uncond*
10.85s	5.82s	5.23s	75.98s	67.59s	41.61s	13.96s	15.28s	14.53s

Rew.-based interpreter of a simple PL with arrays - *permutation generation*							
Maude				ASF/SDF			
8		9		8		9	
Cond	Uncond*	Cond	Uncond*	Cond	Uncond*	Cond	Uncond*
18.06s	12.76s	-[1]	144.56s	5.72s	14.20s	49.56s	-[1]

The optimized transformation always outperformed the original CTRS in our experiments on rewrite engine interpretors. We cannot say what exactly made it slower on some tests performed on ASF/SDF - it might be because the compiler is aimed to be efficiently executed on sequential machines; we actually expect our transformation to perform significantly better on parallel rewrite engines. Note that on the odd/even example, ASF/SDF already performs condition elimination as a stage of its compilation process.

A simple and very practical optimization is as follows: if two or more σ-conditional rules have the same lhs and their conditions also have the same lhs, then we can add only one auxiliary argument to σ in $\overline{\sigma}$ for all of these and only one rule in the TRS for starting the condition. With this, e.g., the optimized TRS generated for the odd/even CTRS is:

$\overline{o}(0, z_1) \to \{\mathit{false}\}$ $\quad \overline{o}(s(x), \{\mathit{false}\}) \to \{\mathit{false}\}$ $\quad \overline{o}(s(x), \bot) \to \overline{o}(s(x), \{\overline{e}(x, \bot)\})$
$\overline{e}(0, z_1) \to \{\mathit{true}\}$ $\quad \overline{o}(s(x), \{\mathit{true}\}) \to \{\mathit{true}\}$ $\quad \overline{e}(s(x), \bot) \to \overline{e}(s(x), \{\overline{o}(x, \bot)\})$
$\{\{x\}\} \to \{x\}$ $\quad \overline{e}(s(x), \{\mathit{false}\}) \to \{\mathit{false}\}$ $\quad \overline{o}(\{x\}, z_1) \to \{\overline{o}(x, z_1)\}$
$s(\{x\}) \to \{s(x)\}$ $\quad \overline{e}(s(x), \{\mathit{true}\}) \to \{\mathit{true}\}$ $\quad \overline{e}(\{x\}, z_1) \to \{\overline{e}(x, z_1)\}$

Another optimization easy to perform statically is to restrict the number of additional conditional arguments added to each operation σ to the maximum number of overlapping rules whose lhs is rooted in σ. The intuition here is that if two conditional rules are orthogonal, their conditions can't be started at the same time for the same term. For orthogonal systems (as our language definition), for example, this means adding at most one argument per operation.

[1] the machine ran out of memory while attempting to reduce the term

5 Theoretical Aspects

We use the terminology in [18] and, as mentioned, here only consider *normal* CTRSs. Before we formalize the relationship between CTRSs and their unconditional variants, we define and discuss several classes of $\overline{\Sigma}$-terms that will be used in the sequel. First, let $\widehat{\cdot} : T_{\overline{\Sigma}}(X) \to T_{\Sigma}(X)$ be a *partial* map, forgetting all the auxiliary arguments of operations, defined as: $\widehat{x} = x$ for any variable x, $\widehat{\{t'\}} = \widehat{t'}$ and $\widehat{\overline{\sigma}(t'_1,..,t'_n,z_1,..,z_{k_\sigma})} = \sigma(\widehat{t'_1},..\widehat{t'_n})$. In particular, $\widehat{\overline{t}} = t$. The map $\widehat{\cdot}$ is partial (not defined for $\overline{\Sigma}$-terms such as, e.g., $\bot$). A $\overline{\Sigma}$-term t' is *structural* iff $\widehat{t'}$ is defined, that is, if it is "resembling" a Σ-term. Note that the lhs and rhs of any (unconditional) rule in $\overline{\mathcal{R}}$ are structural.

A position α is a string of numbers representing a path in a term seen as a tree. Let us define two mutually recursive important types of positions in structural $\overline{\Sigma}$ terms. A position α is *structural* for t' iff α has no conditional position as a prefix. A position α is *conditional* for t' iff $\alpha = \alpha' c_{\sigma,i}$ such that α' is structural for t' and $t'_{\alpha'} = \overline{\sigma}(\overrightarrow{u})$ (recall $c_{\sigma,i}$ is associated to $\rho_{\sigma,i}$'s conditions).

A rewriting step $s' \to_{\overline{\mathcal{R}}} t'$ is *structural* iff it occurs at a structural position in s' and either uses a rule of form $\overline{\rho}'_{\sigma,i}$ or one corresponding to an unconditional rule in $\mathcal{R}$. A ground $\overline{\Sigma}$-term t' is *reachable* iff there is some ground Σ-term t such that $\{\overline{t}\} \to^*_{\overline{\mathcal{R}}} \{t'\}$. The set of all conditions started for a reachable term s', written $cond(s')$, is defined as $\bigcup_C(\{s'|_\alpha\} \cup cond(s'|_\alpha))$ where C is the set of conditional positions α in s' such that $s'|_\alpha \neq \bot$.

Proposition 1. *(1) Any subterm of a structural term on a structural position is also structural; (2) If t' is a structural term with variables on structural positions and θ is a substitution giving structural terms for variables of t' then $\theta(t')$ is also structural; (3) Structural terms are closed under $\overline{\mathcal{R}}$; (4) Any reachable term is also structural; (5) Reachable terms are closed under $\overline{\mathcal{R}}$.*

Completeness means that rewriting that can be executed in the original CTRS can also be simulated on the corresponding TRS. We show that for any Σ-term s, "everything that can be done on s in $\mathcal{R}$ can also be done on $\{\overline{s}\}$ in $\overline{\mathcal{R}}$".

Theorem 1. *If $s \to^k_{\mathcal{R}} t$ then $\{\overline{s}\} \to_{\overline{\mathcal{R}}} \{\overline{t}\}$ with k structural steps.*

Although it may not seem so, $\{\overline{s}\} \to^*_{\overline{\mathcal{R}}} \{\overline{t}\}$ does not generally imply that $s \to^*_{\mathcal{R}} t$:

Example 4. Consider the transformation for $\mathcal{R}_s$ from Example 1:

$$
\begin{array}{lll}
A \to \{h(f(a,\bot),f(b,\bot))\} & f(\{x\},y) \to \{f(x,\bot)\} & a \to \{c\} \\
h(x,x) \to \{g(x,x,f(k,\bot))\} & h(\{x\},y) \to \{h(x,y)\} & a \to \{d\} \quad k \to \{l\} \\
g(d,x,x) \to \{B\} & h(x,\{y\}) \to \{h(x,y)\} & b \to \{c\} \quad k \to \{m\} \\
f(x,\bot) \to f(x,\{x\}) & g(\{x\},y,z) \to \{g(x,y,z)\} & b \to \{d\} \quad d \to \{m\} \\
f(x,\{e\}) \to \{x\} & g(x,\{y\},z) \to \{g(x,y,z)\} & c \to \{e\} \\
\{\{x\}\} \to \{x\} & g(x,y,\{z\}) \to \{g(x,y,z)\} & c \to \{l\}
\end{array}
$$

Then the following rewrite sequence can be obtained in $\overline{\mathcal{R}}_s$:

$\{A\} \to^*_{\overline{\mathcal{R}}} \{h(f(a,\bot),f(b,\bot))\} \to^*_{\overline{\mathcal{R}}} \{h(f(\{d\},\{c\}),f(b,\bot))\}$
$\to^*_{\overline{\mathcal{R}}} \{h(f(\{d\},\{c\}),f(\{d\},\{c\}))\} \to^*_{\overline{\mathcal{R}}} \{g(f(\{d\},\{c\}),f(\{d\},\{c\}),f(k,\bot))\}$
$\to^*_{\overline{\mathcal{R}}} \{g(f(\{d\},\{e\}),f(\{d\},\{c\}),f(k,\bot))\} \to^*_{\overline{\mathcal{R}}} \{g(d,f(\{d\},\{c\}),f(k,\bot))\}$
$\to^*_{\overline{\mathcal{R}}} \{g(d,f(\{m\},\{l\}),f(k,\bot))\} \to^*_{\overline{\mathcal{R}}} \{g(d,f(\{m\},\{l\}),f(\{m\},\{l\}))\} \to^*_{\overline{\mathcal{R}}} \{B\}$,

but it is not the case that $A \to^*_{\mathcal{R}_s} B$. □

Even though Theorem 1 is too weak to give us a procedure in $\overline{\mathcal{R}}$ to test reachability in $\mathcal{R}$, it still gives us a technique to test whether a term t is *not* reachable from a term s in $\mathcal{R}$: if it is *not true* that $\{\overline{s}\} \to^*_{\overline{\mathcal{R}}} \{\overline{t}\}$ then it is also *not true* that $s \to^*_{\mathcal{R}} t$. Of course, in order for this to be mechanizable, the set of terms reachable from $\{\overline{s}\}$ must be finite. This does not give us much, but it is the most we can get without additional restrictions on $\mathcal{R}$.

Soundness means that rewrites $\{\overline{s}\} \to^*_{\overline{\mathcal{R}}} t'$ in $\overline{\mathcal{R}}$ correspond to rewrites $s \to^*_{\mathcal{R}} \widehat{t'}$ in $\mathcal{R}$. Unfortunately, as shown by Example 4, soundness does not hold without restricting $\mathcal{R}$. We show that ground confluence *or* left linearity suffices.

Theorem 2. *If $\mathcal{R}$ is ground confluent and s' is reachable such that $s' \to^*_{\overline{\mathcal{R}}} t'$ in k structural steps, then $\widehat{s'} \to^k \widehat{t'}$. In particular, our transformation is sound.*

The claim above may not hold if the original CTRS is not ground confluent:

Example 5. Consider the following CTRS and its corresponding TRS:

$$(\mathcal{R}) \begin{cases} a \to \mathit{true} \\ a \to \mathit{false} \\ f(x) \to \mathit{true} \text{ if } x \to \mathit{true} \end{cases} \qquad (\overline{\mathcal{R}}) \begin{cases} \overline{a} \to \{\overline{\mathit{true}}\} & \overline{a} \to \{\overline{\mathit{false}}\} \\ \overline{f}(x, \bot) \to \overline{f}(x, \{x\}) & \overline{f}(x, \{\overline{\mathit{true}}\}) \to \{\overline{\mathit{true}}\} \\ \overline{f}(\{x\}, y) \to \{\overline{f}(x, \bot)\} & \{\{x\}\} \to \{x\} \end{cases}$$

The following sequence is valid in $\overline{\mathcal{R}}$, but it is *not* the case that $f(\mathit{false}) \to_{\mathcal{R}} \mathit{true}$:

$$\{\overline{f}(\overline{a}, \bot)\} \to_{\overline{\mathcal{R}}} \{\overline{f}(\overline{a}, \{\overline{a}\})\} \to_{\overline{\mathcal{R}}} \{\overline{f}(\{\overline{\mathit{false}}\}, \{\overline{a}\})\} \to_{\overline{\mathcal{R}}} \boxed{\overline{\{\overline{f}(\{\overline{\mathit{false}}\}, \{\overline{\mathit{true}}\})\}} \to^+_{\overline{\mathcal{R}}} \{\overline{\mathit{true}}\}}$$

In Theorem 2, let $s' = \{\overline{f}(\{\overline{\mathit{false}}\}, \{\overline{\mathit{true}}\})\}$ (reachable) and $t' = \{\overline{\mathit{true}}\}$. □

However, the next result shows that our transformation is also sound when the original CTRS is left linear instead of ground confluent:

Theorem 3. *If $\mathcal{R}$ is left linear and $\{\overline{s}\} \to^*_{\overline{\mathcal{R}}} t'$ then $s \to^*_{\mathcal{R}} \widehat{t'}$. Moreover, if $\{\overline{s}\} \to^*_{\overline{\mathcal{R}}} t'$ has k structural steps, then $s \to^{k'}_{\mathcal{R}} \widehat{t'}$ with $k' \geq k$.*

Thus, our transformation is sound for Example 5. However, Example 4 shows that soundness may not hold if $\mathcal{R}$ is neither ground confluent nor left linear.

Corollary 1. *If $\mathcal{R}$ is ground confluent* ***or*** *left linear, then our transformation is sound and complete, i.e., $s \to^*_{\mathcal{R}} t$ iff $\{\overline{s}\} \to^*_{\overline{\mathcal{R}}} \{\overline{t}\}$ for any $s, t \in T_\Sigma$.*

Therefore, we can semi-decide reachability, $s \to^*_{\mathcal{R}} t$, in a ground confluent or left linear CTRS: (1) transform $\mathcal{R}$ to the TRS $\overline{\mathcal{R}}$; (2) do a breadth-first search in $\overline{\mathcal{R}}$ starting with $\{\overline{s}\}$; (3) if $\{\overline{t}\}$ is reached then return true. The breadth-first search may loop forever if there is no solution for the original problem. However, it will return true *iff* the original problem has a solution. This reachability result is operationally important, since searching is very difficult in CTRSs and it can sometimes lead to defectuous implementations.

Example 6. Consider the following three-rule CTRS: $a \to c$ *if* $a \to b$ and $a \to b$ and $c \to b$. A rewrite engine sensitive to the order in which rules are given may crash when asked to verify $a \to^*_{\mathcal{R}} c$; indeed, Maude does so if the rules are given in the order above. The reason is that although Maude does breadth-first search in general, it chooses not to do it within conditions.

This CTRS is transformed to: $a(\bot) \to a(\{a(\bot)\})$, $c \to \{b\}$, $a(\{b\}) \to \{c\}$, $a(x) \to \{b\}$, $\{\{x\}\} \to \{x\}$. Although this TRS does not terminate either, we can use any rewrite engine which supports breadth-first searching, including Maude, to verify any reachability problem which has solutions in the original system. □

If $\mathcal{R}$ is left linear, due to soundness and completeness, ground confluence of $\overline{\mathcal{R}}$ on reachable terms yields ground confluence of $\mathcal{R}$.

Proposition 2. *If $\overline{\mathcal{R}}$ is left linear and ground confluent on reachable terms, then $\mathcal{R}$ is ground confluent.*

Even though a transformation is sound and complete, one may not necessarily simulate $\mathcal{R}$ through $\overline{\mathcal{R}}$ (see Example 3). We show that if $\mathcal{R}$ is ground confluent and left linear then $\widehat{\cdot}$ defines a simulation relation between $\overline{\mathcal{R}}$ and $\mathcal{R}$:

Proposition 3. ***(Simulation)*** *If $\mathcal{R}$ is ground confluent and left linear then $\overline{\mathcal{R}}$ weakly simulates $\mathcal{R}$ on reachable terms: for any reachable s' with $\widehat{s'} \to_{\mathcal{R}}^{k} t$, there is a $\overline{\Sigma}$-term t' with $\widehat{t'} = t$ and $s' \to_{\overline{\mathcal{R}}}^{*} t'$ using* exactly *k structural steps.*

It is worthwhile noticing that the confluence of $\mathcal{R}$ does *not* imply the confluence of $\overline{\mathcal{R}}$, as the following (counter-)example shows.

Example 7. Consider the confluent one-rule CTRS $\mathcal{R}$ $f(x) \to x$ *if* $g(x) \to$ *false* and its corresponding TRS $\overline{\mathcal{R}}$: $\overline{f}(x, \bot) \to \overline{f}(x, \{\overline{g}(x)\})$, $\overline{f}(\{x\}, y) \to \{\overline{f}(x, \bot)\}$, $\overline{f}(x, \{\overline{false}\}) \to \{x\}$, $\{\{x\}\} \to \{x\}$. Then $\overline{f}(\{\overline{false}\}, \{\overline{false}\})$ rewrites in one step to $\{\overline{false}\}$, in normal form, and $\overline{f}(\{\overline{false}\}, \{\overline{false}\}) \to_{\overline{\mathcal{R}}} \{\overline{f}(\overline{false}, \bot)\} \to_{\overline{\mathcal{R}}} \{\overline{f}(\overline{false}, \{\overline{g}(\overline{false})\})\}$, also in normal form. Hence $\overline{\mathcal{R}}$ is not confluent. □

In fact, for computational equivalence purposes, $\overline{\mathcal{R}}$ does *not* need to be confluent. What is needed is its confluence on *reachable* terms. The next result shows that (ground) confluence is preserved in the presence of left linearity.

Theorem 4. *If $\mathcal{R}$ is left linear and ground confluent then $\overline{\mathcal{R}}$ is ground confluent on reachable terms, or, even stronger, for any reachable terms s'_1, s'_2, if $\widehat{s'_1}$ and $\widehat{s'_2}$ are joinable in t then s'_1 and s'_2 are joinable in t' such that $\widehat{t'} = t$.*

The termination of $\overline{\mathcal{R}}$ on reachable terms implies operational termination of $\mathcal{R}$:

Proposition 4. *If $\overline{\mathcal{R}}$ terminates on $\{\overline{s}\}$, then $\mathcal{R}$ operationally terminates on s.*

The other implication does not hold without additional requirements on $\mathcal{R}$. We will show that confluence or left linearity of $\mathcal{R}$ suffices.

Example 8. Consider the system $\mathcal{R}_t$ in Example 1. Since $\mathcal{R}_t = \mathcal{R}_s \cup \{B \to A\}$, its transformed version will be the same as the one in Example 4, except adding one more rule, $B \to \{A\}$. Remember that with the system $\overline{\mathcal{R}}$ in Example 4 we have obtained that $\{A\} \to_{\overline{\mathcal{R}}}^{*} \{B\}$. With the new rule we therefore get that $\{A\} \to_{\overline{\mathcal{R}}}^{+} \{A\}$, thus the transformed version is not terminating. However, the original system is decreasing, so it is operationally terminating. □

Confluence *or* left linearity of $\mathcal{R}$ preserves termination:

Theorem 5. ***(Termination)*** *If $\mathcal{R}$ operationally terminates on s and is either ground confluent or left linear, then $\overline{\mathcal{R}}$ terminates on $\{\overline{s}\}$.*

Finally, we prove that ground confluence yields computational equivalence:

Theorem 6. *(**Computational equivalence**) If $\mathcal{R}$ is finite, ground confluent and operationally terminates on s, then $\overline{\mathcal{R}}$ is ground confluent and terminates on terms reachable from $\overline{s}$. That is, $\overline{\mathcal{R}}$ is computationally equivalent to $\mathcal{R}$.*

Then one can simulate reduction in a confluent CTRS $\mathcal{R}$ by using the transformed TRS $\overline{\mathcal{R}}$. Reducing a Σ-term t to its normal form in $\mathcal{R}$ can be done as follows: start reducing $\{\overline{t}\}$ in $\overline{\mathcal{R}}$; if it does not terminate, there exists a way t might have not terminated in $\mathcal{R}$; if it terminates and $fn(\{\overline{t}\})$ is its normal form, then $\widehat{fn(\{\overline{t}\})}$ is the normal form of t in $\mathcal{R}$.

If one wants computational equivalence by means of reduction, one has to require confluence as a desired property of both the original and the transformed system, because no search is involved in the process of reaching a normal form. Instead, if one allows the underlying engine to search for normal forms, such as in logic programming paradigms, then one can replace confluence by left linearity (Theorems 3 and 5); this was also the approach taken in [2]. Note, however, that search is potentially exponential in the size of the reduction.

6 Discussion and Future Work

We presented a technique to eliminate conditional rules by replacing them with unconditional rules. The generated TRS is computationally equivalent with the original CTRS provided that the CTRS is ground confluent. Besides the theoretical results, we have also empirically shown that the proposed transformation may lead to the development of faster conditional rewrite engines. In the case of constructor-based CTRSs, the operation $\{\_\}$ is not needed, so our transformation becomes the same as the one in [2]; thus, our theoretical results imply that the transformation in [2] preserves ground confluence, a result not proved in [2] but approached in [5].

We believe that the results presented here can be easily generalized to conditional rewriting systems with extra variables in conditions (deterministic(D) CTRSs). In this framework, operational termination is equivalent to quasi-decreasingness and left linearity translates to *semilinearity* [15] of DCTRSs. The nontrivial proofs of the results in Section 5 (see [20]), were engineered to also work for this case. We refer the interested reader to [20] for details.

Techniques to compact the generated TRS are worthwhile investigating. Also, propagation rules for $\{\_\}$ can destroy useful partial reductions; can one adapt our transformation to restart only the conditions that are invalidated when a rewrite step occurred? We believe that confluence is preserved even in the absence of left linearity or termination but we have not been able to prove it. None of the transformations mentioned in this paper can handle arbitrary rewriting *modulo axioms* in the source CTRS. This seems to be a highly non-trivial problem in its entire generality; however some restricted uses of operators modulo axioms can be handled at no additional complexity (see [20]).

Acknowledgments. We warmly thank Andrei Popescu for several technical suggestions, Claude Marché for referring us to Example 1, Salvator Lucas and José Meseguer for encouraging remarks and suggestions on how to extend this work, and the careful referees for insightful comments on the draft of this paper.

References

[1] H. Aida, J. A. Goguen, and J. Meseguer. Compiling concurrent rewriting onto the rewrite rule machine. In *CTRS'90*, volume 516 of *LNCS*, pages 320–332, 1990.
[2] S. Antoy, B. Brassel, and M. Hanus. Conditional narrowing without conditions. In *PPDP'03*, pages 20–31. ACM Press, 2003.
[3] J. A. Bergstra and J. W. Klop. Conditional rewrite rules: Confluence and termination. *J. of Computer and System Sciences*, 32(3):323–362, 1986.
[4] P. Borovansky, H. Cirstea, H. Dubois, C. Kirchner, H. Kirchner, P. Moreau, C. Ringeissen, and M. Vittek. *ELAN: User Manual*, 2000. Loria, Nancy, France.
[5] B. Brassel. Bedingte narrowing-verfahren mit verzögerter auswertung. Master's thesis, RWTH Aachen, 1999. In German.
[6] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *Maude 2.0 Manual*, 2003. `http://maude.cs.uiuc.edu/manual`.
[7] N. Dershowitz and D. A. Plaisted. Equational programming. In J. E. Hayes, D. Michie, and J. Richards, editors, *Machine Intelligence 11*, pages 21–56. 1988.
[8] R. Diaconescu and K. Futatsugi. *CafeOBJ Report*. World Scientific, 1998. AMAST Series in Computing, volume 6.
[9] E. Giovannetti and C. Moiso. Notes on the elimination of conditions. In *CTRS'87*, volume 308 of *LNCS*, pages 91–97. Springer, 1987.
[10] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OBJ. In *Software Engineering with OBJ*, pages 3–167. Kluwer, 2000.
[11] M. Hanus. The integration of functions into logic programming: From theory to practice. *The Journal of Logic Programming*, 19 & 20:583–628, 1994.
[12] C. Hintermeier. How to transform canonical decreasing CTRSs into equivalent canonical TRSs. In *CTRS'94*, volume 968 of *LNCS*, pages 186–205, 1994.
[13] S. Lucas, C. Marché, and J. Meseguer. Operational termination of conditional term rewriting systems. *Inf. Proc. Letters*, 95(4):446–453, August 2005.
[14] M. Marchiori. Unravelings and ultra-properties. In *ALP'96*, volume 1139 of *LNCS*, pages 107–121. Springer, 1996.
[15] M. Marchiori. On deterministic conditional rewriting. Computation Structures Group, Memo 405, MIT Laboratory for Computer Science, 1997.
[16] N. Nishida, M. Sakai, and T. Sakabe. On simulation-completeness of unraveling for conditional term rewriting systems. In *LA Symposium 2004 Summer*, volume 2004-7 of *LA Symposium*, pages 1–6, 2004.
[17] E. Ohlebusch. Transforming conditional rewrite systems with extra variables into unconditional systems. In *LPAR'99*, volume 1705 of *LNCS*, pages 111–130, 1999.
[18] E. Ohlebusch. *Advanced Topics in Term Rewriting*. Springer, 2002.
[19] G. Roşu. From conditional to unconditional rewriting. In *WADT'04*, volume 3423 of *LNCS*, pages 218–233. Springer, 2004.
[20] T. F. Şerbănuţă and G. Roşu. Computationally equivalent elimination of conditions. Technical Report UIUCDCS-R-2006-2693, UIUC, February 2006.
[21] M. van den Brand, J. Heering, P. Klint, and P. A. Olivier. Compiling language definitions: the ASF+SDF compiler. *ACM TOPLAS*, 24(4):334–368, 2002.
[22] P. Viry. Elimination of conditions. *J. of Symb. Comp.*, 28:381–401, Sept. 1999.