Notes on Structure-Preserving Transformations of Conditional Term Rewrite Systems*

Karl Gmeiner[1] and Naoki Nishida[2]

1 **Institute of Computer Science, UAS Technikum Wien**
gmeiner@technikum-wien.at
2 **Graduate School of Information Science, Nagoya University**
nishida@is.nagoya-u.ac.jp

Abstract

Transforming conditional term rewrite systems (CTRSs) into unconditional systems (TRSs) is a common approach to analyze properties of CTRSs via the simpler framework of unconditional rewriting. In the past many different transformations have been introduced for this purpose. One class of transformations, so-called unravelings, have been analyzed extensively in the past.

In this paper we provide an overview on another class of transformations that we call structure-preserving transformations. In these transformations the structure of the conditional rule, in particular their left-hand side is preserved in contrast to unravelings. We provide an overview of transformations of this type and define a new transformation that improves previous approaches.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases conditional term rewriting, unraveling, condition elimination

Digital Object Identifier 10.4230/OASIcs.WPTE.2014.3

Dedicated to the memory of Bernhard Gramlich

1 Introduction

Term rewriting is a widely accepted framework in computer science and has many applications. Conditional rewriting is an intuitive extension of term rewriting that appears naturally in applications like functional programming.

Conditional term rewrite systems (CTRSs) resemble unconditional term rewrite systems (TRSs), yet adding conditions to term rewriting has several drawbacks. From a theoretical point of view, many criteria that hold for unconditional rewriting do not hold for CTRSs, and many properties change their intuitive meaning. From a practical point of view conditional rewriting is complex to implement.

Hence, many transformations have been defined that eliminate the conditions of CTRSs and return unconditional TRSs (e.g. [2, 3, 8]). This way the well-understood framework of unconditional rewriting can be adapted for conditional rewriting, hence giving a better understanding on conditional rewriting and also from a practical point of view allowing us to simulate conditional rewrite sequences.

* The research in this paper is partly supported by the Austrian Science Fund (FWF) international project I963 and the Japan Society for the Promotion of Science (JSPS).

1st International Workshop on Rewriting Techniques for Program Transformations and Evaluation (WPTE'14).
Editors: Manfred Schmidt-Schauß, Masahiko Sakai, David Sabel, and Yuki Chiba; pp. 3–14

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We here provide an overview on a class of transformations that we refer to as structure-preserving and explain similarities and differences to a well-analyzed class of transformations, so-called unravelings. Definitions of structure-preserving derivations are usually complex compared to the ones of unravelings and they are usually only defined for CTRSs without extra variables. Therefore, we here also provide a definition of the transformation of [1] for CTRSs with (deterministic) extra variables. Since the transformation of [1] returns good results only for constructor CTRSs we also formally define a transformation of non-constructor CTRSs into constructor CTRSs. We will show that the combination of both transformation has better properties than other structure-preserving transformations. Proving further properties of this transformation will be part of our future work.

2 Preliminaries, Notions and Notations

We assume basic knowledge of conditional term rewriting and follow the basic notions and notations as they are defined in [13].

A conditional term rewrite system (CTRS) is a rewrite system that consists of conditional rules $l \to r \Leftarrow c$. The condition c is usually a conjunction of equations $s_1 = t_1, \ldots, s_k = t_k$.

There are different possible interpretations of equality in the conditions. CTRSs in which equality is interpreted as joinability $\downarrow$ are join CTRSs. Here we mainly consider oriented CTRSs, in which the conditions are interpreted as reducibility $\to^*$.

In contrast to unconditional TRSs, CTRSs may contain extra variables. We will consider CTRSs with extra variables that can be determined by rewrite steps (deterministic extra variables). CTRSs with only deterministic extra variables are called deterministic CTRSs (DCTRSs). Deterministic conditional rewrite rules $l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_k \to^* t_k$ satisfy the condition $\mathcal{V}ar(s_i) \subseteq \mathcal{V}ar(l, t_1, t_{i-1})$ and $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l, t_1, \ldots, t_k)$.

A symbol $f \in \mathcal{F}$ in the signature of a CTRS $(R, \mathcal{F})$ is a defined symbol ($f \in \mathcal{D}$) if it is the root symbol of the left-hand side of a rule in R. All non-defined symbols are constructor symbols $\mathcal{C}$. A term is a constructor term if it only contains function symbols of $\mathcal{C}$ and variables. A CTRS is a constructor CTRS if the left-hand sides of all rules are of the shape $f(u_1, \ldots, u_n)$ where the arguments $u_1, \ldots, u_n$ are constructor terms.

There are several classes of CTRSs depending on the distribution of extra variables. A CTRS without extra variables is a 1-CTRS. If additionally the right-hand sides of the conditions are irreducible ground terms it is a normal 1-CTRS.

In some cases we will use the notation $\overrightarrow{X}$ where X is a set of terms. $\overrightarrow{X}$ represents the stream of variables in X in an unspecified but fixed order. Furthermore we will refer to rules with f as the root symbol on the left-hand side as f-rules.

3 Transformations of CTRSs

3.1 Overview

In [9] a class of transformations is introduced, so-called *unravelings*, and several properties are proved or disproved. There are some unravelings defined for some CTRSs (so-called normal 1-CTRSs and join 1-CTRSs). In [10] and [13] an unraveling is presented for deterministic CTRSs, a class of CTRSs that allows extra variables to a certain extend. This and similar unravelings have been analyzed extensively in the past (e.g. [11, 5, 12, 6]).

In [16] another transformation is presented that does not match the class of unravelings. This type of transformation is extended in [1, 14, 4].

We refer to these transformations as structure-preserving transformations because in contrast to unravelings these transformations do not encode the conditions in new function symbols but instead they encode them in the left-hand side of conditional rules. Hence, the original structure of terms is much better preserved. Such structure-preserving transformations have not been analyzed as much or consistently as unravelings.

In [15] a structure-preserving transformation is introduced for which "computational equivalence" is proven. In [4] a framework is introduced that allows the description of properties of unravelings and structure-preserving derivations consistently. Furthermore, some theoretical and practical drawbacks of the transformation of [15] are pointed out and another transformation is introduced that does not show these drawbacks, yet it is only applicable for a smaller class of CTRSs.

3.2 Transformations for DCTRSs

In transformations for DCTRSs conditions are eliminated by splitting a conditional rule into multiple unconditional rewrite rules in which the conditions are wrapped. The rule that introduces the first conditional argument is the *introduction rule*. After a condition has successfully been evaluated we switch to the next condition using a switch rule, or we eliminate the conditional argument using an elimination rule.

$$\mathbb{T}(l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_k \to^* t_k) = \left\{ \begin{array}{rl} l \to C_1[s_1] & \text{introduction rule} \\ C_1[t_1] \to C_2[s_2] & \text{switch rules} \\ \vdots \quad \vdots & \text{switch rules} \\ C_k[t_k] \to r & \text{elimination rule} \end{array} \right\}$$

3.3 Unravelings

The class of unravelings that was introduced in [9] contains transformations that keep the original signature of the transformed system but add some new function symbols in which the conditions are wrapped while being evaluated. For every conditional rule a new function symbol is used.

▶ **Example 1.** Consider the following simple CTRS

$$\mathcal{R} = \left\{ \alpha : or(x,y) \to true \Leftarrow x \to^* true \qquad \beta : or(x,y) \to true \Leftarrow y \to^* true \right\}$$

Using the unraveling of [13] we obtain the following TRS

$$\mathbb{U}(\mathcal{R}) = \left\{ \begin{array}{ll} or(x,y) \to U_1^\alpha(x,x,y) & or(x,y) \to U_1^\beta(y,x,y) \\ U_1^\alpha(true,x,y) \to true & U_1^\beta(true,x,y) \to true \end{array} \right\}$$

In order to simulate the conditional rewrite sequence $or(true, false) \to_{\mathcal{R}}^* true$ we first apply the introduction rule of α and then the elimination rule:

$$or(true, false) \to_{\mathbb{U}(\mathcal{R})} U_1^\alpha(true, true, false) \to_{\mathbb{U}(\mathcal{R})} true$$

If we apply the introduction rule of β we obtain $or(true, false) \to_{\mathbb{U}(\mathcal{R})} U_1^\beta(false, true, false)$ where the latter term cannot be reduced any further.

Observe that in the previous example the unraveled TRS is not confluent although the original CTRS is confluent. If multiple conditional rules are applicable we must choose one introduction rule that should be applied. The other conditional rule cannot be applied then anymore. In [7] we introduced an unraveling that preserves confluence for some confluent CTRSs (including the CTRS of the previous example), yet for overlapping CTRSs, unravelings usually do not preserve confluence.

3.4 Structure-Preserving Transformations

Structure-preserving transformations stem from [16]. Further transformations of this class were introduced in [1], [15] and [4].

In structure-preserving transformations no new defined symbols are added by the transformation but instead additional conditional arguments are added to defined symbols but increases their arity in order to wrap the conditions. Hence, we need to replace function symbols in terms by the new function symbol. In order to translate terms from the original CTRS into the transformed TRS we will use an initialization mapping ϕ. Such a mapping is not needed in unravelings.

The additional argument in the defined function symbols contains the conditional argument. If the left-hand sides of multiple conditional rules are rooted by the same function symbol, the root symbol contains one conditional argument for each condition. An uninitialized conditional argument is marked by the constant $\bot$.

▶ **Example 2.** Consider the CTRS of Example 1:

$$\mathcal{R} = \left\{ \alpha : or(x,y) \to true \Leftarrow x \to^* true \qquad \beta : or(x,y) \to true \Leftarrow y \to^* true \right\}$$

The transformation of [1] ($\mathbb{S}$ in the following) increases the arity of *or* by two because we need one conditional argument for each conditional rule. If a term matches the left-hand side of a rule and the conditional argument is not initialized we can introduce the conditional argument. The other conditional argument is preserved so that both conditions can be evaluated in parallel.

$$\mathbb{S}(\mathcal{R}) = \left\{ \begin{array}{ll} or'(x,y,\bot,z) \to or'(x,y,x,z) & or'(x,y,z,\bot) \to or'(x,y,z,y) \\ or'(x,y,true,z) \to true & or'(x,y,z,true) \to true \end{array} \right\}$$

In order to simulate the conditional rewrite sequence $or(true, false) \to^*_{\mathcal{R}} true$ we obtain the correct normalform even if we apply the introduction rule of β first:

$$\begin{aligned} or'(true, false, \bot, \bot) &\to_{\mathbb{T}(\mathcal{R})} or'(true, false, \bot, false) \\ &\to_{\mathbb{T}(\mathcal{R})} or'(true, false, true, false) \to_{\mathbb{T}(\mathcal{R})} true \end{aligned}$$

The advantage of structure-preserving transformations compared to unravelings is that the conditions are encoded as decorators of the original terms. Hence, other rules remain applicable even if we evaluate conditions.

Another benefit of this approach is that we can exploit parallelism in reductions. Failed conditions do not block other derivations, in particular we can introduce other conditional rules. While in the unravelings of [9] and [13] we need to make an assumption which condition is satisfied already in the introduction step, we can postpone this decision in structure-preserving transformations until we evaluated all possible conditions.

One drawback of this type of derivations is their more complex definition. While unravelings have been defined for deterministic CTRSs, such a definition is only hinted in [15] and [4] for structure-preserving transformations.

4 Properties of Transformations

4.1 Soundness and Completeness

In order to prove properties of CTRSs by transforming them into unconditional TRSs we need to show that the rewrite relation of the original CTRS is properly approximated by the transformed TRS.

There are two main properties that we are interested in. *Completeness* means that a rewrite sequence in the original CTRS corresponds to a rewrite sequence in the transformed TRS. This property is usually satisfied and easy to prove.

The other direction, soundness, means that a rewrite sequence in the transformed system corresponds to a rewrite sequence in the original system. This property is more difficult to prove and usually not satisfied which has first been shown in [9]. In the past, unravelings have been proven to be sound for many syntactic properties and strategies ([5][12][6]).

4.2 Unsoundness of Structure-Preserving Transformations

In order to preserve confluence in transformations structure-preserving transformations encode conditions in parallel if multiple rules share the same root symbol on their left-hand side. Parallel evaluations of conditions of different rules allow us to postpone the decision which conditional rule will ultimately be applied in the transformed TRS. Yet, this also allows us to interleave multiple conditional rules by applying non-linear rules before applying an elimination step. This in fact causes unsoundness, even for constructor normal 1-CTRSs for which unravelings are known to be sound.

▶ **Example 3.** Consider the following overlay normal 1-CTRS

$$\mathcal{R} = \left\{ \begin{array}{cc} a \to c & f(x) \to C \Leftarrow x \to^* c \\ a \to d & f(x) \to D \Leftarrow x \to^* d \\ g(x,x) \to h(x,x) & \end{array} \right\}$$

f is the root symbol of the left-hand side of two conditional rules, hence we append to conditional arguments to f-terms in the rewrite system and insert the conditional arguments:

$$\mathbb{S}(\mathcal{R}) = \left\{ \begin{array}{cc} a \to c & f'(x,\bot,z) \to f'(x,x,z) \\ a \to d & f'(x,c,z) \to C \\ g(x,x) \to h(x,x) & f'(x,z,\bot) \to f'(x,z,x) \\ & f'(x,z,d) \to D \end{array} \right\}$$

In the original CTRS the term $g(f(a), f(b))$ rewrites to $h(C,C)$ and $h(D,D)$ but not to $h(C,D)$ because $f(a)$ and $f(b)$ do not have a common reduct that rewrites to both $f(a)$ and $f(b)$.

The term $g(f(a), f(b))$ corresponds to the term $g(f'(a,\bot,\bot), f'(b,\bot,\bot))$ in $\mathbb{T}(\mathcal{R})$. Observe the following derivation in the transformed TRS:

$$\begin{aligned} g(f'(a,\bot,\bot), f'(b,\bot,\bot)) &\to^* g(f'(a,a,a), f'(b,b,b)) \to^* g(f'(c,c,d), f'(c,c,d)) \\ &\to^* h(f'(c,c,d), f'(c,c,d)) \to h(C, f'(c,c,d)) \to h(C,D) \end{aligned}$$

Since this derivation is not possible in the original CTRS the transformation is unsound.

In the previous example the term $f'(c, c, d)$ contains two conditional arguments and both of them are satisfied. Therefore we can apply two elimination rules to this term. If we only encoded one conditional argument this would not be possible. In fact, the unravelings of [13] and also [11] are sound for this concrete example.

Therefore, soundness of unravelings do not imply soundness for structure-preserving derivations.

5 Structure-Preserving Transformations for Non-Constructor CTRSs

The structure-preserving transformation of [1] $\mathbb{S}$ is unsound for many non-constructor CTRS:

▶ **Example 4.** Consider the following CTRS from [1]:

$$\mathcal{R} = \{\ f(g(x)) \to x \Leftarrow x \to^* s(0) \qquad g(s(x)) \to g(x)\ \}$$

The CTRS is transformed into the following unconditional TRS:

$$\mathbb{S}(\mathcal{R}) = \left\{ \begin{array}{ll} f'(g(x), \bot) \to f'(g(x), x) & g(s(x)) \to g(x) \\ f'(g(x), s(0)) \to x & \end{array} \right\}$$

In $\mathcal{R}$, $f(g(s(0)))$ rewrites to $s(0)$ because the condition is satisfied. It also rewrites to $f(g(0))$ using the g-rule. The latter term is in normalform because the condition $0 \to^* s(0)$ is not satisfied.

In $\mathbb{S}(\mathcal{R})$, $f(g(s(0)))$ corresponds to the term $f'(g(s(0)), \bot)$. We obtain the following unsound derivation:

$$f'(g(s(0)), \bot) \to f'(g(s(0)), s(0)) \to f'(g(0), s(0)) \to 0$$

In the previous example we obtain unsoundness because both the redex and the reduct of the rewrite step $f(g(s(0))) \to f(g(0))$ match the left-hand side of the conditional rule, yet the variable bindings cannot be reduced to each other. In unravelings this does not cause soundness because the introduction step destroys the structure of the left-hand side of the conditional rule and only keeps the variable bindings. In structure-preserving transformations the structure is preserved and hence it can be modified.

5.1 Transformation $\mathbb{S}_{sr}$

In [15] a transformation is presented that extends the transformation $\mathbb{S}$ so that also overlapping CTRSs can be transformed appropriately. The transformation adds a complex unary operator that is propagated to outer positions and resets conditional arguments.

▶ **Example 5** (Transformation of [15])**.** The transformation of [15] extends the transformation of [1] by a unary function symbol $\{\ .\ \}$ that creates a layer around contracted redexes. The transformed TRS of the CTRS of Example 4 therefore contains the following rules:

$$\mathcal{R}'_1 = \left\{ \begin{array}{ll} f'(g(x), \bot) \to f'(g(x), \{x\}) & g(s(x)) \to \{g(x)\} \\ f'(g(x), \{s(0)\}) \to \{x\} & \end{array} \right\}$$

Now overlapping rewrite steps are blocked because of the new function symbol:

$$f'(g(s(0)), \bot) \to f'(g(s(0)), \{s(0)\}) \to f'(\{g(0)\}, \{s(0)\})$$

The new unary symbol is propagated to outer positions by adding one new rule for each argument of each function symbol. Such propagation steps reset conditional arguments and thereby avoid that outdated conditional arguments are used in elimination steps. Furthermore the new function symbol must be idempotent:

$$\mathcal{R}_2' = \left\{ \begin{array}{cc} f'(\{x\},z) \to \{f'(x,\bot)\} & g(\{x\}) \to \{g(x)\} \\ s(\{x\}) \to \{s(x)\} & \{\{x\}\} \to \{x\} \end{array} \right\}$$

The transformed TRS then is $\mathbb{S}_{sr}(\mathcal{R}) = \mathcal{R}_1 \cup \mathcal{R}_2$.

The term $f'(\{g(0)\},\{s(0)\})$ now can only be reduced by propagating the unary function symbol to the root position which resets the conditional argument:

$$f'(\{g(0)\},\{s(0)\}) \to \{f'(g(0),\bot)\} \to \{f'(g(0),\{0\})\}$$

The last term is irreducible.

5.2 Transformation $\mathbb{S}_{gg}$

In [4] it is pointed out that the transformation of [15] has some disadvantages. Apart from the complex definition of the new function symbol $\{\ .\ \}$ it is also non-preserving for many important syntactic properties like being non-overlapping, being a constructor system or being an overlay system.

From a practical point of view the transformation resets conditional arguments too often. The transformation of [4] tries to resolve these problems. Since the transformation of [4] is very complex in its definition we here provide a simpler refinement.

The main idea of the transformation of [4] is to add information to subterms of redexes to see whether an overlapping rewrite step was applied and the conditional argument should be reset. For this purpose we increase the arity of all defined function symbols (instead of just the root symbol) on the left-hand side of a conditional rule. While the root symbol encodes the condition, defined symbols strictly below the root contain a check argument. If they are uninitialized they contain $\bot$. After the introduction step these additional check arguments are marked with $\top$ to indicate that they were used in a conditional argument. In a rewrite step, all these check arguments are reset to $\bot$ to indicate that a conditional argument might be outdated. An elimination step is only allowed if all check arguments contain $\top$. For the introduction step it is sufficient if the conditional argument or one check argument is uninitialized. Therefore, one conditional rule might give rise to multiple introduction rules.

▶ **Example 6.** The left-hand side of the conditional rule of Example 4 contains the defined symbol g that therefore is replaces by a new binary symbol g' where the second argument is a check argument. If the conditional argument or the check argument is uninitialized we introduce the conditional argument.

$$\mathbb{S}_{\mathrm{gg}}(\mathcal{R}) = \left\{ \begin{array}{cc} f'(g'(x,z),\bot) \to f'(g'((x,\top)),\langle x\rangle) & f'(g'(x,\top),\langle s(0)\rangle) \to x \\ f'(g'(x,\bot),z) \to f'(g(\mathrm{'}(x,\top)),\langle x\rangle) & g'(s(x),z) \to g(x,\bot) \end{array} \right\}$$

Now, $f(g(s(0)))$ gives rise to the following derivation:

$$f'(g'(s(0),\bot),\bot) \to f'(g'(s(0),\top),\langle s(0)\rangle) \to f'(g'(0,\bot),\langle s(0)\rangle) \to f'(g'(0,\top),\langle 0\rangle)$$

It is not possible to reproduce the unsound derivation of Example 4.

In contrast to the transformation of [15] the transformation of [4] and also the refinement that is sketched here preserves many properties like being a constructor system (for normal 1-CTRSs), yet does not return satisfying results in all cases. For non-left-linear confluent CTRSs we might obtain non-confluence even if [15] returns a confluent CTRS. Furthermore, in collapsing CTRSs we still might obtain unsoundness (see [4, Example 8]) even though other transformations are sound.

6 New Transformation

The transformation of [1] is only applicable for constructor normal 1-CTRSs. The transformation of [15] allows the transformation also of non-constructor normal 1-CTRSs, yet it is syntactically complex. The transformation of [4] conservatively extends the transformation of [1], but its definition is complex and furthermore it is less powerful than the transformation of [15].

In our new approach we therefore modularize the transformational approach and use a transformation from non-constructor CTRSs into constructor CTRS before eliminating the conditions. This way we only need to consider constructor CTRSs in our new transformation.

First we define the transformation from non-constructor CTRSs into constructor CTRSs.

▶ **Definition 7** (transformation for non-constructor CTRSs)**.** Let $\alpha : l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_k \to^* t_k$ be a conditional rule, then $cons$ is defined recursively as follows:

$$cons(\alpha) = \begin{cases} cons(\alpha') & \text{where } \alpha' = l[z]_p \to r \Leftarrow z \to^* l|_p, s_1 \to^* t_1, \ldots, s_k \to^* t_k, \\ & z \text{ is a fresh new variable } (z \notin \mathcal{V}ar(\alpha)) \text{ and} \\ & l|_p \ (p \in \mathcal{P}os(l) \setminus \{\epsilon\}) \text{ is not a constructor term} \\ \alpha & \text{if } l|_p \text{ is a constructor terms for all } p \in \mathcal{P}os(l) \setminus \{\epsilon\} \end{cases}$$

The mapping $cons$ is extended to CTRSs as follows: $cons(R) = \bigcup_{\alpha \in R} cons(\alpha)$.

▶ **Example 8.** Consider the CTRS of Example 4. The conditional rule $\alpha : f(g(x)) \to x \Leftarrow x \to^* s(0)$ is not a constructor rule because g is a defined symbol.

Using $cons$, the g-subterm is replaced by the fresh new variable z and a condition $z \to^* g(x)$ is added:

$$cons(\alpha) \quad = \quad f(z) \to x \Leftarrow z \to^* g(x), x \to^* s(0)$$

Observe that the rule α does not contain extra variables while $cons(\alpha)$ contains the deterministic extra variable z.

If the left-hand side of a conditional rule contains multiple non-constructor terms as subterms $cons$ does not imply any order of subterms. Our theoretical results do not depend on a specific order of positions, yet from a practical point of view choosing outer positions over inner positions first leads to less conditions.

▶ **Lemma 9.** *Let $\mathcal{R}$ be a deterministic CTRS. Then $cons(\mathcal{R})$ is a constructor DCTRS.*

Proof. Straightforward from the definition. ◀

▶ **Lemma 10** (Completeness)**.** *Let $\mathcal{R}$ be a DCTRS and $s, t \in \mathcal{T}$ be two terms. Then $s \to_{\mathcal{R}} t$ implies $s \to_{cons(\mathcal{R})} t$.*

Proof. In this case, the variable binding of the left-hand sides of the new conditions immediately matches the right-hand sides. ◀

▶ **Lemma 11** (Soundness). *Let $\mathcal{R}$ be a DCTRS and $s, t \in \mathcal{T}$ be two terms. Then if $s \to_{cons(\mathcal{R})} t$ then also $s \to^+_{\mathcal{R}} t$.*

Proof. We can extract the rewrite sequences in the new conditions and insert them in the replaced subterms. This is possible because all new variables are only used once. ◀

Next, we define the transformation for DCTRSs that conservatively extends the transformation of [1].

In order to transform a CTRS we group conditional rules by the root symbol of their left-hand side. We then transform these groups. In order to encode CTRSs with extra variables we sequentially encode all conditions. If a conditional argument matches the right-hand side of a condition, then we can apply a switch rule to evaluate the next condition. In this switch rule we do not keep the evaluated conditional argument to avoid derivations similar to the unsound derivation in Example 4 but instead encode variables that are needed on the right-hand side of the conditional rule or in one of the following conditions. This set of variables resembles the variables that are encoded in the optimized unraveling of [11], but in our case we only need to encode extra variables because we preserve the left-hand side of the conditional rule.

In order to further distinguish which condition is currently evaluated we also label the tuples that contain the conditional argument and the bindings of extra variables.

▶ **Definition 12** (structure-preserving transformation for sets of conditional rules). Let R_f be a set of conditional rules such that the left-hand sides of all rules are rooted by the same function symbol f with arity n.

Then, the mappings $\phi^{R_f}_X : \mathcal{T} \mapsto \mathcal{T}'$ and $\phi^{R_f}_\bot : \mathcal{T} \mapsto \mathcal{T}'$ are defined as follows:

$$\phi^{R_f}_\bot(u) = \begin{cases} f'(\phi^{R_f}_\bot(u_1), \ldots, \phi^{R_f}_\bot(u_n), \overbrace{\bot, \ldots, \bot}^{|R_f| \text{ times}}) & \text{if } u = f(u_1, \ldots, u_n) \\ g(\phi^{R_f}_\bot(u_1), \ldots, \phi^{R_f}_\bot(u_m)) & \text{if } u = g(u_1, \ldots, u_m) \\ u & \text{if } u \text{ is a variable} \end{cases}$$

$$\phi^{R_f}_X(u) = \begin{cases} f'(\phi^{R_f}_{X_1}(u_1), \ldots, \phi^{R_f}_{X_n}(u_n), z_1, \ldots, z_{|R_f|}) & \text{if } u = f(u_1, \ldots, u_n) \\ g(\phi^{R_f}_{Y_1}(u_1), \ldots, \phi^{R_f}_{Y_m}(u_m)) & \text{if } u = g(u_1, \ldots, u_m) \\ u & \text{if } u \text{ is a variable} \end{cases}$$

where $\{z_1, \ldots, z_{|R_f|}\} \subset X$, $X_1, \ldots, X_n$ are pairwise distinct subsets of $X \setminus \{z_1, \ldots, z_{|R_f|}\}$ and $Y_1, \ldots, Y_m$ are pairwise distinct subsets of X.

Let $i_\alpha \in \{1, \ldots, |R_f|\}$ be a unique index of the rule α in R_f. Then the transformed rules $\mathbb{S}_{\text{new}}(\alpha)$ of the rule $\alpha \in R_f$ are defined as follows:

$$\mathbb{S}_{\text{new}}(\alpha) = \left\{ \begin{array}{r} l'[\bot]_{n+i_\alpha} \to l'[\left\langle \phi^{R_f}_\bot(s_1), \overrightarrow{\mathcal{Z}_1} \right\rangle_1]_{n+i_\alpha} \\ l'[\left\langle \phi^{R_f}_{X_1}(t_1), \overrightarrow{\mathcal{Z}_1} \right\rangle_1]_{n+i_\alpha} \to l'[\left\langle \phi^{R_f}_\bot(s_2), \overrightarrow{\mathcal{Z}_2} \right\rangle_2]_{n+i_\alpha} \\ \vdots \quad \vdots \\ l'[\left\langle \phi^{R_f}_{X_k}(t_k), \overrightarrow{\mathcal{Z}_k} \right\rangle_k]_{n+i_\alpha} \to \phi^{R_f}_\bot(r) \end{array} \right\}$$

where $l' = \phi_X(l)$, $\mathcal{Z}_i = \bigcup \mathcal{V}ar(t_1, \ldots, t_{i-1}) \cap \mathcal{V}ar(t_i, s_{i+1}, \ldots, s_k, t_k, r) \setminus \mathcal{V}ar(l)$ is the set of extra variables that are still required for the rule application, and $X \cap \mathcal{V}ar(\alpha) = \emptyset$ is an infinite set of fresh variables.

The previous definition shows how to transform conditional rules and how to obtain mappings to map the signature of terms to the transformed system. Since we will obtain groups of unconditional rules with a different signature this way we need to provide a mapping to adjust the signature of other rules.

▶ **Definition 13** (adjusting signature)**.** Let R_f be a set of f-rooted conditional rules. Let $\beta : l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_k \to^* t_k$ be a rule that is not f-rooted such that all subterms of l are constructor terms.

Then, $\phi_{R_f}(\beta)$ is defined as follows

$$\phi_{R_f}(\beta) = \phi^{R_f}_{X_0}(l) \to \phi^{R_f}_{\perp}(r) \Leftarrow \phi^{R_f}_{\perp}(s_1) \to^* \phi^{R_f}_{X_1}(t_1), \ldots \phi^{R_f}_{\perp}(s_1) \to^* \phi^{R_f}_{X_k}(t_k)$$

where $X_0, \ldots, X_k$ are infinite pairwise distinct sets of new variables ($\mathcal{V}ar(\beta) \cap \bigcup_{i=0}^{k} X_i = \emptyset$).

The final transformation itself groups rules by their root symbols and applies the transformation of Definition 12 to them. Then, the signature is adjusted for all rules according to the mappings ϕ_{R_f}.

▶ **Definition 14** (structure-preserving transformation for DCTRSs)**.** Let $\mathcal{R} = (R, \mathcal{F})$ be a constructor DCTRS such that $f_1, \ldots, f_n$ are all defined symbols and R_{f_i} contains all conditional f_i-rooted rules ($i \in \{1, \ldots, n\}$). Let furthermore R_{uc} be all unconditional rules in $\mathcal{R}$. Then the transformation $\mathbb{S}_{\text{new}}$ is defined as follows:

$$\mathbb{S}_{\text{new}}(\mathcal{R}) = \phi_{R_{f_1}}(\cdots \phi_{R_{f_n}}(R_{uc}) \cdots) \cup \bigcup_{i=1}^{n} \phi_{R_{f_1}}(\cdots \phi_{R_{f_{i-1}}}(\phi_{R_{f_{i+1}}}(\cdots \phi_{R_{f_n}}(\mathbb{S}_{\text{new}}(R_{f_i}) \cdots)$$

▶ **Example 15.** Consider the following CTRS of Example 4:

$$\mathcal{R} = \{\ f(g(x)) \to x \Leftarrow x \to^* 0 \qquad g(s(x)) \to g(x) \quad \}$$

In order to apply the transformation $\mathbb{T}_{\text{new}}$ we first must apply *cons* to transform $\mathcal{R}$ into a constructor CTRS.

$$\mathit{cons}(\mathcal{R}) = \{\ f(z) \to x \Leftarrow z \to^* g(x), x \to^* s(0) \qquad g(s(x)) \to g(x)\}$$

Next, we transform the conditional f-rule and the unconditional g-rule:

$$\mathbb{S}_{\text{new}}(R_f) = \{\ f'(z, \perp) \to f'(z, \langle z \rangle_1) \quad f'(z, \langle g(x) \rangle_1) \to f'(z, \langle x, x \rangle_2) \quad f'(z, \langle 0, x \rangle_2) \to x \quad \}$$

$$\phi_{R_f}(R_{uc}) = \{\ g(s(x)) \to g(x) \quad \}$$

Finally, we obtain $\mathbb{S}_{\text{new}}(\mathcal{R})$ by adjusting the signature in the transformed rules. Since the symbol g is preserved this is equivalent to the union of both subsystems.

$$\mathbb{S}_{\text{new}}(\mathcal{R}) = \left\{ \begin{array}{l} f'(z, \perp) \to f'(z, \langle z \rangle_1) \quad f'(z, \langle g(x) \rangle_1) \to f'(z, \langle x \rangle_2) \quad f'(z, \langle 0 \rangle_2) \to x \\ g(s(x)) \to g(x) \end{array} \right\}$$

This transformation does not require a resetting-mechanism like the transformation of [15]. Furthermore it preserves the property of being a constructor system if the rhs's of the conditions are constructor terms, and being non-overlapping if the rhs's of the conditions are non-overlapping with the lhs's of the rules. Compared to the transformation of [4] $\mathbb{S}_{\text{new}}$ does not cause non-confluence in connection with non-left-linear rules or unsoundness in connection with collapsing rules. Finally, our definition of $\mathbb{S}_{\text{new}}$ is the only formal definition known to us of a structure-preserving transformation for deterministic CTRSs.

Our next goals will be to prove soundness properties, in particular to compare our novel approach with recent properties of unravelings. Although it is known that structure-preserving transformations are unsound for certain non-erasing CTRSs while some unravelings are sound we hope to present some results in the near future that show a connection in soundness results.

7 Conclusion

We have presented an overview of transformations that preserve the term structure of left-hand sides of conditional rule. This class of transformations stems from [16]. We refer to these transformations as structure preserving transformations.

The transformation of [1] ($\mathbb{S}$) has nice properties for constructor normal 1-CTRSs. Yet for non-constructor CTRSs we obtain undesirable properties. Therefore, some other transformations were defined in the past that are based on this transformation but also return appropriate transformed TRSs for non-constructor CTRSs.

The extension of [15] ($\mathbb{S}_{\text{sr}}$) adds a complex resetting mechanism to $\mathbb{S}$ that has complex syntactic properties. The drawbacks of this transformation are discussed in [4] where also another transformation based on [1] is defined ($\mathbb{S}_{\text{gg}}$) that has better syntactic properties than $\mathbb{S}_{\text{sr}}$. Since the original definition is very complex we here sketched a simpler refined version.

The transformation $\mathbb{S}_{\text{gg}}$ also has undesirable properties for some collapsing CTRSs. Furthermore none of these transformations is formally defined for deterministic CTRSs. Therefore we here define a transformation for constructor DCTRSs that is similar to the one of [1] but it uses a sequential encoding of conditions similar to unravelings for DCTRSs.

In order to also transform non-constructor CTRSs using this CTRS we also define a transformation of non-constructor CTRSs into constructor DCTRSs. Combining these two transformations we obtain a new approach that shows promising first results compared to other structure preserving derivations.

In our future work we hope to provide more formal results and prove the usefulness of our approach in automated confluence tests of CTRSs.

Acknowledgements. We are deeply grateful to the anonymous referees for their useful comments.

References

1 Sergio Antoy, Bernd Brassel, and Michael Hanus. Conditional narrowing without conditions. In *Proc. 5th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, 27–29 August 2003, Uppsala, Sweden*, pages 20–31. ACM Press, 2003.

2 Jan A. Bergstra and Jan Willem Klop. Conditional rewrite rules: Confluence and termination. *Journal of Computer and System Sciences*, 32(3):323–362, 1986.

3 Elio Giovanetti and Corrado Moiso. Notes on the elimination of conditions. In Stéphane Kaplan and Jean-Pierre Jouannaud, editors, *Proc. 1st Int. Workshop on Conditional Rewriting Systems (CTRS'87), Orsay, France, 1987*, volume 308 of *Lecture Notes in Computer Science*, pages 91–97, Orsay, France, 1988. Springer. ISBN 3-540-19242-5.

4 Karl Gmeiner and Bernhard Gramlich. Transformations of conditional rewrite systems revisited. In Andrea Corradini and Ugo Montanari, editors, *Recent Trends in Algebraic Development Techniques (WADT 2008) – Selected Papers*, volume 5486 of *Lecture Notes in Computer Science*, pages 166–186. Springer, 2009.

5 Karl Gmeiner, Bernhard Gramlich, and Felix Schernhammer. On (un)soundness of unravelings. In Christopher Lynch, editor, *Proc. 21st International Conference on Rewriting Techniques and Applications (RTA 2010), July 11–13, 2010, Edinburgh, Scotland, UK*, LIPIcs (Leibniz International Proceedings in Informatics), July 2010.

6 Karl Gmeiner, Bernhard Gramlich, and Felix Schernhammer. On soundness conditions for unraveling deterministic conditional rewrite systems. In Ashish Tiwari, editor, *Proc. 23rd International Conference on Rewriting Techniques and Applications (RTA 2012), May 30 to June 2, 2012, Nagoya, Japan*, LIPIcs (Leibniz International Proceedings in Informatics), May/June 2012.

7 Karl Gmeiner, Naoki Nishida, and Bernhard Gramlich. Proving confluence of conditional term rewriting systems via unravelings. In Nao Hirokawa and Vincent van Oostrom, editors, *Proceedings of the 2nd International Workshop on Confluence*, pages 35–39, 2013.

8 Claus Hintermeier. How to transform canonical decreasing ctrss into equivalent canonical trss. In *Conditional and Typed Rewriting Systems, 4th International Workshop, CTRS-94, Jerusalem, Israel, July 13–15, 1994, Proceedings*, volume 968 of *Lecture Notes in Computer Science*, pages 186–205, 1995.

9 Massimo Marchiori. Unravelings and ultra-properties. In Michael Hanus and Mario Mario Rodríguez-Artalejo, editors, *Proc. 5th Int. Conf. on Algebraic and Logic Programming, Aachen*, volume 1139 of *Lecture Notes in Computer Science*, pages 107–121. Springer, September 1996.

10 Massimo Marchiori. On deterministic conditional rewriting. Technical Report MIT LCS CSG Memo n.405, MIT, Cambridge, MA, USA, October 1997.

11 Naoki Nishida, Masahiko Sakai, and Toshiki Sakabe. Partial inversion of constructor term rewriting systems. In Jürgen Giesl, editor, *Proc. 16th International Conference on Rewriting Techniques and Applications (RTA'05), Nara, Japan, April 19–21, 2005*, volume 3467 of *Lecture Notes in Computer Science*, pages 264–278. Springer, April 2005.

12 Naoki Nishida, Masahiko Sakai, and Toshiki Sakabe. Soundness of unravelings for deterministic conditional term rewriting systems via ultra-properties related to linearity. In Manfred Schmidt-Schauss, editor, *Proc. 22nd International Conference on Rewriting Techniques and Applications (RTA 2011), May 30 to June 1, 2011, Novi Sad, Serbia*, LIPIcs (Leibniz International Proceedings in Informatics), 2011. pages 267–282.

13 Enno Ohlebusch. *Advanced Topics in Term Rewriting*. Springer, 2002.

14 Grigore Rosu. From conditional to unconditional rewriting. In José Luiz Fiadeiro, Peter D. Mosses, and Fernando Orejas, editors, *WADT*, volume 3423 of *Lecture Notes in Computer Science*, pages 218–233. Springer, 2004.

15 Traian-Florin Şerbănuţă and Grigore Roşu. Computationally equivalent elimination of conditions. In Frank Pfenning, editor, *Proc. 17th International Conference on Rewriting Techniques and Applications, Seattle, WA, USA, August 12–14, 2006*, volume 4098 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2006.

16 Patrick Viry. Elimination of conditions. *J. Symb. Comput.*, 28(3):381–401, 1999.