Towards Effectively Eliminating Conditional Rewrite Rules

Traian Florin Şerbănuţă and Grigore Roşu

Department of Computer Science,
University of Illinois at Urbana-Champaign.
{tserban2,grosu}@cs.uiuc.edu

Abstract. Conditional rewrite rules are notorious for being difficult to implement in rewrite engines. This is because, like in the case of function calls in programming language implementations, rewrite engines need to "freeze" the current rewriting environment and to create a new one in which the condition is reduced. Stacking these rewriting environments efficiently can easily become a nontrivial task, which can have a direct impact on the efficiency of rewriting. Continuation-passing-style (CPS) transformations are used as a front-end in many programming languages to transform the programs to compile into a convenient and highly optimizable form, in which functions never need to return their values: they just pass their computed values to the current data-context, which "knows" how to continue the computation. We argue that a similar transformation technique can be applied to conditional rewrite systems, to transform them into computationally equivalent unconditional rewrite systems. In this paper we present the first steps towards such a transformation. No special support is needed from the underlying unconditional rewrite engine, so the presented technique can be used as a front-end to any of the current rewrite engines. Since unconditional rewriting is more amenable to parallelization, our transformation is expected to lead to efficient concurrent implementations of rewriting.

1 Introduction

Conditional term rewriting is a crucial paradigm in the algebraic specification of abstract data types, since it provides a natural means for executing equational specifications. Many specification languages today, including Maude [4], ELAN [3], OBJ [9], CafeOBJ [7], provide conditional rewrite engines to allow users to execute and reason about specifications. Conditional rewriting also plays a foundational role in functional logic programming [10]. Additionally, there are many researchers, including ourselves, considering rewriting a powerful programming paradigm by itself, who are often frustrated that conditional rewrite "programs" are significantly slower than unconditional ones doing the same thing.

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 rewrite rule applications, such a control context may grow arbitrarily. Our technique 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 for increased efficiency, and our transformation technique can be seen as a compiler.

Experiments performed on two fast rewrite engines, Maude and Elan, show that significant speedups can be obtained right now if one uses our transformation technique as a front-end. However, since these rewrite engines are optimized for conditional rewriting, we expect significant further increases in efficiency if one just focuses on the much simpler problem of developing optimized unconditional rewrite engines and use our technique as a front-end. Moreover, one can now develop *parallel rewrite machines* without worrying about conditions which somehow obstruct the potential for high parallelism.

On the theoretical side, we show that our transformation is *sound*, i.e., any reduction in the original rewriting system has a corresponding reduction in the transformed one; we show that if the original system is confluent then our transformation is also *complete*, i.e., any relevant reduction in the generated system corresponds to some reduction in the original one. However, soundness and completeness of the transformation does not immediately imply the *computational equivalence* of the two rewriting systems. In order to achieve this, backed by several examples and strong intuitions, we conjecture a natural result that hopefully will be rigorously proved in the near future. This is the reason for which we used the word "towards" in the title.

2 Preliminaries

We recall some basic notions of conditional rewriting, referring the interested reader to [14] for more details. An (unsorted) *signature* Σ is a finite set of operational symbols, each having zero or more arguments. We let $\Sigma_n \subseteq \Sigma$ denote the set of operations of n arguments. Operations *of zero arguments* in Σ_0 are called *constants*. We assume an infinite set of *variables* $\mathcal{X}$. Given a signature Σ and a set of variables $X \subseteq \mathcal{X}$, we let $T_\Sigma(X)$ denote the algebra of *Σ-terms* over variables in X. We let T_Σ denote the algebra $T_\Sigma(\emptyset)$ of *ground* terms. A map $\theta : \mathcal{X} \to T_\Sigma(\mathcal{X})$ can be uniquely extended to a morphism of algebras $\theta : T_\Sigma(\mathcal{X}) \to T_\Sigma(\mathcal{X})$ replacing each $x \in X$ by a term $\theta(x)$. A conditional Σ-rewrite rule has the form $l \to r$ `if` $cl \to cr$, where l, r, cl and cr, are Σ-terms in $T_\Sigma(\mathcal{X})$. The term l is called the *left-hand-side (lhs)*, r is called the *right-hand-side (rhs)*, and $cl \to cr$ is called the *condition* of the rule. We disallow rewriting rules whose lhs is a variable and assume that the lhs contains all the variables that occur in the rule. Following the terminology in [13], our rules are *of type 1*. An *unconditional* rewrite rule has the form $l \to r$.

A *conditional (unconditional) Σ-term rewrite system* $\mathcal{R} = (\Sigma, R)$, abbreviated *CTRS (TRS)*, consists of a finite set R of conditional (unconditional) Σ-rewrite rules. We here use only *normal* CTRSs (see [6]), i.e., ones whose rhs

of the condition is a ground normal form for the unconditional system obtained by disregarding all the conditions. Such systems are called of type II [2]. Any Σ-rewrite system $\mathcal{R} = (\Sigma, R)$ generates a relation $\rightarrow_{\mathcal{R}}$ on $T_\Sigma(\mathcal{X})$, defined as follows. For any $\theta : \mathcal{X} \rightarrow T_\Sigma(\mathcal{X})$, $\gamma[\theta(l)] \rightarrow_{\mathcal{R}} \gamma[\theta(r)]$ whenever $\theta(cl) \rightarrow^\star_{\mathcal{R}} cr$, where γ is context, i.e., a term having one occurrence of a special variable $*$, $\gamma[\theta(l)]$ is the term obtained by substituting $*$ with $\theta(l)$ in γ, and $\rightarrow^\star_{\mathcal{R}}$ is the reflexive and transitive closure of $\rightarrow_{\mathcal{R}}$. Note that $\rightarrow^*_{\mathcal{R}}$ is the least relation on $T_\Sigma(\mathcal{X})$ closed under reflexivity, transitivity, congruence and $\mathcal{R}$-substitution.

Positions are strings of numbers describing paths to subterms: the subterm at position $i\alpha$ in a term $\sigma(t_1, \ldots, t_i, \ldots, t_n)$ is the subterm at position α in t_i . A rewrite step *occurred at position* α in a term t when γ is obtained from t by replacing its subterm at position α by $\star$. We may let $t_{\alpha \leftarrow s}$ denote the term obtained by substituting the subterm at position α in t by s.

On termination. Terms which cannot be reduced any further in a rewriting system are called *normal forms* for that system. Rewriting of a given term may not terminate for two reasons [16]: 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, e.g., Maude [4] or ELAN [3], 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-rewriting system *strongly terminating*, or simply *terminating*, iff it always reduces any Σ-term to a normal form regardless of the order of rules appliance. Note that this notion is slightly different from the *effective termination* introduced in [12], as shown by the following example.

Example 1. Consider the following three-rule rewrite system: $a \rightarrow b$, $a \rightarrow c$ if $a \rightarrow b$ and $c \rightarrow b$ The system Maude crashes when asked to search whether $a \rightarrow^*_{\mathcal{R}} c$. This is a good example of a system that is *confluent* and *effectively terminating* but *not* strongly terminating. Strong termination is based on the belief that, in general, one cannot expect a rewrite engine to be "smart" enough to pick the right rewrite sequence to satisfy a condition.

2.1 Related Work

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. Significant efforts have been dedicated to transformations preserving only certain properties, e.g., termination and/or confluence. We do not discuss these here; the interested reader is referred to Ohlebusch [14].

We focus on transformations that generate TRSs *computationally equivalent* to CTRSs. The obtained TRSs are intended to be *transparently* and *efficiently* used to reduce terms or test reachability in the original CTRSs. The first attempt in this category is due to Bergstra and Klop [2] for a restricted class of CTRSs; the transformation in [2] was shown unsound by Dershowitz and Okada [5]. The transformation in Giovannetti and Moiso [8] works only under severe restrictions

on the original CTRS: no superposition, simply terminating (requires a simplification ordering), and non-overlapping of conditions with lhs terms. Hintermeier [11] proposes a technique where an "interpreter" for CTRS is defined as a TRS, providing explicit rewrite definitions for matching and applications of rewrite rules. The resulting TRS is too inefficient to be usable in practice.

The idea of our transformation comes from the use of *continuations* [15] in the implementation of programming languages, as a mean to pass control context into data context, and builds on two other techniques: that of Viry [17] and that of Roşu [16]. Like in [17, 16], our technique is based on decorations of terms: as many auxiliary arguments are added to each operation σ as conditional rules in the original CTRS having σ at the top of their lhs. The procedure in [17] encodes the condition of each rule within a special data-structure that occurs as the corresponding auxiliary argument associated to the operation occurring at the top of its lhs. Two unconditional rules are added in the generated TRS for each conditional rule in the original CTRS, one for initializing the special data-structure and the other for continuing the rewriting process when the condition was evaluated. For example, the left CTRS (taken from [17]) below is transformed into the right TRS:

$$\begin{cases} f(g(x)) \to p(x) \ \texttt{if}\ c(x) \to^* \mathit{true} \\ f(h(x)) \to q(x) \ \texttt{if}\ d(x) \to^* \mathit{true} \\ c(a) \to \mathit{true} \end{cases} \qquad \begin{cases} f(g(x) \mid \bot, z) \to f(g(x) \mid [c(x), (x)], z) \\ f(x \mid [\mathit{true}, (y)], z) \to p(y) \\ f(h(x) \mid z, \bot) \to f(h(x) \mid z, [d(x), (x)]) \\ f(x \mid z, [\mathit{true}, (y)]) \to q(y) \\ c(a) \to \mathit{true} \end{cases}$$

where "|" is syntactic sugar for ",", separating the normal arguments from the auxiliary ones; "$\bot$" is a special constant whose occurrence states that the corresponding conditional rule has not been tried yet on the current position. A structure $[u, \vec{s}]$ occurring in a rewriting sequence as an auxiliary argument of an operation means that u is the current reduction status of the corresponding condition that started to be evaluated at some previous moment, and that $\vec{s}$ was the substitution at that point that allowed the lhs of that rule to match. The substitution is needed by the second unconditional rule associated to a conditional rule, to correctly initiate the reduction of the rhs of the original conditional rule.

Despite being proved sound and complete by Viry [17], the procedure above, unfortunately, cannot be used *as is* to interpret any CTRS on top of a TRS. That is because it destroys the confluence of the original CTRS, thus leading to normal forms in the TRS which can be further reduced in the CTRS. Indeed, let us consider the following CTRS $\mathcal{R}$, from Antoy, Brassel and Hanus [1], together with Viry's transformation $\mathcal{R}'$:

Example 2.

$$(\mathcal{R}) \begin{cases} f(g(x)) \to x \ \texttt{if}\ x \to^* 0 \\ g(g(x)) \to g(x) \end{cases} \qquad (\mathcal{R}') \begin{cases} f(g(x) \mid \bot) \to f(g(x) \mid [x, (x)]) \\ f(x \mid [0, (y)]) \to y \\ g(g(x)) \to g(x) \end{cases}$$

$\mathcal{R}$ is confluent but $\mathcal{R}'$ is not: $f(g(g(0)) \mid \bot)$ can be reduced to both 0 and $f(g(0) \mid [g(0), (g(0))])$; the latter occurs because the "conditional" rule is first

tried and "failed", then the "unconditional" one is applied successfully thus changing the context so that the "conditional" rule becomes applicable, but it fails to apply since it was already marked as "tried". To solve this problem, Viry [17] proposes a reduction strategy within the generated TRS, called *conditional eagerness*, stating that $t_1, ..., t_n$ must be already in normal form before a "conditional" rule can be applied on a term $f(t_1, ..., t_n \mid \bot, ..., \bot)$. This way, in the example above, $g(g(0))$ is enforced to be first evaluated to $g(0)$ and only then $f(g(0) \mid \bot)$ is applied the "conditional" rule and eventually reduced to 0. However, conditional eagerness does not seem to be trivial to enforce in an unconditional rewrite engine, unless that is internally modified. One simple, but very restrictive, way to ensure conditional eagerness is to enforce innermost rewriting both in the original CTRS and in the resulting TRS.

A different fix to Viry's technique was proposed by Antoy, Brassel and Hanus [1], namely to restrict the input CTRSs to *constructor-based* ones, i.e., ones in which the operations are split into *constructors* and *defined*, and the lhs of each rule is a term of the form $f(t_1, ..., t_n)$, where f is defined and $t_1, ..., t_n$ are all constructor terms. Note that the problematic CTRS above is *not* constructor-based. While constructor-baseness is an easy to check and automatic correctness criterion, we believe that it is an unnecessary strong restriction on the input CTRS, which may make the translation useless in many situations of practical interest. The transformation proposed by Roşu in [16] does not require conditional eagerness but instead requires the rewrite engine to support *contextual rewriting strategies* making it less friendly w.r.t parallelism.

3 The Transformation

Auxiliary arguments are added to some operators to keep the necessary control context information. This way, terms store information about the conditional rules that can be potentially applied on them. Let $\mathcal{R} = (\Sigma, E)$ be any Σ-CTRS. A *σ-conditional rule* [17] is a conditional rule with σ as the top symbol of its lhs, that is, a rule of the form $\sigma(t_1, \ldots, t_n) \ \rightarrow \ r \ \ \texttt{if} \ \ cl \rightarrow cr$.
Let k_σ be the total number of σ-conditional rules. and let $\rho_{\sigma,i}$ denote the i-th σ-conditional rule in $\mathcal{R}$.

3.1 The Signature Transformation

Let $\overline{\Sigma}$ be the signature containing:

- a fresh constant $\bot$;
- a fresh unary operator $\{\_\}$ (first defined in [16]);
- for any $\sigma \in \Sigma_n$, an operator of $n + k_\sigma$ arguments, $\overline{\sigma} \in \overline{\Sigma}_{n+k_\sigma}$; the additional k_σ arguments 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 intuition for the additional arguments comes from the idea of passing the control context (due to conditional rules) into data context:

the additional i-th argument of $\overline{\sigma}$ at some position in a term to rewrite maintains the status of appliance of $\rho_{\sigma,i}$; if $\bot$ then that rule was not tried, otherwise the condition is being under evaluation (it may also be 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 $\overline{\cdot} : T_\Sigma(\mathcal{X}) \to T_{\overline{\Sigma}}(\mathcal{X})$ be a map defined inductively:

$\overline{x} = x$ for any variable $x \in \mathcal{X}$, and
$\overline{\sigma(t_1, \ldots, t_n)} = \overline{\sigma}(\overline{t_1}, \ldots, \overline{t_n}, \bot, \ldots, \bot)$ for any operation $\sigma \in \Sigma_n$ and any terms $t_1, \ldots, t_n \in T_\Sigma(\mathcal{X})$.

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 variable $X \subseteq \mathcal{X}$, as: $\widetilde{x}^X = x$ for any variable $x \in X$, and $\widetilde{\sigma(t_1, \ldots, t_n)}^X = \overline{\sigma}(\widetilde{t_1}^X, \ldots, \widetilde{t_n}^X, b_1, \ldots, b_{k_\sigma})$ for any $\sigma \in \Sigma_n$ and any terms $t_1, \ldots, t_n \in T_\Sigma(X)$, where $b_1, \ldots, b_{k_\sigma} \in \mathcal{X} - X$ are some arbitrary but fixed different fresh variables that do not occur neither in X nor 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.

Given a $\overline{\Sigma}$-term t of the form $\overline{\sigma}(t_1, \ldots, t_n, C_1, \ldots, C_{k_\sigma})$ for some $\sigma \in \Sigma_n$, a natural number i between 1 and k_σ, and a $\overline{\Sigma}$ term u, we let $t_{i/u}$ denote the $\overline{\Sigma}$-term $\overline{\sigma}(t_1, \ldots, t_n, C_1, \ldots, C_{i-1}, u, C_{i+1}, \ldots, C_{k_\sigma})$, that replaces C_i by u.

3.2 The Rewrite Rules Transformation

Given a Σ-CTRS $\mathcal{R}$, let $\overline{\mathcal{R}}$ be the $\overline{\Sigma}$-TRS obtained as follows.For each 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 $\overline{\rho}_{\sigma,i} : \widetilde{l}^X_{i/\bot} \to \widetilde{l}^X_{i/\{\overline{cl}\}}$ and $\overline{\rho}'_{\sigma,i} : \widetilde{l}^X_{i/\{cr\}} \to \{\overline{r}\}$.
For each unconditional rewrite rule $l \to r$ in $\mathcal{R}$, add to $\overline{\mathcal{R}}$ rule $\widetilde{l}^X \to \{\overline{r}\}$.
For each $\sigma \in \Sigma_n$ and each $1 \le i \le n$, add to $\overline{\mathcal{R}}$

$$\overline{\sigma}(x_1, ..., x_{i-1}, \{x_i\}, x_{i+1}, ..., x_n, b_1, ..., b_{k_\sigma}) \to \\ \to \{\overline{\sigma}(x_1, ..., x_{i-1}, x_i, x_{i+1}, ..., x_n, \bot, ..., \bot)\}\,,$$

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. 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.

3.3 Examples

We next illustrate our transformation on several examples.

Confluence is preserved. Consider the system presented in Example 2 together with its transformation using our technique:

$$\begin{cases} f(g(x)) \to x \ \texttt{if}\ x \to^* 0 \\ g(g(x)) \to g(x) \end{cases} \qquad \begin{cases} \overline{f}(\overline{g}(x), \bot) \to \overline{f}(\overline{g}(x), \{x\}) \\ \overline{f}(\overline{g}(x), \{\overline{0}\}) \to \{x\} \\ \overline{g}(\overline{g}(x)) \to \{\overline{g}(x)\} \end{cases} \qquad \begin{array}{l} \overline{g}(\{x\}) = \{\overline{g}(x)\} \\ \overline{f}(\{x\}, b) = \{\overline{f}(x, \bot)\} \\ \{\{x\}\} = \{x\} \end{array}$$

The problem that required conditional eagerness 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 resembling this 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 (taken from [16]). 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:

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

In order to check whether a natural number n, i.e., a term consisting of n successor operations applied to 0, is odd, a 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.

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 aesthetical reasons we overline only those operations that change):

$$\begin{array}{ll} \overline{o}(0, c_1, c_2) \to \{\mathit{false}\}, & \overline{e}(0, c_1, c_2) \to \{\mathit{false}\}, \\ \overline{o}(s(x), \{\mathit{false}\}, c_2) \to \{\mathit{false}\} & \overline{e}(s(x), \{\mathit{false}\}, c_2) \to \{\mathit{false}\} \\ \overline{o}(s(x), c_1, \{\mathit{true}\}) \to \{\mathit{true}\} & \overline{e}(s(x), c_1, \{\mathit{true}\}) \to \{\mathit{true}\} \\ \overline{o}(s(x), \bot, c_2) \to \overline{o}(s(x), \{\overline{e}(x, \bot, \bot)\}, c_2) & \overline{e}(s(x), \bot, c_2) \to \overline{e}(s(x), \{\overline{o}(x, \bot, \bot)\}, c_2) \\ \overline{o}(s(x), c_1, \bot) \to \overline{o}(s(x), c_1, \{\overline{e}(x, \bot, \bot)\}) & \overline{e}(s(x), c_1, \bot) \to \overline{e}(s(x), c_1, \{\overline{o}(x, \bot, \bot)\}) \\ s(\{x\}) \to \{s(x)\} & \overline{o}(\{x\}, c_1, c_2) \to \{\overline{o}(x, c_1, c_2)\} \\ \{\{x\}\} \to \{x\} & \overline{e}(\{x\}, c_1, c_2) \to \{\overline{e}(x, c_1, c_2)\} \end{array}$$

The unconditional rule for $\overline{o}$ says that 0 is not an odd number, regardless of the control context. The first conditional rule for $\overline{o}$ has the constant $\perp$ as the first auxiliary argument of its lhs, telling that if the condition is not currently under evaluation, then one can enable its evaluation process by plugging its appropriately modified lhs into the auxiliary position. Next, if the condition's lhs reaches the desired rhs, then 0 is returned. The argument c_2 is just a fresh variable, stating that the status of evaluating the condition of the second conditional rule for odd has no influence on the evaluation of the condition of the first conditional rule for odd. Note, however that the evaluation of both conditions is attempted concurrently. 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 $\{odd(n, \perp, \perp)\}$, under the resulting TRS.

Quotient/Reminder (inspired from [14]). Consider the following CTRS for computing the quotient and reminder of two numbers.

$$
\begin{array}{ll}
x < 0 \to \mathit{false} & \\
0 < s(x) \to \mathit{true} & s'(\langle x, y\rangle) \to \langle s(x), y\rangle \\
s(x) < s(y) \to x < y & \%(0, s(y)) \to \langle 0, 0\rangle \\
0 - s(y) \to 0 & \%(s(x), s(y)) \to \langle 0, s(x)\rangle \text{ if } x < y \to \mathit{true} \\
x - 0 \to x & \%(s(x), s(y)) \to s'(\%(x - y, s(y))) \text{ if } x < y \to \mathit{false} \\
s(x) - s(y) \to x - y &
\end{array}
$$

The corresponding TRS is the following:

$$
\begin{array}{ll}
x < 0 \to \{\mathit{false}\} & \overline{\%}(0, s(y), c_1, c_2) \to \{\langle 0, 0\rangle\} \\
0 < s(x) \to \{\mathit{true}\} & \overline{\%}(s(x), s(y), \perp, c_2) \to \overline{\%}(s(x), s(y), \{x < y\}, c_2) \\
s(x) < s(y) \to \{x < y\} & \overline{\%}(s(x), s(y), \{\mathit{true}\}, c_2) \to \{< 0, s(x) >\} \\
0 - s(y) \to \{0\} & \overline{\%}(s(x), s(y), c_1, \perp) \to \overline{\%}(s(x), s(y), c_1, \{x < y\}) \\
x - 0 \to \{x\} & \overline{\%}(s(x), s(y), c_1, \{\mathit{false}\}) \to \{s'(\overline{\%}(x - y, s(y), \perp, \perp))\} \\
s(x) - s(y) \to \{x - y\} & s'(\langle x, y\rangle) \to \{\langle s(x), y\rangle\}
\end{array}
$$

$$
\begin{array}{ll}
\{\{x\}\} \to \{x\} & s'(\{x\}) \to \{s'(x)\} \\
s(\{x\}) \to \{s(x)\} & \langle \{x\}, y\rangle \to \{\langle x, y\rangle\} \\
\{x\} < y \to \{x < y\} & \langle x, \{y\}\rangle \to \{\langle x, y\rangle\} \\
x < \{y\} \to \{x < y\} & \overline{\%}(\{x\}, y, c_1, c_2) \to \{\overline{\%}(x, y, \perp, \perp)\} \\
\{x\} - y \to \{x - y\} & \overline{\%}(x, \{y\}, c_1, c_2) \to \{\overline{\%}(x, y, \perp, \perp)\} \\
x - \{y\} \to \{x - y\} &
\end{array}
$$

3.4 An Improved Transformation

Notice that in the previous two examples there were several pairs of conditional rewrite rules having their lhs's and their left sides of the condition identical. In such cases, one can reduce the number of arguments added to the top operation and the number of rewrite rules. For example, the transformation of

$$
(\mathcal{R}) \begin{cases} \%(s(x), s(y)) \to \langle 0, s(x)\rangle \text{ if } x < y \to \mathit{true} \\ \%(s(x), s(y)) \to s'(\%(x - y, s(y))) \text{ if } x < y \to \mathit{false} \end{cases}
$$

can be:

$$(\mathcal{R})\begin{cases} \%(s(x),s(y),\bot) \to \%(s(x),s(y),\{x<y\}) \\ \%(s(x),s(y),\{true\}) \to \{\langle 0,s(x)\rangle\} \\ \%(s(x),s(y),\{false\}) \to \{s'(\%(x-y,s(y),\bot))\} \end{cases}$$

This leads leads to more efficient generated TRSs, e.g., the Odd/Even CTRS is translated into a TRS that computes odd/even in linear time. We only mention this possible improvement here, without proofs.

4 Theoretical aspects

4.1 $\overline{\Sigma}$-terms

$\overline{\Sigma}$-terms are more complex than the Σ-terms. There can even be some $\overline{\Sigma}$-terms that do not resemble any Σ-term. We next define and discuss several classes of $\overline{\Sigma}$-terms that will be used in the sequel.

Definition 1. *A $\overline{\Sigma}$-term t' is **structural** iff $t' = x$ where x is a variable, or $t' = \{t''\}$ where t'' is structural, or $t' = \overline{\sigma}(t'_1,\ldots,t'_n,C_1,\ldots,C_{k_\sigma})$ where $\sigma \in \Sigma_n$ and t'_i is structural for each $1 \le i \le n$. We say that a position α is **structural for** t', where t' is a structural term, iff α is empty, or $t' = \{t''\}$ and $\alpha = 1\alpha'$ with α' being structural for t'', or $t' = \overline{\sigma}(t'_1,\ldots,t'_n,C_1,\ldots,C_{k_\sigma})$ with $\sigma \in \Sigma_n$ and $\alpha = i\alpha'$ where $1 \le i \le n$ and α' is structural for t'_i. A ground $\overline{\Sigma}$-term t' is **reachable** iff there is some ground Σ-term t such that $\{\overline{t}\} \to^*_{\overline{\mathcal{R}}} \{t'\}$.*

Note that the lhs and rhs of any (unconditional) rule in $\overline{\mathcal{R}}$ are structural.

Proposition 1. *The following properties hold:*

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

Before we formalize the relationship between CTRSs and their unconditional variants, let us define a partial map on terms, $\widehat{\cdot}: T_{\overline{\Sigma}}(X) \to T_\Sigma(X)$ only defined for $\overline{\Sigma}$-structural terms:

- $\widehat{x} = x$ for any variable x.
- $\widehat{\{t'\}} = \widehat{t'}$
- $\overline{\sigma}(t'_1,\ldots,\widehat{t'_n,C_1},\ldots,C_{k_\sigma}) = \sigma(\widehat{t'_1},\ldots\widehat{t'_n})$

Therefore, $\widehat{t'}$ forgets all the auxiliary arguments of each operation occurring in t'. Note in particular that $\widehat{\overline{t}} = t$ for any $t \in T_\Sigma$.

4.2 Soundness

Soundness means that rewriting that can be executed in the original CTRS can also be simulated on the corresponding TRS. In other words, "everything that can be done on a term s in $\mathcal{R}$ can also be done on the term $\{\overline{s}\}$ in $\overline{\mathcal{R}}$".

Proposition 2. ***Soundness.*** *If $s, t \in T_\Sigma$ and $s \to_{\mathcal{R}}^* t$ then $\{\overline{s}\} \to_{\overline{\mathcal{R}}}^* \{\overline{t}\}$. In particular, if $fn(t)$ is a normal form of t in $\mathcal{R}$ then $\{\overline{t}\} \to_{\overline{\mathcal{R}}}^\star \{\overline{fn(t)}\}$.*

Although it may not seem so, the converse of the above proposition is a hard problem and does not hold in general as shown by the following example from [12] (see also [14]).

Example 3. Consider the following conditional rewriting system $\mathcal{R}$

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

and its unconditional transformation $\overline{\mathcal{R}}$

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

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

$$\begin{array}{l} \{A\} \to_{\overline{\mathcal{R}}}^+ \{h(f(a,\bot), f(b,\bot)\} \to_{\overline{\mathcal{R}}}^+ \{h(f(\{d\},\{c\}), f(b,\bot))\} \\ \to_{\overline{\mathcal{R}}}^+ \{h(f(\{d\},\{c\}), f(\{d\},\{c\}))\} \to_{\overline{\mathcal{R}}}^+ \{g(f(\{d\},\{c\}), f(\{d\},\{c\}), f(k,\bot))\} \\ \to_{\overline{\mathcal{R}}}^+ \{g(f(\{d\},\{e\}), f(\{d\},\{c\}), f(k,\bot))\} \to_{\overline{\mathcal{R}}}^+ \{g(d, f(\{d\},\{c\}), f(k,\bot))\} \\ \to_{\overline{\mathcal{R}}}^+ \{g(d, f(\{m\},\{l\}), f(k,\bot))\} \to_{\overline{\mathcal{R}}}^+ \{g(d, f(\{m\},\{l\}), f(\{m\},\{l\}))\} \to_{\overline{\mathcal{R}}}^+ \{A\}, \end{array}$$

but is not the case that $A \to_{\mathcal{R}}^+ A$.

Even though Proposition 2 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 is not true that $\{\overline{s}\} \to_{\overline{\mathcal{R}}}^\star \{\overline{t}\}$ then it is also not true that $s \to_{\mathcal{R}}^* t$. Of course, in order for this to work, 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 $\mathcal{R}$.

4.3 Completeness

Completeness, in this context, means that any rewrite in $\overline{\mathcal{R}}$ of a $\overline{\Sigma}$-term $\{\overline{s}\}$ corresponding to a Σ-term s, corresponds to a rewrite of s in $\mathcal{R}$. Unfortunately, this result does not hold without restricting $\mathcal{R}$. We next show that confluence of $\mathcal{R}$ suffices.

Theorem 1. *Suppose that $\mathcal{R}$ is ground confluent. If s' and t' are reachable terms such that $s' \to^{\star}_{\overline{\mathcal{R}}} t'$. then $\widehat{s'} \to^{*}_{\mathcal{R}} \widehat{t'}$. In particular, our transformation is sound and complete, i.e., $s \to^{*}_{\mathcal{R}} t$ iff $\{\overline{s}\} \to^{\star}_{\overline{\mathcal{R}}} \{\overline{t}\}$ for any $s, t \in T_{\Sigma}$.*

This theorem establishes a procedure for semi-deciding the reachability problem for a confluent CTRS:

Consider a ground confluent CTRS $\mathcal{R}$ and a reachability problem $s \to^{\star}_{\mathcal{R}} t$.
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.

The importance of this reachability result is given by the fact that searching is very difficult when using conditional rewrite rules and it can sometimes lead to defectuous implementations.

Example 4. Recall Example 1. There Maude crashes trying to solve a reachability problem that has a solution. That system is transformed to:

$$\begin{array}{lll} a(\bot) \to a(\{a(\bot)\}) & c \to \{b\} & \\ a(\{b\}) \to \{c\} & a(x) \to \{b\} & \{\{x\}\} \to \{x\} \end{array}$$

Although the above system doesn't 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.

Unfortunately, the former claim in Theorem 1 may not hold if the original system is not confluent:

Example 5. Consider the following CTRS and its unconditional transformation:

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

Note that $\overline{f}(\{\overline{\mathit{false}}\}, \{\overline{\mathit{true}}\}) \to^{*}_{\overline{\mathcal{R}}} \{\mathit{true}\}$ satisfies the hypothesis of Theorem 1 as shown by the following rewriting sequence:

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

However, it is *not* the case that $f(\mathit{false}) \to_{\mathcal{R}} \mathit{true}$.

One may argue that confluence is needed because we tried to prove a stronger result than just the completeness. However, Example 3 shows clearly that completeness may not hold if $\mathcal{R}$ is not confluent.

4.4 Normal Forms

We say that a term t is a *strong normal form* if the process of verifying that it is a normal form (meaning trying and failing all the rules at any positions) terminates regardless of the order in which the rules are applied. It is obvious that if a CTRS is strongly terminating then all its normal forms are strong.

Proposition 3. *Let t be a Σ-term which is a strong normal form w.r.t $\mathcal{R}$. Then there exists a normal form $fn(\overline{t})$ w.r.t. $\overline{\mathcal{R}}$ such that*

1. $\overline{t} \to^*_{\overline{\mathcal{R}}} fn(\overline{t})$
2. *$fn(\overline{t})$ has no brackets at top.*
3. *$\widehat{fn(\overline{t})} = t$, meaning that the structure of the normal form is the same.*

The $fn(\overline{t})$ in the proposition above is the decorated variant of t showing explicitly all the failed application of conditional rewrite rules.

Note that Proposition 3 does *not* say that normal forms in $\overline{\mathcal{R}}$ correspond to normal forms in $\mathcal{R}$.

Example 6. Consider the following CTRS and its unconditional transformation:

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

One can notice that $\overline{f}(\overline{a}, \{\overline{\mathit{true}}\})$ is a reachable normal form in $\overline{\mathcal{R}}$; it can be reached by the following rewriting sequence:

$$\{\overline{f}(\overline{a}, \bot)\} \to_{\overline{\mathcal{R}}} \{\overline{f}(\overline{a}, \{\overline{g}(\overline{a})\})\} \to_{\overline{\mathcal{R}}} \{\overline{f}(\overline{a}, \{\{\overline{\mathit{true}}\}\})\} \to_{\overline{\mathcal{R}}} \{\overline{f}(\overline{a}, \{\overline{\mathit{true}}\})\}$$

But $f(a)$ is not a normal form in $\mathcal{R}$ since $f(a) \to_{\mathcal{R}} a$.

5 Conjecture

The original motivation of our work was to obtain a simple an effective translation of a CTRS into a *computationally equivalent* TRS. While a general purpose transformation would be highly desirable, we believe that there is enough evidence that such a transformation would be highly non-trivial. However, thinking in terms of (deterministic) "computation", one can take the liberty to first focus just on confluent and terminating CTRS's.

Despite proving our transformation sound and complete, and despite showing that it can be used as a semi-decision procedure for the reachability problem for the input CTRS, unfortunately we still do not have a solution to our original problem. Theorem 1 and the various non-confluent counter-examples suggest that confluence of the input CTRS is a key requirement in order for our translation to yield a computationally equivalent TRS.

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 CTRS $\mathcal{R}$ consisting of one conditional rule, $f(x) \to x$ if $g(x) \to \mathit{false}$ and its corresponding TRS $\overline{\mathcal{R}}$:

$$\begin{array}{ll} \overline{f}(x,\bot) \to \overline{f}(x,\{\overline{g}(x)\}) & \overline{f}(\{x\},y) \to \{\overline{f}(x,\bot)\} \\ \overline{f}(x,\{\overline{\mathit{false}}\}) \to \{x\} & \{\{x\}\} \to \{x\} \end{array}$$

We can see that $\mathcal{R}$ is trivially confluent. However, $\overline{f}(\{\overline{\mathit{false}}\},\{\overline{\mathit{false}}\})$ rewrites in one step to $\{\overline{\mathit{false}}\}$, which is in normal form, and

$$\overline{f}(\{\overline{\mathit{false}}\},\{\overline{\mathit{false}}\}) \to_{\overline{\mathcal{R}}} \{\overline{f}(\overline{\mathit{false}},\bot)\} \to_{\overline{\mathcal{R}}} \{\overline{f}(\overline{\mathit{false}},\{\overline{g}(\overline{\mathit{false}})\})\}$$

which is also in normal form. Therefore, $\overline{\mathcal{R}}$ is not confluent.

In fact, for computational equivalence purposes, $\overline{\mathcal{R}}$ does *not* really need to be confluent. What is needed is its confluence on *reachable* terms. Counting on intuitions gathered while developing the transformation in this paper and the one in [16], as well as their associate proofs and (counter-)examples, we strongly claim the following important result. Despite being quite intuitive, its rigorous proof must be very difficult:

Conjecture 1. *If $\mathcal{R}$ is ground confluent then*

1. *$\overline{\mathcal{R}}$ is also confluent on reachable terms (reachable terms are ground);*
2. *$\mathcal{R}$ is (strongly) terminating iff $\overline{\mathcal{R}}$ is terminating for reachable terms.*

Proposition 4. *If Conjecture 1 holds then $t \in T_\Sigma$ has a normal form $\mathit{fn}(t)$ in $\mathcal{R}$ iff $\{\overline{t}\}$ has a normal form $\mathit{fn}(\{\overline{t}\})$ in $\overline{\mathcal{R}}$ and $\widehat{\mathit{fn}(\{\overline{t}\})} = \mathit{fn}(t)$.*

Then one can simulate reduction in a confluent and terminating CTRS $\mathcal{R}$ using the transformed TRS $\overline{\mathcal{R}}$. Reducing a Σ-term t to its normal form in $\mathcal{R}$ can be done as follows: reduce its corresponding $\overline{\Sigma}$-term $\{\overline{t}\}$ to its (unique) normal form $fn(\{\overline{t}\})$ in $\overline{\mathcal{R}}$ and return $\widehat{fn(\{\overline{t}\})}$ the (unique) normal form of t in $\mathcal{R}$.

6 Experiments

The major reason for which we wanted to translate a CTRS into a computationally equivalent TRS that can run on any unrestricted (unconditional) rewrite engine was essentially the potential to device highly parallelizable rewrite engines. Like in [16], it was again a pleasant surprise to note that our transformation can actually bring immediate benefits if implemented as a front-end to *existing* 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 more efficient.

We next give some numbers regarding the speed of the generated TRS. We used Maude and Elan as rewrite engines and the examples **Odd/Even** and **Quotient/Reminder**. 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***

those using the transformation described in section 3.4. Times presented in the table were obtained on a computer equipped with a single 2 GHz Pentium 4 CPU and 512MB RAM.

Odd/Even	Maude			Elan		
reducing odd(22)	Cond	Uncond	Uncond*	Cond	Uncond	Uncond*
	20 sec	6 sec	∼0 sec	1286 sec	151 sec	∼0 sec

Quotient/Reminder	Maude			Elan		
1275000/130	Cond	Uncond	Uncond*	Cond	Uncond	Uncond*
	5.9 sec	7.5 sec	5 sec	5.8 sec	3.1 sec	3 sec

For technical reasons, we used several built-in arithmetic operators in Elan when defining quotient/reminder.

A relatively good computation speed-up is obtained for the odd/even example. The compact transformation, overcame the speed of computation of the original CTRS in all our experiments. Both Maude and Elan are sequential; we actually expect our transformation to be significantly better on parallel rewrite engines.

7 Discussion and Future Work

We have presented a sound transformation of a CTRS into a TRS, which is also complete when the original CTRS is confluent. A conjecture is stated which, if true, would allow one to use the TRS as a computationally equivalent variant of the original CTRS. Since unconditional rewriting is much easier to implement efficiently and can take immediate full advantage of parallel computer architectures, the presented transformation is expected to lead to very efficient implementations of conditional rewriting. We showed experimentally that significant increases in speed can be achieved even on the current rewrite engines if our transformation is used as a front-end.

One should *not* use our transformation for equational theorem proving, because it is not sound for this purpose. Indeed, consider the equational variant of Example 7. Then one can deduce $\{\overline{false}\} = \overline{f}(\{\overline{false}\}, \{\overline{false}\}) = \{\overline{f}(\overline{false}, \bot)\}$ in the transformed specification, which has no counterpart in the original specification. However, if the original equational specification is terminating and confluent as a rewrite system, then one can use the transformed system to prove equalities in the original system by reducing them to their normal forms.

We have not considered here rewrite systems *modulo equations*, such as associativity, commutativity and/or idempotency; extending our transformation to such CTRSs is expected to be a non-trivial task. To keep the presentation simple, we only considered normal 1-CTRSs. Nevertheless, we believe that our transformation can be easily accommodated to other types of CTRS, by appropriately transforming the CTRS. Techniques to compact the generated TRS are also worthwhile investigating in detail (see the discussion at the end of Section 3.3). Finally, one may have noticed that the rules for up-propagating $\{\_\}$ can destroy useful partial reductions. It would be very useful to adapt our transformation to restart only the conditions that are invalidated when a rewrite step occurred.

We are very grateful to Andrei Popescu for several technical suggestions that helped smoothing the presentation of the results in this paper. We also thank Claude Marché for referring us to Example 3.

References

1. S. Antoy, B. Brassel, and M. Hanus. Conditional narrowing without conditions. In *5th ACM SIGPLAN international conference on Principles and practice of declaritive programming (PPDP'03)*, pages 20–31. ACM Press, 2003.
2. J. Bergstra and J. Klop. Conditional rewrite rules: Confluence and termination. *Journal of Computer and System Sciences*, 32(3):323–362, 1986.
3. P. Borovansky, H. Cirstea, H. Dubois, C. Kirchner, H. Kirchner, P. Moreau, C. Ringeissen, and M. Vittek. *ELAN: User Manual*, 2000. Loria, Nancy, France.
4. 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`.
5. N. Dershowitz and M. Okada. A rationale for conditional equational programming. *Theoretical Computer Science*, 75:111–138, 1990.
6. N. Dershowitz, M. Okada, and G. Sivakumar. Cannonical conditional rewrite systems. In *9th Conference on Automated DeductionTheoretical Computer Science*, volume 310 of *LNCS*, pages 538–549. Springer, 1988.
7. R. Diaconescu and K. Futatsugi. *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*. World Scientific, 1998. AMAST Series in Computing, volume 6.
8. E. Giovannetti and C. Moiso. Notes on the elimination of conditions. In *1st International Workshop on Conditional Term Rewriting Systems (CTRS'87)*, volume 308 of *LNCS*, pages 91–97. Springer, 1987.
9. J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OBJ. In *Software Engineering with OBJ: algebraic specification in action*, pages 3–167. Kluwer, 2000.
10. M. Hanus. The integration of functions into logic programming: From theory to practice. *The Journal of Logic Programming*, 19 & 20:583–628, 1994.
11. C. Hintermeier. How to transform canonical decreasing ctrss into equivalent canonical trss. In *4th International Workshop on Conditional and Typed Rewriting Systems (CTRS'94)*, volume 968 of *LNCS*, pages 186–205, 1994.
12. M. Marchiori. Unravelings and ultra-properties. In *5th International Conference on Algebraic and Logic Programming*, volume 1139 of *LNCS*, pages 107–121. Springer, 1996.
13. A. Middeldorp and E. Hamoen. Completeness results for basic narrowing. *Journal of Applicable Algebra in Eng., Communication and Computing*, 5:313–353, 1994.
14. E. Ohlebusch. *Advanced Topics in Term Rewriting*. Springer, 2002.
15. J. C. Reynolds. The discoveries of continuations. *LISP and Symbolic Computation*, 6(3–4):233–247, 1993.
16. G. Roşu. From conditional to unconditional rewriting. In *Recent Trends in Algebraic Development Techniques; Proceedings with selected papers presented at WATD'04*. Lecture Notes in Computer Science, to appear, 2004.
17. P. Viry. Elimination of conditions. *Journal of Symbolic Computation*, 28:381–401, Sept. 1999.

We added this appendix for reviewers' convenience. If the paper is accepted, the appendix will be removed and an extended technical report will be made available and cited from the 15-page proceedings paper.

A Selected Proofs

In proofs, we will denote by $\underline{t}$ the linear variant of $\widetilde{t}^X$, that is the one also replacing the variables of t with fresh ones, giving distinct variables for distinct occurrences of the same variable.

We will also use the notion of *multi-context rewriting*, which allows a context γ to have more that one occurrence of $\star$. We let $\Rightarrow_{\mathcal{R}}$ denote the relation associated with multi-context rewriting. Note that $\rightarrow_{\mathcal{R}}^* = \Rightarrow_{\mathcal{R}}^*$.

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 $\overline{\mathcal{R}}$.*
6. *Let t' be a reachable term and s be a Σ-term such that $\{\overline{s}\} \Rightarrow_{\overline{\mathcal{R}}}^{k} \{t'\}$. Let t_t be a Σ-term with variables and θ_t a $\overline{\Sigma}$-substitution giving for any variable in t_t a subterm of t' on a structural position. Then there exists a Σ-term s_t such that $\{\overline{s_t}\} \Rightarrow_{\overline{\mathcal{R}}}^{k'} \{\theta_t(\overline{t_t})\}$ with $k' \leq k$.*
7. *Let t' be a reachable term and t be a Σ-term such that $\{\overline{t}\} \Rightarrow_{\overline{\mathcal{R}}}^{k} \{t'\}$. Let s' be a subterm of t' in a structural position. Then there exists a Σ-term s such that $\{\overline{s}\} \Rightarrow_{\overline{\mathcal{R}}}^{k'} \{s'\}$ and $k' \leq k$.*

Proof. 1. We will prove that for all $\overline{\Sigma}$-terms t' and all structural positions α in t', the subterm s' at position α is also structural by induction over the length of α. If α is empty, then $s' = t'$ and thus it is structural. Now suppose that $\alpha = i\alpha'$. If $t' = \{t''\}$ then i is 1; since t'' is structural we apply the induction hypothesis for t'' and α' getting that s' is structural. Otherwise, assume that $t' = \overline{\sigma}(t'_1, \ldots, t'_n, C_1, \ldots, C_{k_\sigma})$ with $\sigma \in \Sigma_n$ and $t'_1, \ldots, t'_n$ structural, then apply the induction hypothesis for t'_i and α' getting that s' is structural.

2. Let $\mathcal{X}$ be a set of variables, t' a structural term with variables from X and θ a substitution from $\mathcal{X}$ to structural terms. We'll prove that $\theta(t)$ is structural by induction on the structure of t'. If t' is a variable then it's obvious, since θ' substitutes variables by structural terms. Now suppose that $t' = \overline{\sigma}(t'_1, \ldots, t'_n, C_1, \ldots, C_{k_\sigma})$. Applying the induction hypothesis for each t'_i, $1 \leq i \leq n$. we get that $\theta'(\overline{t'_i})$ is structural. Then $\theta(t')$ must also be structural since

$$\theta(t') = \theta(\overline{\sigma}(t'_1, \ldots, t'_n, C_1, \ldots, C_{k_\sigma})) = \overline{\sigma}(\theta'(t'_1), \ldots, \theta'(t'_n), C_1, \ldots, C_{k_\sigma})$$

and the latter is obviously structural. Finally, if $t' = \{t'_1\}$ we can apply induction hypothesis for t'_1 and get that $\theta'(\overline{t'_1})$ is structural. Then obviously $\theta(\{t'_1\})$ must also be structural.

3. Let s' be a structural term, let t' a $\overline{\Sigma}$-term such that $s' \rightarrow_{\overline{\mathcal{R}}} t'$ and let α be the position of s' where the rewriting step occurred. If α is a non-structural position then t' must also be structural, since the definition of structural terms is done using only structural positions. Let us now prove that for any structural s' and any $\overline{\Sigma}$-term t' such that $s' \rightarrow_{\overline{\mathcal{R}}} t'$ the rewriting step occurring at the structural position α we have that t' is also structural. We do that by induction on the length of α. If $\alpha = i\alpha'$ then either $s' = \{s'_1\}$, $i = 1$ and $t' = \{t'_1\}$ or $s' = \overline{\sigma}(s'_1, \ldots, s'_n, C_1, \ldots, C_{k_\sigma})$, $1 \leq i \leq n$ and $t' = \overline{\sigma}(s'_1, \ldots, s'_{i-1}, t'_i, s'_{i+1}, \ldots, s'_n, C_1, \ldots, C_{k_\sigma})$. Also, we must have that s'_i is structural and $s'_i \rightarrow_{\overline{\mathcal{R}}} t'_i$ using the same rule and the same substitution a position α'. Applying the induction hypothesis for we get that t'_i is structural, whence t' is structural. If α is the empty word, then let $l' \rightarrow r'$ be the rule used and θ' be the substitution used. If $l' \rightarrow r'$ is of form:
   - $\{\{x\}\} \rightarrow \{x\}$ then since structural terms are closed under $\mathcal{SS}$, $\theta'(x)$ is structural, whence $t' = \{\theta'(x)\}$ is also structural;
   - $\overline{\sigma}(x_1, \ldots, x_{i-1}, \{x_i\}, x_{i+1}, \ldots, x_n, C_1, \cdots, C_{k_\sigma}) \rightarrow$ $\rightarrow \{\overline{\sigma}(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n, \bot, \cdots, \bot)\}$ then since variables x_j, $1 \leq j \leq n$ are in structural positions, $\theta'(x_j)$ is structural and we apply that structural terms are closed under $\overline{C\Sigma}$ for $\sigma(x_1, \ldots, x_n)$;
   - $\widetilde{l}_{i/\bot} \rightarrow \widetilde{l}_{i/\{\{\overline{cl}\}}$, then we don't have anything to prove since nothing changes on structural positions;
   - $\widetilde{l}_{i/\{cr\}} \rightarrow \{\overline{r}\}$ or $\widetilde{l} \rightarrow \{\overline{r}\}$, then since all variables in l are on structural positions in $\widetilde{l}$ we have that $\theta'(x)$ is structural for each variable x occurring in l then we can apply that structural terms are closed under $\overline{C\Sigma}$ for r.
4. $\overline{s}$ is structural and structural terms are closed under $\overline{\mathcal{R}}$.
5. Obvious from the definition of reachable terms.
6. We prove the affirmation by well founded induction on k. Let t_t be a Σ-term with variables from $X_t = \{x_1, \ldots, x_n\}$ and θ_t a $\overline{\Sigma}$-substitution with variables from X_t such that $\theta_t(x_i) = t'_i$ for each $1 \leq i \leq n$. Then there exists a Σ-term s_t such that $\{\overline{s_t}\} \Rightarrow^{k'}_{\overline{\mathcal{R}}} \{\theta_t(\overline{t_t})\}$ with $k' \leq k$.

   Let s' be a $\overline{\Sigma}$-term such that $\{\overline{s}\} \Rightarrow^{k-1}_{\overline{\mathcal{R}}} \{s'\} \Rightarrow_{\overline{\mathcal{R}}} \{t'\}$. We will show that there exist a Σ-term t_s with variables from X_s and a $\overline{\Sigma}$-substitution θ_s giving for each variable in X_s a subterm of s' on a structural position and that $\theta_s(\overline{t_s}) = \theta_t(\overline{t_t})$ or $\theta_s(\overline{t_s}) \Rightarrow_{\overline{\mathcal{R}}} \theta_t(\overline{t_t})$.

   Let c be a $\overline{\Sigma}$ multi-context, $l' \rightarrow r'$ be a rule in $\overline{\mathcal{R}}$ with variables in a set X and θ be a $\overline{\Sigma}$-substitution such that $s' = c[\theta(l')]$ and $t' = c[\theta(r')]$. Let us build the set X_s, a substitution θ' giving for any variable in X_t a Σ-term with variables form $X \cup X_t$ and a set of positions $\mathcal{P}$. For any $x \in X_t$, perform the following operation. Let β be the position of $\theta_t(x)$ in t'.
   - If β is a position in c then
     - if $c/_\beta$ doesn't contain any variables, then $s'/_\beta = t'/_\beta$. Add x to X_s and let $\theta'(x) = x$.

- otherwise, we have that $s'/_\beta = c/_\beta[\theta(l')]$ and $t'/_\beta = c/_\beta[\theta(r')]$. Add then x to X_s, let $\theta'(x) = x$ and for each position γ of x in t_t and each position δ of a variable in c/β add to $\mathcal{P}$ the position $\gamma\delta$.

– If β is not a position in c then
  - If $\beta = \alpha\gamma\delta$ where α is a position of a variable in c and γ is a position of a variable y in r', then let γ' be a position of y in l'. We have that $s'/_{\alpha\gamma'\delta} = t'/_{\alpha\gamma\delta}$. Add x to X_s and let $\theta'(x) = x$.
  - If $\beta = \alpha\gamma$ where α is a position of a variable in c and γ is a non-empty non-variable position in r', then let's analyze the possible cases for $l' \to r'$:
    (a) $\{\{x\}\} \to \{x\}$
    (b) $\overline{\sigma}(x_1, \ldots, x_{i-1}, \{x_i\}, x_{i+1}, \ldots, x_n, C_1, \ldots, C_{k_\sigma}) \to \{\overline{\sigma}(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n, \bot, \ldots, \bot)\}$
    (c) $\widetilde{l}_{i/\bot} \to \widetilde{l}_{i/C}$, where C is a condition
    (d) $\widetilde{l}_{i/\{cr\}} \to \{\overline{r}\}$
    (e) $\widetilde{l} \to \{\overline{r}\}$
    (a) is impossible since the only position in $\{x\}$ are the one of x and the empty one. Since (c) changes only an auxiliary parameter of the operation on the top of $\widetilde{l}$ we will have that $s'_\beta = t'_\beta$. Then add x to X_s and let $\theta'(x) = x$.
    For (b), (d), and (e) we will have that $r'_\beta = \overline{u}$ for some Σ-term u with variables form X. Add then $var(u)$ to X_s and let $\theta'(x) = u$.

  Let $t_s = \theta'(t_t)$ and let θ_s be defined as follows:

$$\theta_s(x) = \begin{cases} \theta_t(x) \text{ if } x \in X_t \\ \theta(x) \text{ if } x \in X \end{cases}$$

  We then have that for any $x \in X_s$, $\theta_s(x)$ is a subterm of s' on a structural position. Also if $\mathcal{P}$ is empty then $\theta_s(t_s) = \theta_t(t_t)$; otherwise $\theta_s(t_s) \Rightarrow_{\overline{\mathcal{R}}} \theta_t(t_t)$ using rule and substitution θ at any position in $\mathcal{P}$.
  Applying the induction hypothesis, the proposition is proved.

7. Apply 6 for $t_t = x$ and $\theta_t(x) = s'$.

Proposition 2. ***Soundness.*** *If $s, t \in T_\Sigma$ and $s \to^*_{\mathcal{R}} t$ then $\{\overline{s}\} \to^*_{\overline{\mathcal{R}}} \{\overline{t}\}$. In particular, if fn(t) is a normal form of t in $\mathcal{R}$ then $\{\overline{t}\} \to^\star_{\overline{\mathcal{R}}} \{\overline{fn(t)}\}$.*

Proof. Rewriting relation can be defined as the least relation closed under R (reflexivity), T (transitivity), $C\Sigma$ (compatibility with the operations) and $Sub_{\mathcal{R}}$ ($\mathcal{R}$-substitution). We will show that the relation:

$$D = \{(s, t) \in T_\Sigma \times T_\Sigma \mid \{\overline{s}\} \to^*_{\overline{\mathcal{R}}} \{\overline{t}\}\}$$

is closed under the above rules whence it contains $\to^*_{\mathcal{R}}$. Closure under reflexivity and transitivity is obvious. For proving the closure under $Sub_{\mathcal{R}}$ let us first formally describe it:

> For any rule $l \to r$ if $cl = cr \in \mathcal{R}$ and any Σ-substitution θ such that $(\theta(cl), cr) \in D$ we have that $(\theta(l), \theta(r)) \in D$.

Let $\rho_{\sigma,i} : l \to r$ if $cl = cr$ be a rule in $\mathcal{R}$, θ be a Σ-substitution such that $(\theta(cl), cr) \in D$. Consider the $\overline{\Sigma}$-substitution θ' defined as $\theta'(x) = \overline{\theta(x)}$. for any variable x occurring in $\rho_{\sigma,i}$ and $\theta'(x) = \bot$ for any new variable occurring in $\overline{\rho_{\sigma,i}}$. We then have that $\theta'(\overline{t}) = \overline{\theta(t)}$ for $t \in \{l, r, cl\}$. We then can apply rule $\overline{\rho_{\sigma,i}}$ on $\{\overline{\theta(l)}\}$ using the substitution θ' and the context $\{\cdot\}$, getting $\{\overline{\theta(l)}_{i/\{\overline{cl}\}}\}$. Now we use that $(\theta(cl), cr) \in D$ which means that $\{\overline{\theta(cl)}\} \to^*_{\overline{\mathcal{R}}} \{cr\}$, whence

$$\{\overline{\theta(l)}_{i/\{\overline{\theta(cl)}\}}\} \to^*_{\overline{\mathcal{R}}} \{\overline{\theta(l)}_{i/\{cr\}}\} \to^*_{\overline{\mathcal{R}}} \{\overline{\theta(r)}\}$$

Finally let us prove that D is closed under $C\Sigma$, that is: if $(s, t) \in D$ then for any operation $\sigma \in \Sigma_n$, for any Σ-terms t_j, $1 \leq j \leq n$ and for any $1 \leq i \leq n$ we have that: $(\sigma(t_1, \ldots, t_{i-1}, s, t_{i+1}, \ldots, t_n), \sigma(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_n)) \in D$. In order to prove it we first have to observe that for any $\overline{\Sigma}$-terms s' and t', if $\{s'\} \to^*_{\overline{\mathcal{R}}} \{t'\}$, then $s' \to^*_{\overline{\mathcal{R}}} t'$ or $s' \to^*_{\overline{\mathcal{R}}} \{t'\}$. This is obvious since the only rule containing $\{\cdot\}$ which has effect on the term in the brackets is that dissolving another bracket.

Since $\overline{\sigma(t_1, \ldots, t_{i-1}, s, t_{i+1}, \ldots, t_n)} = \overline{\sigma}(\overline{t_1}, \ldots, \overline{t_{i-1}}, \overline{s}, \overline{t_{i+1}}, \ldots, \overline{t_n}, \bot, \ldots, \bot)$, it follows that one can rewrite $\{\overline{\sigma(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_n)}\}$ to one of this two: $\{\overline{\sigma}(\overline{t_1}, \ldots, \overline{t_{i-1}}, \overline{t}, \overline{t_{i+1}}, \ldots, \overline{t_n}, \bot, \ldots, \bot)\}$ or $\{\overline{\sigma}(\overline{t_1}, \ldots, \overline{t_{i-1}}, \{\overline{t}\}, \overline{t_{i+1}}, \ldots, \overline{t_n}, \bot, \ldots, \bot)\}$. In the first case our proof is complete, in the second we just need to apply the rule for propagating $\{\cdot\}$ up and then the one for dissolving one $\{\cdot\}$ at the top.

Theorem 1. *Suppose that $\mathcal{R}$ is ground confluent. If s' and t' are reachable terms such that $s' \to^\star_{\overline{\mathcal{R}}} t'$. then $\widehat{s'} \to^*_{\mathcal{R}} \widehat{t'}$. In particular, our transformation is sound and complete, i.e., $s \to^*_{\mathcal{R}} t$ iff $\{\overline{s}\} \to^\star_{\overline{\mathcal{R}}} \{\overline{t}\}$ for any $s, t \in T_\Sigma$.*

Proof. Since $\to^*_{\overline{\mathcal{R}}} = \Rightarrow^*_{\overline{\mathcal{R}}}$ and $\to^*_{\mathcal{R}} = \Rightarrow^*_{\mathcal{R}}$ we can change the affirmation in the proposition the following way:

Let s', t' be reachable terms such that $s' \Rightarrow^*_{\overline{\mathcal{R}}} t'$. Then $\widehat{s'} \Rightarrow^*_{\mathcal{R}} \widehat{t'}$.

Since s' is reachable there is some Σ-term s such that $\{\overline{s}\} \to^*_{\overline{\mathcal{R}}} \{s'\}$ equivalent to $\{\overline{s}\} \Rightarrow^*_{\overline{\mathcal{R}}} \{s'\}$. We will prove that for any Σ-term s and any $\overline{\Sigma}$ terms s' and t' such that $\{\overline{s}\} \Rightarrow^p_{\overline{\mathcal{R}}} \{s'\} \Rightarrow^k_{\overline{\mathcal{R}}} \{t'\}$ we have that $\widehat{s'} \Rightarrow^*_{\mathcal{R}} \widehat{t'}$, by well founded induction over $m = p + k$.

If $k = 0$ then $s' = t'$ and there is nothing more to prove. If $k \neq 0$ let s'_t be a $\overline{\Sigma}$-term such that $s' \Rightarrow^{k-1}_{\overline{\mathcal{R}}} s'_t \Rightarrow_{\overline{\mathcal{R}}} t'$. We then can apply the induction hypothesis for s, s' and s'_t and get that $\widehat{s'} \Rightarrow^*_{\mathcal{R}} \widehat{s'_t}$. It suffices now to show that $\widehat{s'_t} \Rightarrow^*_{\mathcal{R}} \widehat{t'}$.

If $\widehat{s'_t} = \widehat{t'}$ then our proof is done.If $\widehat{s'_t} \neq \widehat{t'}$ then it must be the case that was applied either a rule of the form $\widetilde{l} \to \{\overline{r}\}$ or one of the form $\overline{\rho}'_{\sigma,i} : \widetilde{l}_{i/cr} \to \{\overline{r}\}$ using a substitution θ and a multi-context c having at least one variable in a structural position. If a rule of the first type was applied, $l \to r$ can also be applied to $\widehat{s'_t}$ using the context $\widehat{c}$ and substitution $\widehat{\theta}$ to obtain $\widehat{t'}$.

In order to prove the affirmation in the latter case, we will use the following lemma

Lemma 1. *Let s'_t be a reachable term and s be a Σ-term such that $\{\overline{s}\} \Rightarrow^{k}_{\overline{\mathcal{R}}} \{s'_t\}$. Let β_t be a structural position in s'_t, $\rho_{\sigma,i} : l \to r$ if $cl = cr$ a rule in $\mathcal{R}$ and θ_t a $\overline{\Sigma}$ substitution such that $s'_t/_{\beta_t} = \theta_t(\widetilde{l}_{i/\{cr\}})$.*

Then there exists a reachable term $s'_\perp$ such that $\{\overline{s}\} \Rightarrow^{k_s}_{\overline{\mathcal{R}}} \{s'_\perp\} \Rightarrow^{k-k_s}_{\overline{\mathcal{R}}} \{s'_t\}$ and a substitution $\theta_\perp$ such that $\theta_\perp(\widetilde{l}_{i/\perp})$ is a subterm of $s'_\perp$ on a structural position and $\theta_\perp(\widetilde{l}_{i/\perp}) \Rightarrow^{k'}_{\overline{\mathcal{R}}} \theta_t(\widetilde{l}_{i/\{cr\}})$ with $k' \leq k - k_s$ and all terms in the rewriting sequence matching $\underline{l}$ but only the first matching $\underline{l}_{i/\perp}$.

Proof. Let us first show the following:

> Let s' and t' be two structural terms such that $s' \Rightarrow_{\overline{\mathcal{R}}} t'$ using the rule $l' \to r' \in \overline{\mathcal{R}}$, the multi-context c and the substitution θ. Let β be a structural position in t', $\rho_{\sigma,i} : l \to r$ if $cl = cr$ a rule in $\mathcal{R}$ and θ'' a $\overline{\Sigma}$ substitution such that $t'/_\beta = \theta''(\underline{l}_{i/C})$ where C is a variable and $\theta''(C) \neq \perp$.
> Then there exists a structural position α of s' and a $\overline{\Sigma}$-substitution θ' such that $s'/_\alpha = \theta'(\underline{l})$ and $\theta'(\underline{l}) = \theta''(\underline{l})$ or $\theta'(\underline{l}) \Rightarrow_{\overline{\mathcal{R}}} \theta''(\underline{l})$.

If β is a position in c then if $c/_\beta$ doesn't contain any variables, we can consider $\alpha = \beta$, $\theta' = \theta''$; otherwise, for any variable x of $\underline{l}$ on a structural position γ_x we must have that $\beta\gamma_x$ is also a position in c (otherwise the structure of $\underline{l}$ would have changed). We can then take $\alpha = \beta$ and $\theta'(x) = \theta''(c/_{\beta\gamma_x})$.

If β is not a position in c, then $\beta = \beta'\gamma$ where β' is a position of a variable in c. Then γ must be of the form $\gamma = \delta\epsilon$ where δ is a position of a variable x in r' (because otherwise we couldn't have $\theta''(C) \neq \perp$). Let then δ' be a position in l' of the same variable x, and take $\alpha = \beta'\delta'\epsilon$ and $\theta' = \theta''$.

We can now apply the statement above repeatedly as long as its hypothesis is satisfied, meaning as long as $\theta''(C) \neq \perp$. We know that it must reach an $\perp$ in at most k steps, since $\overline{s}$ has nothing but $\perp$s on any non-structural position. Let $s'_\perp$ be the term where the above proposition cannot be applied anymore and k_s its position in the rewriting sequence $\overline{s} \Rightarrow^{k}_{\overline{\mathcal{R}}} t'$. We have thus obtained a sequence of terms $l'_1 \Rightarrow_{\overline{\mathcal{R}}} l'_2 \Rightarrow_{\overline{\mathcal{R}}} \ldots \Rightarrow_{\overline{\mathcal{R}}} l'_{k'}$ with $k' \leq k - k_s$ and all terms matching $\underline{l}$. Furthermore, $l'_{k'} = \theta_t(\widetilde{l}_{i/\{cr\}})$ and l'_1 is the only one in sequence matching $\underline{l}_{i/\perp}$. Since also $l'_1 \Rightarrow_{\overline{\mathcal{R}}} l'_2$ we must have that l'_1 matches $\widetilde{l}_{i/\perp}$ since this is the only way to change the ith auxiliary variable of σ from $\perp$ to something different from $\perp$ - by applying the $\overline{\rho}_{\sigma,i}$ rule.

Applying the lemma we get a $\overline{\Sigma}$-term $s'_\perp$ such that $\{\overline{s}\} \Rightarrow^{k_s}_{\overline{\mathcal{R}}} \{s'_\perp\} \Rightarrow^{k-k_s}_{\overline{\mathcal{R}}} \{t'\}$ and substitution $\theta_\perp$ such that $\theta_\perp(\widetilde{l}_{i/\perp})$ is a subterm of s' on a structural position and $\theta_\perp(\widetilde{l}_{i/\perp}) \Rightarrow^{k'}_{\overline{\mathcal{R}}} \theta_t(\widetilde{l}_{i/\{cr\}})$ with $k' \leq k - k_s$ and all terms in the rewriting sequence matching $\underline{l}$ but only the first matching $\underline{l}_{i/\perp}$.

Since $\theta_\perp(\widetilde{l}_{i/\perp}) \Rightarrow^{k'}_{\overline{\mathcal{R}}} \theta_t(\widetilde{l}_{i/\{cr\}})$ it must be that $\theta_\perp(\{\overline{cl}\}) \Rightarrow^{\{}_{\overline{\mathcal{R}}} k''\}\{cr\}$ with $k'' \leq k'$. Since all variables of the rule are replaced by $\theta_\perp$ with subterms of $\theta_\perp(\widetilde{l}_{i/\perp})$ on structural positions we can use Proposition 1.6 to verify the the

induction hypothesis for $\theta_\perp(\overline{cl})$, thus obtaining that

$$\widehat{\theta_\perp(\overline{cl})} \Rightarrow^*_{\mathcal{R}} cr \tag{1}$$

Since the lemma assures us that all the rewriting steps from $\theta_\perp(\widetilde{l}_{i/\perp})$ to $\theta_t(\widetilde{l}_{i/\{cr\}})$ which occur on structural positions don't occur on non-variable positions of l, we also have that for each variable x occurring in l $\theta_\perp(x) \Rightarrow^{k_x}_{\overline{\mathcal{R}}} \theta_t(x)$ with $k_x \leq k'$. Also, we can use Proposition 1.7 to verify the induction hypothesis and get $\widehat{\theta_\perp(x)} \Rightarrow^*_{\mathcal{R}} \widehat{\theta_t(x)}$. Define now $\widehat{\theta_\perp}$ and $\widehat{\theta_t}$ to be Σ-substitutions such that $\widehat{\theta_\perp}(x) = \widehat{\theta_\perp(x)}$ and $\widehat{\theta_t}(x) = \widehat{\theta_t(x)}$ for each variable x occurring in l. We then have that: $\widehat{\theta_\perp}(cl) \Rightarrow^*_{\mathcal{R}} \widehat{\theta_t}(cl)$. and using Equation (1) and the confluence of $\mathcal{R}$ we get that $\widehat{\theta_t}(cl) \rightarrow^*_{\overline{\mathcal{R}}} cr$. Since $\widehat{s'_n} = \widehat{c}[\widehat{\theta_t}(l)]$ we can apply rule $\rho_{\sigma,i}$ and get $\widehat{c}[\widehat{\theta_t}(r)]$ which is exactly $\widehat{t'}$.

Proposition 3. *Let t be a Σ-term which is a strong normal form w.r.t $\mathcal{R}$. Then there exists a normal form $fn(\overline{t})$ w.r.t. $\overline{\mathcal{R}}$ such that*

1. $\overline{t} \rightarrow^*_{\overline{\mathcal{R}}} fn(\overline{t})$
2. *$fn(\overline{t})$ has no brackets at top.*
3. $\widehat{fn(\overline{t})} = t$, *meaning that the structure of the normal form is the same.*

Proof. We prove the statement by induction on the number of operations (matching and rewriting steps) needed to verify that t is in the normal form, that means the steps required for trying and failing all conditional rules for t.

One can observe that there exists a Σ-term s with variables, $\mathcal{X} = var(s)$, Σ-substitution θ such that $t = \theta(s)$ and for any $x \in \mathcal{X}$, $\theta(x)$ matches at least one lhs of a conditional rule and all matches of any lhs on any conditional rule don't occur in any position of t being a non-variable position of s.

First, if $\mathcal{X} = \emptyset$ then none of the lhs terms of the rules in $\mathcal{R}$ matches any subterm of t whence it is also the case that none of the rules in $\overline{\mathcal{R}}$ can be applied on $\overline{t}$ and this means exactly that $\overline{t}$ is in normal form and we can take $fn(\overline{t})$ to be exactly $\overline{t}$, satisfying all the requirements.

If $\mathcal{X}$ has more than one element then let $x_1, \ldots, x_n$ be the elements of $\mathcal{X}$ ($n \geq 2$), Then, since any $\theta(x_i)$ needs at least one less matching than t we can apply the induction hypothesis for them; define a $\overline{\Sigma}$-substitution θ' such that $\theta'(x_i) = fn(\overline{\theta(x_i)})$ for each $1 \leq i \leq n$. Also consider $\overline{\Sigma}$-substitution $\overline{\theta}(x_i) = \overline{(\theta(x_i))}$. we the have that $\overline{\theta}(x_i) \rightarrow^*_{\overline{\mathcal{R}}} \theta'(x_i)$, and since $\overline{t} = \overline{\theta(s)} = \overline{\theta}(s)$ we get that $\overline{t} \rightarrow^*_{\overline{\mathcal{R}}} \theta'(s)$ which is in normal form since there are no brackets on top of any $fn(\overline{s_i})$ and also the structure of their normal form stays the same and we know that we had no match in $\overline{t}$ above their top position.

Finally, $\mathcal{X}$ has exactly one element x then we must have a subterm s of t such that we have a matching of a lhs at top of s and any matching of a lhs in t occurs in s. Let then $\rho_{\sigma,i} : l \rightarrow r$ if $cl = cr$ be a rule in $\mathcal{R}$ and θ be a substitution such that $\theta(l) = s$. Then, as in proof of Proposition 2 we will obtain that $\overline{s} \rightarrow_{\overline{\mathcal{R}}} \overline{s}_{i/\{\overline{\theta(cl)}\}}$. Also, from Corollary 2 we deduce that this can be further

rewritten to $\overline{s}_{i/\{\overline{fn(\theta(cl))}\}}$ Now, in order to show that t is in normal form we have to verify that rule $\rho_{\sigma,i}$ fails on top of s, that meaning reducing $\theta(cl)$ its normal form including trying and failing all rules for its normal form. Thus we need less operations to prove $\theta(cl)$ is in the normal form and we can apply the induction hypothesis for it, leading us to $\overline{s}_{i/\{fn(\overline{fn(cl))}\}}$. Since $fn(cl) \neq cr$ the rule $\overline{\rho_{\sigma,i}}'$ cannot be applied.

After applying the above steps for all matching lhs at top of s we get $\overline{s} \rightarrow^*_{\overline{\mathcal{R}}} \overline{\sigma}(\overline{s_1}, \ldots, \overline{s_n}, C_1, \ldots, C_{k_\sigma})$ where all of the C_js are in a normal form different form their corresponding $\{cr\}$ and there is no rule that can be applied at the top of the expression anymore. Now we apply the induction hypothesis for all s_is getting to $\overline{\sigma}(fn(\overline{s_1}), \ldots, fn(\overline{s_n}), C_1, \ldots, C_{k_\sigma})$ which is in normal form,and using the reasoning above for $t = c[s]$ we get

$$fn(t) = c[\overline{\sigma}(fn(\overline{s_1}), \ldots, fn(\overline{s_n}), C_1, \ldots, C_{k_\sigma})].$$

B Maude sources

In the following maude-sources, $\star$ denotes $\bot$.

Odd/Even

Conditional

```
fmod ODD-ECOND is
sort S .
ops 0 true false : -> S .
op s : S -> S .
ops odd even : S -> S .
var x : S .
eq odd(0) = false .
ceq odd(s(x)) = true if even(x) = true .
ceq odd(s(x)) = false if even(x) = false .
eq even(0) = true .
ceq even(s(x)) = true if odd(x) = true .
ceq even(s(x)) = false if odd(x) = false .
endfm

red odd(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(0)))))))))))))))))))))) .
---here are indeed 22s successors
```

Unconditional

```
fmod ODD-EUCOND is
sort S .
ops 0 true false * : -> S .
op s : S -> S .
op {_} : S -> S .
ops odd even : S S S -> S .
var x c1 c2 : S .
eq odd(0,c1,c2) = {false} .
eq odd(s(x),*,c2) = odd(s(x),{even(x,*,*)},c2) .
eq odd(s(x),{true},c2) = {true} .
eq odd(s(x),c1,*) = odd(s(x),c1,{even(x,*,*)}) .
eq odd(s(x),c1,{false}) = {false} .
eq even(0,c1,c2) = {true} .
eq even(s(x),*,c2) = even(s(x),{odd(x,*,*)},c2) .
eq even(s(x),{true},c2) = {true} .
eq even(s(x),c1,*) = even(s(x),c1,{odd(x,*,*)}) .
eq even(s(x),c1,{false}) = {false} .

eq {{x}} = {x} .
```

```
eq s({x}) = {s(x)} .
eq odd({x},c1,c2) = {odd(x,*,*)} .
eq even({x},c1,c2) = {even(x,*,*)} .
endfm

red {odd(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(0))))))))))))))))))),*,*)} .
```

Compact Unconditional

```
fmod ODD-EUCOND1 is
protecting NAT .
sort S .
ops 0 true false * : -> S .
op s : S -> S .
op {_} : S -> S .
ops odd even : S S -> S .
var x c1 : S .
eq odd(0,c1) = {false} .
eq odd(s(x),*) = odd(s(x),{even(x,*)}) .
eq odd(s(x),{true}) = {true} .
eq odd(s(x),{false}) = {false} .
eq even(0,c1) = {true} .
eq even(s(x),*) = even(s(x),{odd(x,*)}) .
eq even(s(x),{true}) = {true} .
eq even(s(x),{false}) = {false} .

eq {{x}} = {x} .
eq s({x}) = {s(x)} .
eq odd({x},c1) = {odd(x,*)} .
eq even({x},c1) = {even(x,*)} .
endfm


red {odd(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(0))))))))))))))))))),*)} .
```

Quotient/Reminder

Here we have used Maude's internal built natural numbers in order perform testing for bigger numbers. Nevertheless, we implemented the difference and the less predicate. Therefore we have given them different notations ($\cdot - -\cdot$ and $\cdot less \cdot$) to avoid clashing with built-in ones

Conditional

```
fmod QUOREM-ECOND is
```

```
including NAT .
sort S .
subsort Nat < S .
ops true false : -> S .
op quorem : S S -> S .
op <_,_> :  S S  -> S .
op  _--_ : S S -> S .
op _less_ : S S -> S .
op s' : S -> S .
vars x y q r : S .
eq x less 0 = false .
eq 0 less s(x) = true .
eq s(x) less s(y) = x less y .

eq 0 -- s(y) = 0 .
eq x -- 0 = x .
eq s(x) -- s(y) = x -- y .

eq s'(< x,y >) = < s(x),y > .

eq quorem(0,s(y)) = < 0,0 > .
ceq quorem(s(x),s(y)) = < 0,s(x) >
  if x less y = true .
ceq quorem(s(x),s(y)) = s'(quorem(x -- y,s(y)))
  if x less y = false .
endfm

red quorem(1275000,130) .
```

Unconditional

```
fmod QUOREM-EUCOND is
including NAT .
sort S .
subsort Nat < S .
ops true false * : -> S .
op quorem : S S S S -> S .
op <_,_> :  S S  -> S .
op {_} : S -> S .
op _--_ : S S -> S .
op _less_ : S S -> S .
op s' : S -> S .
vars x y q r : S .
eq x less 0 = {false} .
eq 0 less s(x) = {true} .
eq s(x) less s(y) = {x less y} .
```

```
vars c1 c2 : S .
eq 0 -- s(y) = {0} .
eq x -- 0 = {x} .
eq s(x) -- s(y) = {x -- y} .
eq s'(< x,y >) = {< s(x),y >} .
eq quorem(0,s(y),c1,c2) = {< 0,0 >} .
eq quorem(s(x),s(y),*,c2) = quorem(s(x),s(y),{x less y},c2) .
eq quorem(s(x),s(y),c1,*) = quorem(s(x),s(y),c1,{x less y}) .
eq quorem(s(x),s(y),{true},c2) = {< 0,s(x) >} .
eq quorem(s(x),s(y),c2,{false}) = {s'(quorem(x -- y,s(y),*,*))} .

eq {x} -- y = {x -- y} .
eq x -- {y} = {x -- y} .
eq {x} less y = {x less y} .
eq x less {y} = {x less y} .
eq < {x},y > = {< x,y >} .
eq < x,{y} > = {< x,y >} .
eq s'({x}) = {s'(x)} .
eq s({x}) = {s(x)} .
eq quorem({x},y,c1,c2) = {quorem(x,y,*,*)} .
eq quorem(x,{y},c1,c2) = {quorem(x,y,*,*)} .
eq {{x}} = {x} .
endfm

red quorem(1275000,130,*,*) .
```

Compact Unconditional

```
fmod QUOREM-EUCOND1 is
including NAT .
sort S .
subsort Nat < S .
ops true false * : -> S .
op quorem : S S S -> S .
op <_,_> :  S S  -> S .
op {_} : S -> S .
op _--_ : S S -> S .
op _less_ : S S -> S .
op s' : S -> S .
vars x y q r : S .
eq x less 0 = {false} .
eq 0 less s(x) = {true} .
eq s(x) less s(y) = {x less y} .
vars c1 c2 : S .
eq 0 -- s(y) = {0} .
eq x -- 0 = {x} .
```

```
eq s(x) -- s(y) = {x -- y} .
eq s'(< x,y >) = {< s(x),y >} .
eq quorem(0,s(y),c1) = {< 0,0 >} .
eq quorem(s(x),s(y),*) = quorem(s(x),s(y),{x less y}) .
eq quorem(s(x),s(y),{true}) = {< 0,s(x) >} .
eq quorem(s(x),s(y),{false}) = {s'(quorem(x -- y,s(y),*))} .

eq {x} -- y = {x -- y} .
eq x -- {y} = {x -- y} .
eq {x} less y = {x less y} .
eq x less {y} = {x less y} .
eq < {x},y > = {< x,y >} .
eq < x,{y} > = {< x,y >} .
eq s'({x}) = {s'(x)} .
eq s({x}) = {s(x)} .
eq quorem({x},y,c1) = {quorem(x,y,*)} .
eq quorem(x,{y},c1) = {quorem(x,y,*)} .
eq {{x}} = {x} .
endfm

red quorem(1275000,130,*) .
```

C Elan sources

In the following Elan-sources, $\star$ denotes $\bot$.

Odd/Even

Note: We have used $b(\cdot)$ instead of $\{\cdot\}$ for parsing reasons.

Conditional

```
module oddc
import global bool;
end
sort S;
end
operators global
0       :       S;
s(@)    : (S)   S;
odd(@)  : (S)   bool;
even(@) : (S)   bool;
end

rules for bool
```

```
    x : S;
global
[] odd(0)       => false                        end
[] odd(s(x))    => true         if even(x)      end
[] odd(s(x))    => false        if not even(x)  end
[] even(0)      => true                         end
[] even(s(x))   => true         if odd(x)       end
[] even(s(x))   => false        if not odd(x)   end
end
end
```

Unconditional

```
module oddu
sort S;
end
operators global
0               :               S;
*               :               S;
true            :               S;
false           :               S;
s(@)            : (S)           S;
odd(@,@,@)      : (S S S)       S;
even(@,@,@)     : (S S S)       S;
b(@)            : (S)           S;
end

rules for S
    x,c1,c2 : S;
global
[] odd(0,c1,c2)                 => b(false)                     end
[] odd(s(x),*,c2)               => odd(s(x),b(even(x,*,*)),c2)  end
[] odd(s(x),b(true),c2)         => b(true)                      end
[] odd(s(x),c1,*)               => odd(s(x),c1,b(even(x,*,*)))  end
[] odd(s(x),c1,b(false))        => b(false)                     end
[] even(0,c1,c2)                => b(true)                      end
[] even(s(x),*,c2)              => even(s(x),b(odd(x,*,*)),c2)  end
[] even(s(x),b(true),c2)        => b(true)                      end
[] even(s(x),c1,*)              => even(s(x),c1,b(odd(x,*,*)))  end
[] even(s(x),c1,b(false))       => b(false)                     end
[] b(b(x))                      => b(x)                         end
[] s(b(x))                      => b(s(x))                      end
[] odd(b(x),c1,c2)              => b(odd(x,*,*))                end
[] even(b(x),c1,c2)             => b(even(x,*,*))               end
end
end
```

Compact Unconditional

```
module oddu1
sort S;
end
operators global
0               :       S;
*               :       S;
true            :       S;
false           :       S;
s(@)            : (S)   S;
odd(@,@)        : (S S) S;
even(@,@)       : (S S) S;
b(@)            : (S)   S;
end

rules for S
    x,c1 : S;
global
[] odd(0,c1)            => b(false)                     end
[] odd(s(x),*)          => odd(s(x),b(even(x,*)))       end
[] odd(s(x),b(true))    => b(true)                      end
[] odd(s(x),b(false))   => b(false)                     end
[] even(0,c1)           => b(true)                      end
[] even(s(x),*)         => even(s(x),b(odd(x,*)))       end
[] even(s(x),b(true))   => b(true)                      end
[] even(s(x),b(false))  => b(false)                     end
[] b(b(x))              => b(x)                         end
[] s(b(x))              => b(s(x))                      end
[] odd(b(x),c1)         => b(odd(x,*))                  end
[] even(b(x),c1)        => b(even(x,*))                 end
end
end
```

Quotient/Reminder

We have used $b(\cdot)$ instead of $\{\cdot\}$, $pair(\cdot,\cdot)$ instead of $\langle\cdot,\cdot\rangle$ and $ss(\cdot)$ instead of $s'(\cdot)$ for parsing reasons.

Conditional

```
module quoremc
import global int;
end
sort S;
end
```

```
operators global
quorem(@,@)     : (int int)     S;
pair(@,@)       : (int int)     S;
ss(@)           : (S)           S;
end

rules for S
    x,y : int;
global
 [] ss(pair(x,y))       => pair(x+1,y)                          end
 [] quorem(x,y)         => pair(0,x)            if x<y          end
 [] quorem(x,y)         => ss(quorem(x-y,y))    if not x<y      end
end
end
```

Unconditional

```
module quoremu
import global int bool;
end
sort S;
end
operators global
b(@)            : (bool)        S;
b(@)            : (S)           S;
quorem(@,@,@,@) : (int int S S) S;
pair(@,@)       : (int int)     S;
ss(@)           : (S)           S;
*               :               S;
end

rules for S
    x,y : int;
    c1,c2 : S;
global
 [] ss(pair(x,y))               => b(pair(x+1,y))               end
 [] quorem(x,y,*,c2)            => quorem(x,y,b(x<y),c2)        end
 [] quorem(x,y,b(true),c2)      => b(pair(0,x))                 end
 [] quorem(x,y,c1,*)            => quorem(x,y,c1,b(x<y))        end
 [] quorem(x,y,c1,b(false))     => b(ss(quorem(x-y,y,*,*)))     end
 [] b(b(c1))                    => b(c1)                        end
 [] ss(b(c1))                   => b(ss(c1))                    end
end
end
```

Compact Unconditional

```
module quoremu1
import global int bool;
end
sort S;
end
operators global
b(@)            : (bool)        S;
b(@)            : (S)           S;
quorem(@,@,@)   : (int int S)   S;
pair(@,@)       : (int int)     S;
ss(@)           : (S)           S;
*               :               S;
end

rules for S
    x,y : int;
    c1 : S;
global
 [] ss(pair(x,y))               => b(pair(x+1,y))               end
 [] quorem(x,y,*)               => quorem(x,y,b(x<y))           end
 [] quorem(x,y,b(true))         => b(pair(0,x))                 end
 [] quorem(x,y,b(false))        => b(ss(quorem(x-y,y,*)))       end
 [] b(b(c1))                    => b(c1)                        end
 [] ss(b(c1))                   => b(ss(c1))                    end
end
end
```