





Composing Proof Terms

Christina Kohl^(✉)  and Aart Middeldorp 

Department of Computer Science, University of Innsbruck, Innsbruck, Austria
{christina.kohl,aart.middeldorp}@uibk.ac.at

Abstract. Proof terms are a useful concept for comparing computations in term rewriting. We analyze proof terms with composition, with an eye towards automation. We revisit permutation equivalence and projection equivalence, two key notions presented in the literature. We report on the integration of proof terms with composition into ProTeM, a tool for manipulating proof terms.

Keywords: Proof terms · Term rewriting · Automation

1 Introduction

Proof terms represent proofs in rewriting logic [4, 5]. Because proof terms are terms, they are subject to techniques common in automated reasoning, like termination orders and critical pair analysis. In term rewriting proof terms are used to study equivalence of reductions [6, 7] and for confluence analysis [2]. In [7, Chapter 8] ([6] is a condensed version) van Oostrom and de Vrijer present a thorough study of five different notions of equivalence and argue that these are equivalent. Proof terms play a key role in three of these notions: *permutation equivalence*, *parallel standardization equivalence* and *projection equivalence*. In this paper we take a fresh look at permutation equivalence and projection equivalence, from the viewpoint of automation. This leads to a new understanding of the rewrite properties of the important residual operation. In particular, we show the analysis in [6, 7] of the residual operation to be incorrect.

We implemented decision procedures for permutation equivalence and projection equivalence in ProTeM, a recent tool [3] for manipulating proof terms. Automating permutation equivalence is non-trivial since the rewrite system for parallel standardization is only complete modulo *structural equivalence*. The latter is a weaker notion of equivalence that is easily decidable by means of a confluent and terminating rewrite system, but no rewrite system is known that avoids rewriting modulo.

In the next section we recall proof terms and define structural equivalence. Permutation equivalence is the topic of Sect. 3. In Sect. 4 we study the residual operation on proof terms and the related notions of projection order and projection equivalence. We present a variant of the residual system defined in [7,

This research is supported by FWF (Austrian Science Fund) project P27528.

Definition 8.7.54 and proof of Theorem 8.7.57] and [6, Definition 6.9 and proof of Theorem 6.12]. By imposing an innermost evaluation strategy, we ensure that our rewrite system has a well-defined rewrite semantics. We establish (innermost) confluence and termination, and use these properties to define projection order and projection equivalence. The extensions to ProTeM are described in Sect. 5 before we conclude in Sect. 6.

We assume familiarity with first-order term rewriting [1, 7] but knowledge of proof terms is not required. All definitions needed for this paper are given. Much more information on proof terms and notions of equivalence can be found in [7, Chapter 8]. Throughout the paper we deal with *left-linear* rewrite systems.

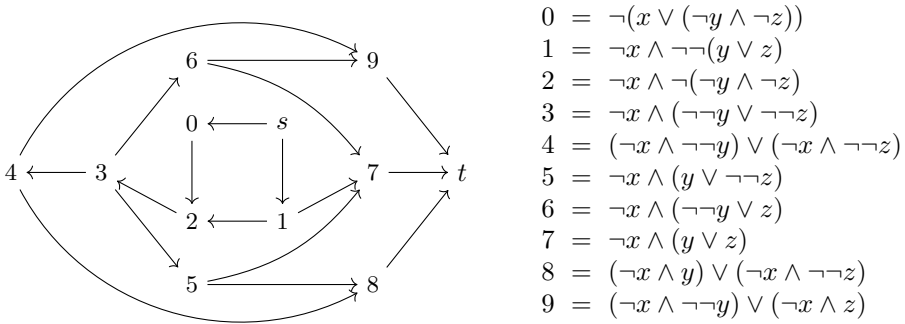
2 Proof Terms

Before formally defining proof terms, we give a motivating example that demonstrates their use. This example will reappear many times throughout the paper to illustrate the concepts we discuss.

Example 1. Consider the following TRS representing the necessary steps of computing the disjunctive normal form of a propositional formula:

$$\begin{array}{ll}
 \alpha & \neg(x \wedge y) \rightarrow \neg x \vee \neg y & \gamma & x \wedge (y \vee z) \rightarrow (x \wedge y) \vee (x \wedge z) \\
 \beta & \neg(x \vee y) \rightarrow \neg x \wedge \neg y & \delta & (x \vee y) \wedge z \rightarrow (x \wedge z) \vee (y \wedge z) \\
 \varepsilon & \neg\neg x \rightarrow x & &
 \end{array}$$

As illustrated by the diagram below there are 13 different rewrite sequences from $s = \neg(x \vee \neg(y \vee z))$ to $t = (\neg x \wedge y) \vee (\neg x \wedge z)$. If we want to compare them, for example to determine if some of them are equivalent, we can translate them into proof terms and do our analysis in the well-known realm of terms.



We refer to a specific sequence from s to t by listing the numbers of the intermediate terms. For instance, the sequence $s \rightarrow \neg x \wedge \neg\neg(y \vee z) \rightarrow \neg x \wedge (y \vee z) \rightarrow t$ is named 17.

Proof terms are built from function symbols, variables, rule symbols as well as the binary *composition* operator $;$ which is used in infix notation. Rule symbols represent rewrite rules and have a fixed arity which is the number of different variables in the represented rule. We use Greek letters $(\alpha, \beta, \gamma, \dots)$ as rule symbols, and uppercase letters (A, B, C, \dots) for proof terms. We can represent any rewrite sequence \rightarrow^* by a suitable proof term. A proof term without composition represents a multi-step (\Leftrightarrow) , a proof term without composition and nested rule symbols represents a parallel step $(\dashv\vdash)$, and a proof term without composition and only one rule symbol represents a single step (\rightarrow) . If a proof term contains neither compositions nor rule symbols, it denotes an empty step $(=)$.

If α is a rule symbol then lhs_α (rhs_α) denotes the left-hand (right-hand) side of the rewrite rule represented by α . Furthermore var_α denotes the list (x_1, \dots, x_n) of variables appearing in α in some fixed order. The length of this list is the arity of α . Given a rule symbol α with $\text{var}_\alpha = (x_1, \dots, x_n)$ and proof terms A_1, \dots, A_n , we write $\langle A_1, \dots, A_n \rangle_\alpha$ for the substitution $\{x_i \mapsto A_i \mid 1 \leq i \leq n\}$. A proof term A witnesses a rewrite sequence from its source $\text{src}(A)$ to its target $\text{tgt}(A)$, which are computed as follows:

$$\begin{aligned} \text{src}(x) &= \text{tgt}(x) = x & \text{src}(A ; B) &= \text{src}(A) & \text{tgt}(A ; B) &= \text{tgt}(B) \\ \text{src}(f(A_1, \dots, A_n)) &= f(\text{src}(A_1), \dots, \text{src}(A_n)) \\ \text{src}(\alpha(A_1, \dots, A_n)) &= \text{lhs}_\alpha \langle \text{src}(A_1), \dots, \text{src}(A_n) \rangle_\alpha \\ \text{tgt}(f(A_1, \dots, A_n)) &= f(\text{tgt}(A_1), \dots, \text{tgt}(A_n)) \\ \text{tgt}(\alpha(A_1, \dots, A_n)) &= \text{rhs}_\alpha \langle \text{tgt}(A_1), \dots, \text{tgt}(A_n) \rangle_\alpha \end{aligned}$$

Here f is an n -ary function symbol. The expression $\text{lhs}_\alpha \langle \text{src}(A_1), \dots, \text{src}(A_n) \rangle_\alpha$ denotes the result of replacing every variable x_i in the left-hand side of α with the source of the corresponding argument A_i of α . We assume $\text{tgt}(A) = \text{src}(B)$ whenever the composition $A ; B$ is used in a proof term. Proof terms A and B are *co-initial* if they have the same source. We omit parentheses in nested compositions in examples for better readability, assuming association to the right of the composition operator.

Example 2. The sequence 17 in Example 1 is represented by the proof term $\beta(x, \neg(y \vee z)); \neg x \wedge \epsilon(y \vee z); \gamma(\neg x, y, z)$. For the proof term $A = \alpha(\epsilon(x), \neg\epsilon(x))$ we have $\text{src}(A) = \neg(\neg\neg x \wedge \neg\neg\neg x)$ and $\text{tgt}(A) = \neg(x \vee \neg x)$. The proof term

$$\beta(x, \beta(y, z)); \neg x \wedge \alpha(\neg y, \neg z); \gamma(\neg x, \epsilon(y), \epsilon(z))$$

represents the sequence $s \Leftrightarrow 2 \rightarrow 3 \Leftrightarrow t$, which can be viewed as a compact version of 12348 and several other rewrite sequences from s to t . The expression $A ; \beta(x, x)$ is not a proof term since $\text{src}(\beta(x, x)) = \neg(x \vee x) \neq \text{tgt}(A)$.

Structural equivalence [7, Definition 8.3.1] equates proof terms that only differ in the left-to-right order in which steps are executed.

Definition 1. *The structural identities consist of the following four equation schemas:*

$$A; t \approx A \quad (1)$$

$$t; A \approx A \quad (2)$$

$$(A; B); C \approx A; (B; C) \quad (3)$$

$$f(A_1, \dots, A_n); f(B_1, \dots, B_n) \approx f(A_1; B_1, \dots, A_n; B_n) \quad (4)$$

Here t denotes a term without rule symbols and composition whereas f denotes an arbitrary function symbol in the underlying TRS. The induced congruence relation \equiv on proof terms is called structural equivalence. The instances of scheme (4) are known as functorial identities.

Structural equivalence is easily decidable by rewriting proof terms.

Definition 2. *The canonicalization TRS consists of the following rule schemas:*

$$A; t \rightarrow A \quad (5)$$

$$t; A \rightarrow A \quad (6)$$

$$(A; B); C \rightarrow A; (B; C) \quad (7)$$

$$f(A_1, \dots, A_n); f(B_1, \dots, B_n) \rightarrow f(A_1; B_1, \dots, A_n; B_n) \quad (8)$$

$$f(A_1, \dots, A_n); (f(B_1, \dots, B_n); C) \rightarrow f(A_1; B_1, \dots, A_n; B_n); C \quad (9)$$

Normal forms of the canonicalization TRS are called canonical.

Example 3. Returning to Example 1, the proof terms

$$(\neg x \wedge \epsilon(y)) \vee (\neg x \wedge \neg \neg z); (\neg x \wedge y) \vee (\neg x \wedge \epsilon(z)); (\neg x \wedge y) \vee (\neg x \wedge z)$$

$$(\neg x \wedge \neg y) \vee (\neg x \wedge \epsilon(z)); (\neg x \wedge \epsilon(y)) \vee (\neg x \wedge z); (\neg x \wedge y) \vee (\neg x \wedge z)$$

are structurally equivalent because both rewrite to the canonical proof term

$$(\neg x \wedge \epsilon(y)) \vee (\neg x \wedge \epsilon(z))$$

Theorem 1. *Canonical proof terms are unique representatives of structural equivalence classes.* \square

A proof sketch is given in [7, Exercise 8.3.6]. We remark that automatic tools for proving confluence and termination are not applicable here since the rules in Definition 2 are rule schemas; for every function symbol f in the signature and every term t of the underlying TRS, the rule schemas are suitably instantiated to obtain a concrete (and infinite) rewrite system that operates on proof terms of the underlying TRS. Nevertheless, standard confluence and termination techniques are readily applicable. In particular, schema (9) is added to make the critical pair between (7) and (8) convergent.

3 Permutation Equivalence

Adjacent steps in which the contracted redexes are at parallel positions can be swapped, which is captured by structural equivalence. Permutation equivalence [7, Definition 8.3.1] extends this by also allowing swapping adjacent steps in which the contracted redexes are above each other. This is similar to the variable overlap case in the well-known critical pair lemma.

Definition 3. *The permutation identities consist of the structural identities of Definition 1 together with the following two equation schemas:*

$$\alpha(A_1, \dots, A_n) \approx \text{lhs}_\alpha \langle A_1, \dots, A_n \rangle_\alpha ; \alpha(t_1, \dots, t_n) \quad (10)$$

$$\alpha(A_1, \dots, A_n) \approx \alpha(s_1, \dots, s_n) ; \text{rhs}_\alpha \langle A_1, \dots, A_n \rangle_\alpha \quad (11)$$

Here $\text{src}(A_i) = s_i$ and $\text{tgt}(A_i) = t_i$ and thus s_i and t_i are terms without rule symbols and compositions, for $i = 1, \dots, n$. Furthermore, α ranges over the rule symbols of the underlying TRS. The induced congruence relation on proof terms is denoted by \cong and called permutation equivalence. The permutation order \sqsubseteq is defined as follows: $A \sqsubseteq B$ if there exists a proof term C such that $A ; C \cong B$.

Example 4. We have $\neg x \wedge (\varepsilon(y) \vee \varepsilon(z)) ; \gamma(\neg x, y, z) \cong \gamma(\neg x, \varepsilon(y), \varepsilon(z))$ by an application of (10) from right to left (with $\alpha = \gamma$, $A_1 = \neg x$, $A_2 = \varepsilon(y)$, and $A_3 = \varepsilon(z)$). Hence $\neg x \wedge (\varepsilon(y) \vee \varepsilon(z)) \sqsubseteq \gamma(\neg x, \varepsilon(y), \varepsilon(z))$. Furthermore, $\gamma(\neg x, \varepsilon(y), \varepsilon(z)) \cong \gamma(\neg x, \neg\neg y, \neg\neg z) ; (\neg x \wedge \varepsilon(y)) \vee (\neg x \wedge \varepsilon(z))$ by using (11).

The following lemma generalizes the defining Eqs. (10) and (11). In ProTeM we use the second equation to move compositions inside arguments of rule symbols outside, which is necessary for translating proof terms into rewrite sequences.

Lemma 1. *For arbitrary proof terms A_1, \dots, A_n and B_1, \dots, B_n :*

$$\alpha(A_1 ; B_1, \dots, A_n ; B_n) \cong \text{lhs}_\alpha \langle A_1, \dots, A_n \rangle_\alpha ; \alpha(B_1, \dots, B_n)$$

$$\alpha(A_1 ; B_1, \dots, A_n ; B_n) \cong \alpha(A_1, \dots, A_n) ; \text{rhs}_\alpha \langle B_1, \dots, B_n \rangle_\alpha$$

Proof. To simplify the notation, we assume the arity n of α equals 1:

$$\alpha(A_1 ; B_1) \cong \text{lhs}_\alpha \langle A_1 ; B_1 \rangle_\alpha ; \alpha(\text{tgt}(B_1)) \quad (10)$$

$$\cong (\text{lhs}_\alpha \langle A_1 \rangle_\alpha ; \text{lhs}_\alpha \langle B_1 \rangle_\alpha) ; \alpha(\text{tgt}(B_1)) \quad (\star)$$

$$\cong \text{lhs}_\alpha \langle A_1 \rangle_\alpha ; (\text{lhs}_\alpha \langle B_1 \rangle_\alpha ; \alpha(\text{tgt}(B_1))) \quad (3)$$

$$\cong \text{lhs}_\alpha \langle A_1 \rangle_\alpha ; \alpha(B_1) \quad (10)$$

$$\alpha(A_1 ; B_1) \cong \alpha(\text{src}(A_1)) ; \text{rhs}_\alpha \langle A_1 ; B_1 \rangle_\alpha \quad (11)$$

$$\cong \alpha(\text{src}(A_1)) ; (\text{rhs}_\alpha \langle A_1 \rangle_\alpha ; \text{rhs}_\alpha \langle B_1 \rangle_\alpha) \quad (\star)$$

$$\cong (\alpha(\text{src}(A_1)) ; \text{rhs}_\alpha \langle A_1 \rangle_\alpha) ; \text{rhs}_\alpha \langle B_1 \rangle_\alpha \quad (3)$$

$$\cong \alpha(A_1) ; \text{rhs}_\alpha \langle B_1 \rangle_\alpha \quad (11)$$

In the steps labeled (\star) we use equation (4) repeatedly, depending on the structure of lhs_α and rhs_α . \square

The following lemma captures the connection between permutation equivalence and permutation order, a result that is mentioned in passing after the permutation order is introduced in [7, Definition 8.3.1].

Lemma 2. *For proof terms A and B , $A \cong B$ if and only if both $A \sqsubseteq B$ and $B \sqsubseteq A$.* \square

Standard Reductions are unique representatives of permutation equivalence classes, that are obtained by sorting rewrite steps in an outside-in and left-to-right order. For transforming reductions to outside-in order, called *parallel standard form*, the authors in [7, Section 8.5] propose two different approaches based on selection sort and insertion sort respectively. Since the latter, discussed in [7, Section 8.5.3], relies on proof terms it is of particular interest to us. Standard reductions are then obtained from parallel standard ones by imposing a left-to-right order when evaluating parallel steps.

Definition 4. *The parallel standardization TRS consists of the following rewrite schemas:*

$$\text{lhs}_\alpha \langle A_1, \dots, A_n \rangle_\alpha ; \alpha(t_1, \dots, t_n) \rightarrow \alpha(A_1, \dots, A_n) \quad (12)$$

$$\alpha(A_1, \dots, A_n) \rightarrow \alpha(s_1, \dots, s_n) ; \text{rhs}_\alpha \langle A_1, \dots, A_n \rangle_\alpha \quad (13)$$

These rules are applied modulo structural equivalence. The conditions on the symbols are the same as in Definition 3, but additionally we demand that in (13) at least one of A_1, \dots, A_n is not structurally equivalent to a proof term without rules symbols. A proof term is parallel standard if it is in normal form with respect to parallel standardization.

Parallel standardness is invariant with respect to structural equivalence by definition. As shown in the example below, structural equivalence is needed to move intermediate parallel reductions out of the way such that steps in the wrong order become adjacent. In particular, using canonical forms as representatives of structural equivalence classes, is not sufficient to compute parallel standard forms. This considerably complicates the automation of permutation equivalence.

Example 5. Consider $A = \varepsilon(x) \wedge \neg(y \wedge z) ; x \wedge \alpha(y, z) ; \gamma(x, \neg y, \neg z)$. The inner step $\varepsilon(x) \wedge \neg(y \wedge z)$ does not contribute to the outer step $\gamma(x, \neg y, \neg z)$ and hence these two steps need to be swapped to obtain a parallel standard normal form. To be able to apply the rules of the parallel standardization TRS, we first make the steps adjacent by moving the second step $x \wedge \alpha(y, z)$ out of the way with an appeal to structural equivalence:

$$\begin{aligned}
A &\equiv \varepsilon(x) \wedge \alpha(y, z); \gamma(x, \neg y, \neg z) \\
&\equiv \neg \neg x \wedge \alpha(y, z); \varepsilon(x) \wedge (\neg y \vee \neg z); \gamma(x, \neg y, \neg z) \\
&\rightarrow \neg \neg x \wedge \alpha(y, z); \gamma(\varepsilon(x), \neg y, \neg z) \\
&\rightarrow \neg \neg x \wedge \alpha(y, z); \gamma(\neg \neg x, \neg y, \neg z); (\varepsilon(x) \wedge \neg y) \vee (\varepsilon(x) \wedge \neg z)
\end{aligned}$$

The resulting proof term is parallel standard. Note that the canonical form of A is $\varepsilon(x) \wedge \alpha(y, z); \gamma(x, \neg y, \neg z)$, which is a normal form with respect to (12).

The conditions on A_1, \dots, A_n in rule (13) are there to avoid trivial cases of non-termination; e.g. $\gamma(y) \rightarrow \gamma(y); y \equiv \gamma(y)$ is excluded. In [7, Section 8.5] a proof sketch of the following result is given.

Theorem 2. *The parallel standardization TRS is complete modulo structural equivalence. \square*

Instead of rule (12), in our implementation we use the more liberal rewrite rule

$$\text{lhs}_\alpha \langle A_1, \dots, A_n \rangle_\alpha; \alpha(B_1, \dots, B_n) \rightarrow \alpha(A_1; B_1, \dots, A_n; B_n) \quad (14)$$

which is based on Lemma 1. Since we rewrite modulo structural equivalence, (14) simulates (12); simply substitute $\text{tgt}(A_i)$ for B_i . So for the case that the B_i do not contain rule symbols, the two rules behave exactly the same. If there is some rule symbol contained in one of the B_i , the term $\text{lhs}_\alpha \langle A_1, \dots, A_n \rangle_\alpha; \alpha(B_1, \dots, B_n)$ with $\text{tgt}(A_i) = \text{src}(B_i) = t_i$ for $1 \leq i \leq n$ always rewrites to a proof term that is structurally equivalent to $\alpha(\text{src}(A_1), \dots, \text{src}(A_n)); \text{rhs}_\alpha \langle A_1; B_1, \dots, A_n; B_n \rangle_\alpha$ independent of which of the two rules we use:

$$\begin{aligned} &\text{lhs}_\alpha \langle A_1, \dots, A_n \rangle_\alpha; \alpha(B_1, \dots, B_n) \\ &\rightarrow \text{lhs}_\alpha \langle A_1, \dots, A_n \rangle_\alpha; \alpha(t_1, \dots, t_n); \text{rhs}_\alpha \langle B_1, \dots, B_n \rangle_\alpha \end{aligned} \quad (13)$$

$$\rightarrow \alpha(A_1, \dots, A_n); \text{rhs}_\alpha \langle B_1, \dots, B_n \rangle_\alpha \quad (12)$$

$$\rightarrow \alpha(\text{src}(A_1), \dots, \text{src}(A_n)); \text{rhs}_\alpha \langle A_1, \dots, A_n \rangle_\alpha; \text{rhs}_\alpha \langle B_1, \dots, B_n \rangle_\alpha \quad (13)$$

and

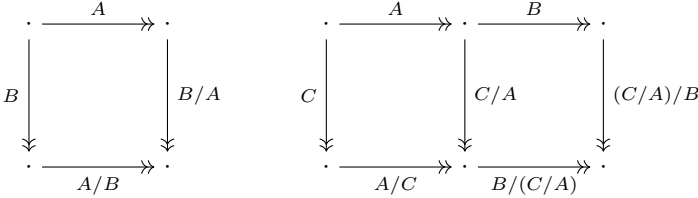
$$\begin{aligned} &\text{lhs}_\alpha \langle A_1, \dots, A_n \rangle_\alpha; \alpha(B_1, \dots, B_n) \\ &\rightarrow \alpha(A_1; B_1, \dots, A_n; B_n) \end{aligned} \quad (14)$$

$$\rightarrow \alpha(\text{src}(A_1), \dots, \text{src}(A_n)); \text{rhs}_\alpha \langle A_1; B_1, \dots, A_n; B_n \rangle_\alpha \quad (13)$$

Since it is not necessary to check whether the arguments of α are the targets of the A_i , rule (14) is easier to implement than rule (13). More details about the implementation can be found in Sect. 5.

4 Projection Equivalence

In the preceding section proof terms were declared to be equivalent if they can be obtained from each other by reordering (permuting) steps. In this section we give an account of *projection equivalence*, which is a completely different way of equating proof terms. It is based on the residual operation which computes which steps of A remain after performing B , for co-initial proof terms A and B . The steps in B need not be contained in A in order to compute their residual A / B . The diagram on the left shows a desirable result of residuals and the diagram on the right provides the intuition behind Eqs. (17) and (18) below:



In [7, Definition 8.7.54] the residual A / B is defined by means of the following equations:

$$x / x = x \tag{15}$$

$$f(A_1, \dots, A_n) / f(B_1, \dots, B_n) = f(A_1 / B_1, \dots, A_n / B_n)$$

$$\alpha(A_1, \dots, A_n) / \alpha(B_1, \dots, B_n) = \text{rhs}_\alpha \langle A_1 / B_1, \dots, A_n / B_n \rangle_\alpha$$

$$\alpha(A_1, \dots, A_n) / \text{lhs}_\alpha \langle B_1, \dots, B_n \rangle_\alpha = \alpha(A_1 / B_1, \dots, A_n / B_n) \tag{16}$$

$$\text{lhs}_\alpha \langle A_1, \dots, A_n \rangle_\alpha / \alpha(B_1, \dots, B_n) = \text{rhs}_\alpha \langle A_1 / B_1, \dots, A_n / B_n \rangle_\alpha$$

$$C / (A ; B) = (C / A) / B \tag{17}$$

$$(A ; B) / C = (A / C) ; (B / (C / A)) \tag{18}$$

$$A / B = \#(\text{tgt}(B)) \tag{otherwise}$$

Here $A, B, C, A_1, \dots, A_n, B_1, \dots, B_n$ are proof term *variables* that can be instantiated with arbitrary proof terms (so without $/$). The x in (15) denotes an *arbitrary* variable (in the underlying TRS), which cannot be instantiated.¹ In the final defining equation, $\#$ is the rule symbol of the special *error rule* $x \rightarrow \perp$. This rule is adopted to ensure that A / B is defined for arbitrary left-linear TRSs.² These defining equations are taken modulo (4) and

$$t ; t \approx t \tag{19}$$

The need for the functorial identities (4) is explained in the following example (Vincent van Oostrom, personal communication).

¹ In [7, Remark 8.2.21] variables are treated as constants and (15) is absent.

² In both [6, Definition 6.9] and [7, Definition 8.7.54] the wrong definition $A / B = \#(\text{tgt}(A))$ is given.

Example 6. Consider $A = f(g(\beta); g(\gamma))$ and $B = \alpha(a)$ in the TRS

$$\alpha: f(g(x)) \rightarrow x \quad \beta: a \rightarrow b \quad \gamma: b \rightarrow c$$

When computing A/B without (4), the α -instance $f(g(A_1))/\alpha(B_1) = A_1/B_1$ of schema (16) does not apply to A/B since the g in $f(g(A_1))$ needs to be extracted from $g(\alpha); g(\gamma)$ when computing A/B . As a consequence, the (otherwise) equation kicks in, producing the proof term $\#(b)$ that indicates an error. With (4) in place, the result of evaluating A/B is the proof term $\beta; \gamma$, representing the desired sequence $a \rightarrow b \rightarrow c$.

It is not immediately clear that the defining equations on the preceding page constitute a well-defined definition of the residual operation. In [7, proof of Theorem 8.7.57] the defining equations together with (4) and (19) are oriented from left to right, resulting in a rewrite system $\mathcal{R}es$ that is claimed to be terminating and confluent. The residual of A over B is then defined as the unique normal form of A/B in $\mathcal{R}es$.

There are two problems with this approach. First of all, when is the (otherwise) rule applied? In [7] this is not specified, resulting in an imprecise rewrite semantics of $\mathcal{R}es$. Keeping in mind that A/B is supposed to be a total operation on *proof terms* (so no $/$ in A and B), a natural solution is to adopt an innermost evaluation strategy. This ensures that C/A is evaluated before $(C/A)/B$ in the right-hand side of (17) and before $B/(C/A)$ in the right-hand side of (18). The (otherwise) condition is taken into account by imposing the additional restriction that the (otherwise) rule is applied to A/B (with A and B in normal form) only if the other rules are not applicable. The second, and more serious, problem is that $\mathcal{R}es$ is *not* confluent.

Example 7. Consider the TRS consisting of the rules

$$\alpha: f(x, y) \rightarrow f(y, x) \quad \beta: a \rightarrow b \quad \gamma: f(a, x) \rightarrow x$$

and the proof terms $A = f(\beta, a)$, $B = \alpha(b, \beta)$, $C = \alpha(a, a)$, and $D = \gamma(a)$. There are two ways to compute $(A; B)/(C; D)$, starting with (17) or (18):

$$\begin{aligned} ((A; B)/C)/D &\rightarrow ((A/C); (B/(C/A)))/D \\ &\rightarrow^* (f(a/a, \beta/a); (B/\alpha(a/\beta, a/a)))/D \\ &\rightarrow^* (f(a, \beta); (B/\alpha(b, a)))/D \\ &\rightarrow (f(a, \beta); f(\beta/a, b/b))/D \\ &\rightarrow^* (f(a, \beta); f(\beta, b))/D \rightarrow f(a; \beta, \beta; b)/D \rightarrow \#(a) \\ (A/(C; D)); (B/((C; D)/A)) & \\ &\rightarrow^* ((A/C)/D); (B/((C/A); (D/(A/C)))) \\ &\rightarrow^* (f(a, \beta)/D); (B/(\alpha(b, a); (D/f(a, \beta)))) \\ &\rightarrow^* \beta; (B/(\alpha(b, a); \gamma(b))) \rightarrow^* \beta; (f(\beta, b)/\gamma(b)) \\ &\rightarrow^* \beta; \#(b) \end{aligned}$$

The normal forms $\#(\mathbf{a})$ and $\beta; \#(\mathbf{b})$ represent different failing computations: $\mathbf{a} \rightarrow \perp$ and $\mathbf{a} \rightarrow \mathbf{b} \rightarrow \perp$.

To solve this problem we propose a drastic solution. When facing a term A / B with A and B in normal form, the defining equations are evaluated from top to bottom and the first equation that matches is applied. This essentially means that the ambiguity between (17) and (18) is resolved by giving preference to the former. Due to innermost evaluation, no other critical situations arise. So we arrive at the following definition, where we turned Eq. (19) into rule (28), which is possible due to the presence of (29).

Definition 5. *The residual TRS for proof terms consists of the following rules:*

$$x / x \rightarrow x \quad (20)$$

$$f(A_1, \dots, A_n) / f(B_1, \dots, B_n) \rightarrow f(A_1 / B_1, \dots, A_n / B_n) \quad (21)$$

$$\alpha(A_1, \dots, A_n) / \alpha(B_1, \dots, B_n) \rightarrow \text{rhs}_\alpha \langle A_1 / B_1, \dots, A_n / B_n \rangle_\alpha \quad (22)$$

$$\alpha(A_1, \dots, A_n) / \text{lhs}_\alpha \langle B_1, \dots, B_n \rangle_\alpha \rightarrow \alpha(A_1 / B_1, \dots, A_n / B_n) \quad (23)$$

$$\text{lhs}_\alpha \langle A_1, \dots, A_n \rangle_\alpha / \alpha(B_1, \dots, B_n) \rightarrow \text{rhs}_\alpha \langle A_1 / B_1, \dots, A_n / B_n \rangle_\alpha \quad (24)$$

$$C / (A ; B) \rightarrow (C / A) / B \quad (25)$$

$$(A ; B) / C \rightarrow (A / C) ; (B / (C / A)) \quad (26)$$

$$A / B \rightarrow \#(\text{tgt}(B)) \quad (27)$$

$$x ; x \rightarrow x \quad (28)$$

$$f(A_1, \dots, A_n) ; f(B_1, \dots, B_n) \rightarrow f(A_1 ; B_1, \dots, A_n ; B_n) \quad (29)$$

We adopt innermost evaluation with the condition that the rules (20)–(27) are evaluated from top to bottom.

The residual TRS operates on *closed* proof terms, which are proof terms without proof term variables, to ensure that $\text{tgt}(B)$ in the right-hand side of (27) can be evaluated. (Variables of the underlying TRS are allowed in proof terms.)

Lemma 3. *The residual TRS is terminating and confluent on closed proof terms.*

Proof. Confluence of the residual TRS is obvious because of the innermost evaluation strategy and the fact that there is no root overlap between its rules (due to the imposed evaluation order). Showing termination is non-trivial because of the nested occurrences of $/$ in the right-hand sides of (25) and (26). As suggested in [7, Exercise 8.7.58] one can use semantic labeling [8]. We take the well-founded algebra \mathcal{A} with carrier \mathbb{N} equipped with the standard order $>$ and the following weakly monotone interpretation and labeling functions:

$$\begin{aligned}
\alpha_{\mathcal{A}}(x_1, \dots, x_n) &= f_{\mathcal{A}}(x_1, \dots, x_n) = \max\{x_1, \dots, x_n\} \\
;_{\mathcal{A}}(x, y) &= x + y + 1 & /_{\mathcal{A}}(x, y) &= x & \#_{\mathcal{A}}(x) &= \perp_{\mathcal{A}} = 0 \\
L_{;} &= L_f = L_{\alpha} = L_{\#} = L_{\perp} = \emptyset & L_{/} &= \mathbb{N} & \text{lab}_{/}(x, y) &= x + y
\end{aligned}$$

The algebra \mathcal{A} is a quasi-model of the residual TRS. Hence termination is a consequence of termination of its labeled version. The latter follows from LPO with well-founded precedence $/_i > /_j$ for all $i > j$ and $/_0 > ; > f > \alpha > \# > \perp$ for all function symbols f and rule symbols α . For instance, (26) gives rise to the labeled versions $(A ; B) /_{a+b+c+1} C \rightarrow (A /_{a+c} C) ; (B /_{b+c} (C /_{c+a} A))$ for all natural numbers a , b , and c , and each of them is compatible with the given LPO. \square

The termination argument in the above proof does not depend on the imposed evaluation strategy. In the following we write $A ! B$ for the unique normal form of A / B .

Definition 6. *The projection order \lesssim and projection equivalence \simeq are defined on co-initial proof terms as follows: $A \lesssim B$ if $A ! B = \text{tgt}(B)$ and $A \simeq B$ if both $A \lesssim B$ and $B \lesssim A$.*

Lemma 3 provides us with an easy decision procedure for projection equivalence: $A \simeq B$ if and only if $A ! B$ and $B ! A$ coincide and contain neither rule symbols nor compositions.

Example 8. We can use this decision procedure to check which of the 13 sequences of Example 1 are projection equivalent. The (proof terms representing the) sequences 02357 and 12349 in Example 1 are projection equivalent since 02357/12349 and 12349/02357 rewrite to the same normal form $(\neg x \wedge y) \vee (\neg x \wedge z)$ in the residual TRS. As a matter of fact, all sequences from s to t are projection equivalent, except for 17. For instance, both 02357/17 and 17/02357 rewrite to $\#((\neg x \wedge y) \vee (\neg x \wedge z)) ; \#(\perp)$, but this normal form of the residual TRS contains the rule symbol $\#$ associated to the error rule.

Even though the residual TRS is designed to compute A/B for co-initial *proof terms*, there is no restriction on term formation. So in principle it is conceivable that $A ! B$ is not a well-formed proof term, which can only happen if $A ! B$ contains a subterm $A_1 ; A_2$ with $\text{tgt}(A_1) \neq \text{src}(A_2)$. The key properties that exclude this are $\text{src}(A ! B) = \text{tgt}(B)$ and $\text{tgt}(A ! B) = \text{tgt}(B ! A)$, because then the right-hand sides of rules (25) and (26) are well-defined, meaning that one obtains proof terms as normal forms if A , B , C are instantiated by proof terms. The first property ($\text{src}(A ! B) = \text{tgt}(B)$) can be proved by induction on the length of a normalizing sequence in the residual TRS starting from A / B . The second property ($\text{tgt}(A ! B) = \text{tgt}(B ! A)$, see also the diagrams at the beginning of this section) we have not yet been able to establish; the case where both A and B are headed by composition causes complications due to the imposed evaluation strategy.

5 Automation

In this section we describe the extensions to ProTeM³ that we implemented as part of this work. ProTeM is a tool for manipulating proof terms and has been previously described in [3], with the focus on proof terms that represent multisteps, so without composition, and methods for measuring overlap between multisteps.

Apart from the decision procedure for projection equivalence based on the residual TRS described in the previous section, we implemented procedures dealing with parallel standardization as well as algorithms to translate between rewrite sequences and proof terms.

5.1 Rewrite Sequences and Proof Terms

We implemented an algorithm that takes as input two terms t and u , and computes a proof term A without compositions such that $\text{src}(A) = t$ and $\text{tgt}(A) = u$. If there is no multi-step $t \rightarrow u$, A does not exist. Otherwise, there may be different proof terms A that satisfy the requirements. ProTeM returns the first solution it encounters by trying to match the rules of the current TRS in top-down order and recursively in the arguments. This algorithm is extended to generate a proof term for a sequence of multisteps. We do this by applying it to each consecutive pair of terms, resulting in proof terms A_1, \dots, A_k for a sequence consisting of $k + 1$ terms, which are then combined into $A = A_1 ; \dots ; A_k$.

Input: • proof term A
Output: • proof terms B_1, \dots, B_k without composition such that $A \cong B_1 ; \dots ; B_k$

```

if  $A = x$  then return  $A$ 
else if  $A = f(A_1, \dots, A_n)$  or  $A = \alpha(A_1, \dots, A_n)$  then
   $B_1 := \text{expand}(A_1); \dots ; B_n := \text{expand}(A_n);$ 
   $m :=$  maximum length of the sequences  $B_1, \dots, B_n;$ 
  for  $i = 1, \dots, n$  do
    extend  $B_i$  with  $m - \text{length}(B_i)$  copies of the term  $\text{tgt}(B_i)$ 
  if  $A = f(A_1, \dots, A_n)$  then return
     $f(B_1[1], \dots, B_n[1]), \dots, f(B_1[m], \dots, B_n[m])$ 
  else return
     $\alpha(B_1[1], \dots, B_n[1]), \text{rhs}_\alpha(B_1[2], \dots, B_n[2])_\alpha, \dots, \text{rhs}_\alpha(B_1[m], \dots, B_n[m])_\alpha$ 
else if  $A = A_1 ; A_2$  then return  $\text{expand}(A_1), \text{expand}(A_2)$ 

```

Fig. 1. Expansion algorithm (expand).

Conversely, for a given proof term A , ProTeM computes terms t_1, \dots, t_n such that A represents the sequence $t_1 \rightarrow \dots \rightarrow t_n$. To achieve this, first A is

³ <http://informatik-prottem.uibk.ac.at/>.

transformed into a permutation equivalent proof term $A_1 ; \dots ; A_n$ such that the A_i themselves do not contain compositions. To move inner compositions outside we repeatedly apply the functorial identities (4) and a generalized form of (11) (similar to the extension of (12) to (14)). We call this procedure *expansion*. Detailed steps are displayed in Fig. 1. The terms t_1, \dots, t_n are then obtained by computing the sources and targets of A_1, \dots, A_n . Expansion is also needed for the marking algorithm, presented in the next subsection. Here we give a simple example.

Example 9. Consider the TRS of Example 1. Expanding the proof term $A = \alpha(\beta(\neg x, \neg z); (\varepsilon(x) \wedge \varepsilon(y)), \varepsilon(z))$ yields the proof terms $A_1 = \alpha(\beta(\neg x, \neg z), \varepsilon(z))$ and $A_2 = \neg((\varepsilon(x) \wedge \varepsilon(y))) \vee \neg z$.

5.2 Standardization

In this subsection we report on ProTeM's implementations in connection with Sect. 3. When automating parallel standardization it is very useful to have some way of determining whether a given proof term is already parallel standard, other than going through all proof terms in its (theoretically infinite) structural equivalence class and trying to apply the parallel standardization rules. For this we use a modified version of the marking procedure [7, p. 366] that operates on proof terms instead of steps of a reduction. Our implementation is described in Fig. 2. We first transform the input A into its canonical form to get rid of trivial steps, then we use expansion to remove nested compositions and check if every proof term of the sequence A_1, \dots, A_n represents a parallel step (i.e., there are no nested rule symbols). Only then do we start with the actual marking. The basic idea is to go through the sequence A_1, \dots, A_n from left to right and mark the positions of the redexes. While moving right we check whether the next step contains markings below its redex pattern (i.e., in the arguments of its rule symbols). If it does we know that the next step takes place above the one that produced the marking and hence the given sequence of proof terms is not parallel standard.

Automating parallel standardization is a non-trivial task, since the rules of parallel standardization are applied modulo structural equivalence. Figure 3 displays our full algorithm to transform any proof term into a permutation equivalent parallel standard one. We start by computing the canonical form of our input A . Then we check if it is already parallel standard using the marking procedure. If not, we first apply the parallel standardization rules (13) and (14) as much as possible. If that does not result in a parallel standard proof term, a structurally equivalent proof term has to be computed to which we can again apply the parallel standardization rules. Structural equivalence classes are infinite, but only due to harmless compositions with trivial terms. Nevertheless, we do not search blindly through them. First we simplify our problem by determining which part of the proof term is not parallel standard and recursively call the parallel standardization algorithm on that subterm. When a composition $A_1 ; A_2$ is encountered where A_1 and A_2 are parallel standard but $A_1 ; A_2$ is not, neither A_1 nor A_2

Input: • proof term A
Output: • *yes* if A is parallel standard and *no* otherwise

$A_1, \dots, A_n := \text{expand}(\text{canonical}(A));$
if one of A_1, \dots, A_n contains nested rule symbols **then** return *no*;
 $M := \emptyset;$
for $i = 1, \dots, n$ **do**
 $P := \{p \in \mathcal{P}\text{os}(A_i) \mid A_i(p) \text{ is a rule symbol}\};$
 for $p \in P$ **do**
 $\alpha(t_1, \dots, t_m) := A_i|_p;$
 $t := \text{lhs}_\alpha\langle t_1, \dots, t_m \rangle_\alpha;$
 mark the symbols in $A_i[t]_p$ at positions in M that are below p ;
 if a symbol in t_1, \dots, t_m is marked **then** return *no*
 $M := \{q \in M \mid q \text{ is parallel to all positions in } P\} \cup P$
return *yes*

Fig. 2. Marking algorithm (mark).

can contain nested rules symbols since these would have been expanded by (13). Because we always compute canonical forms of proof terms before trying the parallel standardization rules, A_1 and A_2 cannot have the same function symbol as root. The fact that $A_1 ; A_2$ is not parallel standard further implies that A_1 is of the form $A_1 = f(T_1, \dots, T_n)$ and A_2 contains an outer step that must be performed before one of the inner steps in A_1 . We try to find a structurally equivalent proof term $A = C_1 ; (C_2 ; A_2)$ with $C_1 = f(T_1, \dots, \text{src}(T_i), \dots, T_n)$ and $C_2 = f(\text{tgt}(T_1), \dots, T_i, \dots, \text{tgt}(T_n))$ such that rule (13) is applicable to $C_2 ; A_2$. For each argument position i we first check if $C_2 ; A_2$ is already parallel standard to make sure not to perform useless steps which may cause non-termination of the procedure. If $C_2 ; A_2$ is parallel standard, we split A_1 at the next argument position. After we have identified C_1 and C_2 such that $C_2 ; A_2$ is not parallel standard, there is still the possibility that (13) is blocked, because C_2 contains composition. In that case C_2 is serialized into C_3 and C_4 such that $C_2 = C_3 ; C_4$ and C_4 contains exactly one rule symbol and no composition.

Since the parallel standardization TRS is terminating modulo structural equivalence (Theorem 2), its rules cannot be applied infinitely often to a proof term A and since we always perform at least one application of its rules in each iteration, our algorithm is bound to terminate after a finite number of steps.

Example 10. We apply the parallel standardization algorithm to the proof term A of Example 5. The canonical form of A is $A' = \varepsilon(x) \wedge \alpha(y, z); \gamma(x, \neg y, \neg z)$ and A' is not parallel standard according to the marking algorithm. Neither (12) nor (13) is applicable, though. Since both $\varepsilon(x) \wedge \alpha(y, z)$ and $\gamma(x, \neg y, \neg z)$ are parallel standard, we start splitting up $\varepsilon(x) \wedge \alpha(y, z)$ into $C_1 ; C_2$. For $i = 1$ we obtain $C_1 = \neg \neg x \wedge \alpha(y, z)$ and $C_2 = \varepsilon(x) \wedge (\neg y \vee \neg z)$, and so we try to apply (12) and (13) to the proof term $C_1 ; (C_2 ; A_2)$:

Input: • proof term A
Output: • canonical parallel standard proof term B such that $A \cong B$

```

A := canonical(A);
while mark(A) = false do
  B is result of applying rules (12) and (13) to A;
  if A ≠ B then A := canonical(B)
  else if A = A1 ; A2 and mark(A1) = false then
    A := canonical(ps(A1) ; A2)
  else if A = A1 ; A2 and mark(A2) = false then
    A := canonical(A1 ; ps(A2))
  else if A = f(A1, ..., An) then
    A := canonical(f(ps(A1), ..., ps(An)))
  else if A = A1 ; A2 then
    i := 0;
    repeat
      i++;
      C1 := f(T1, ..., src(Ti), ..., Tn);
      C2 := f(tgt(T1), ..., Ti, ..., tgt(Tn))
    until mark(C2 ; A2) = false;
    B is result of applying rules (14) and (13) to C1 ; (C2 ; A2);
    if A ≠ B then A := canonical(B)
  else
    C2 is serialized into C3 ; C4 such that C4 contains one rule symbol;
    B is result of applying rules (12) and (13) to C1 ; (C3 ; (C4 ; A2));
    A := canonical(B);
return A

```

Fig. 3. Parallel standardization algorithm (ps).

$$C_1 ; (C_2 ; A_2) \rightarrow \neg\neg x \wedge \alpha(y, z) ; \gamma(\varepsilon(x), \neg y, \neg z) \quad (12)$$

$$\rightarrow \neg\neg x \wedge \alpha(y, z) ; (\gamma(\neg\neg x, \neg y, \neg z) ; (\varepsilon(x) \wedge \neg y) \vee (\varepsilon(x) \wedge \neg z)) \quad (13)$$

At this point we are done since the final term is parallel standard.

We also implemented full standardization of proof terms by serializing the parallel steps of parallel standard proof terms such that steps are performed in a left-to-right order.

6 Conclusion

In this paper we described the extensions to ProTeM that deal with the permutation and projection equivalences as well as the projection order, important notions to compare rewrite sequences. Along the way, we corrected a mistake in [6, 7] concerning the well-definedness of the residual operation, which is used to decide projection equivalence.

This does not complete our investigations. We already remarked the difficulty of establishing $\text{tgt}(A ! B) = \text{tgt}(B ! A)$ which is needed to guarantee that A / B is a proper proof term. It is conceivable that the evaluation order we impose on the residual TRS needs to be relaxed to obtain this result. Then the error propagating rules $A ; \#(B) \rightarrow \#(A)$ and $\#(A) ; B \rightarrow \#(A)$ would be added to the residual TRS to resolve the non-confluence in Example 7. In addition the error rule $\# : x \rightarrow \perp$ would be promoted to the underlying TRS, in order to make $A ; \#(B)$, $\#(A) ; B$ and $\#(A)$ also permutation equivalent.

Another desirable result is a proof of equivalence of permutation and projection equivalence which is based on properties of the residual TRS. The question whether there exists a characterisation of permutation equivalence that avoids rewriting modulo structural equivalence is also worth investigating. Further, the complexity of computing (parallel) standard reductions and residuals needs to be investigated.

Acknowledgments. We thank Vincent van Oostrom and members of the master seminar of the Computational Logic research group for insightful discussions. Comments by the reviewers helped to improve the presentation.

References

1. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998). <https://doi.org/10.1017/CBO9781139172752>
2. Hirokawa, N., Middeldorp, A.: Decreasing diagrams and relative termination. *J. Autom. Reasoning* **47**(4), 481–501 (2011). <https://doi.org/10.1007/s10817-011-9238-x>
3. Kohl, C., Middeldorp, A.: ProTeM: a proof term manipulator (system description). In: Kirchner, H. (ed.) *Proceedings of 3rd International Conference on Formal Structures for Computation and Deduction*. Leibniz International Proceedings in Informatics, vol. 108, pp. 31:1–31:8 (2018). <https://doi.org/10.4230/LIPIcs.FSCD.2018.31>
4. Martí-Oliet, N., Meseguer, J.: Rewriting logic: roadmap and bibliography. *Theoret. Comput. Sci.* **285**(2), 121–154 (2002). [https://doi.org/10.1016/S0304-3975\(01\)00357-7](https://doi.org/10.1016/S0304-3975(01)00357-7)
5. Meseguer, J.: Conditioned rewriting logic as a united model of concurrency. *Theoret. Comput. Sci.* **96**(1), 73–155 (1992). [https://doi.org/10.1016/0304-3975\(92\)90182-F](https://doi.org/10.1016/0304-3975(92)90182-F)
6. van Oostrom, V., de Vrijer, R.: Four equivalent equivalences of reductions. In: *Proceedings of 2nd International Workshop on Reduction Strategies in Rewriting and Programming*. *Electronic Notes in Theoretical Computer Science*, vol. 70(6), pp. 21–61 (2002). [https://doi.org/10.1016/S1571-0661\(04\)80599-1](https://doi.org/10.1016/S1571-0661(04)80599-1)
7. Terese (ed.): *Term Rewriting Systems*, Cambridge Tracts in Theoretical Computer Science, vol. 55. Cambridge University Press (2003)
8. Zantema, H.: Termination of term rewriting by semantic labelling. *Fundamenta Informaticae* **24**, 89–105 (1995). <https://doi.org/10.3233/FI-1995-24124>

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

