ProTeM: A Proof Term Manipulator

Christina Kohl
Department of Computer Science, University of Innsbruck, Austria
christina.kohl@uibk.ac.at
https://orcid.org/0000-0002-8470-2485

Aart Middeldorp
Department of Computer Science, University of Innsbruck, Austria
aart.middeldorp@uibk.ac.at
https://orcid.org/0000-0001-7366-8464

Abstract

Proof terms are a useful concept for reasoning about computations in term rewriting. Human calculation with proof terms is tedious and error-prone. We present ProTeM, a new tool that offers support for manipulating proof terms that represent multisteps in left-linear rewrite systems.

2012 ACM Subject Classification Theory of computation → Rewrite systems, Theory of computation → Equational logic and rewriting

Keywords and phrases Proof terms, term rewriting, interactive tool

Digital Object Identifier 10.4230/LIPIcs.FSCD.2018.31

Category System description

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

Acknowledgements We are grateful to the reviewers for suggesting improvements to the interface of ProTeM.

1 Introduction

Proof terms represent computations in term rewriting. They were introduced by van Oostrom and de Vrijer for first-order left-linear rewrite systems to study equivalence of reductions in [12] and [9, Chapter 8]. Extensions to higher-order rewriting and infinitary rewriting are reported in [1] and [5], respectively. Hirokawa and the second author used proof terms for confluence analysis of left-linear rewrite systems [2, 3].

Our motivation for studying proof terms is to close an important gap between proofs produced by automatic confluence checkers and *certified* proofs. Numerous confluence criteria described in the literature have been formalized in IsaFoR, a large Isabelle/HOL library for term rewriting, see [7] for a recent overview. This includes the well-known result of Huet [4] stating that a left-linear rewrite system is confluent if its critical pairs are closed by a parallel step [8]. Its extension to multisteps (also called development steps) by van Oostrom [11] thus far escaped all attempts to obtain a formalized proof. The picture proof in [11] conveys the intuition but is very hard to formalize in a modern proof assistant. We believe that proof terms together with residual theory [9, Section 8.7] will help to close the gap.

Calculations with proof terms are tedious and error-prone to do by hand, which is why we developed ProTeM. Besides providing basic operations for manipulating proof terms that represent multisteps in left-linear rewrite systems, like join and residual, ProTeM supports new operations on proof terms that are required for a formalized proof of the main result of

[11]. The latter include an inductive definition for computing the amount of overlap and a function that returns the critical overlaps between co-initial proof terms.

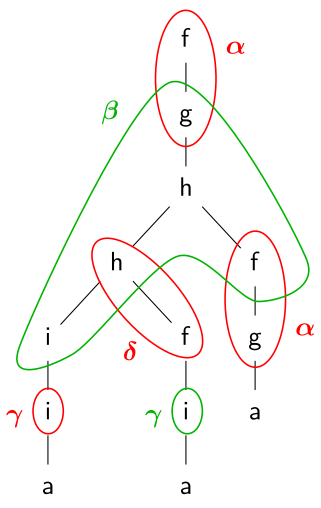
In the next section we recall proof terms and introduce new operations for measuring overlap between two proof terms. The web interface of ProTeM is described in Section 3 and in Section 4 we present some implementation details. We conclude in Section 5 with ideas for future extensions of ProTeM.

2 Proof Terms

Proof terms are built from function symbols, variables, and rule symbols. The latter represent rewrite rules and have a fixed arity which is the number of different variables in the represented rule. We use Greek letters as rule symbols. In this section we present the operations on proof terms that are implemented in ProTeM. The following example will be used to illustrate various definitions.

▶ **Example 1.** Consider the rewrite rules

$$\begin{aligned}
\boldsymbol{\alpha}\colon&\quad \mathsf{f}(\mathsf{g}(x)) \to \mathsf{g}(\mathsf{h}(x,\mathsf{i}(\mathsf{a})))\\
\boldsymbol{\beta}\colon&\quad \mathsf{g}(\mathsf{h}(\mathsf{h}(\mathsf{i}(x),y),\mathsf{f}(z))) \to \mathsf{h}(\mathsf{h}(y,y),\mathsf{f}(z))\\
\boldsymbol{\gamma}\colon&\quad \mathsf{i}(x) \to x\\
\boldsymbol{\delta}\colon&\quad \mathsf{h}(x,\mathsf{f}(y)) \to \mathsf{h}(\mathsf{i}(\mathsf{f}(y)),\mathsf{f}(y))\\
\boldsymbol{\varepsilon}\colon&\quad \mathsf{g}(\mathsf{h}(x,y)) \to \mathsf{h}(x,y)
\end{aligned}$$



and the term $s = \mathsf{f}(\mathsf{g}(\mathsf{h}(\mathsf{h}(\mathsf{i}(\mathsf{i}(\mathsf{a})),\mathsf{f}(\mathsf{i}(\mathsf{a}))),\mathsf{f}(\mathsf{g}(\mathsf{a})))))$. By marking certain redexes in s we obtain the two proof terms

$$\begin{aligned}
A &= \boldsymbol{\alpha}(\mathsf{h}(\boldsymbol{\delta}(\mathsf{i}(\boldsymbol{\gamma}(\mathsf{a})),\mathsf{i}(\mathsf{a})),\boldsymbol{\alpha}(\mathsf{a})))\\
B &= \mathsf{f}(\boldsymbol{\beta}(\mathsf{i}(\mathsf{a}),\mathsf{f}(\boldsymbol{\gamma}(\mathsf{a})),\mathsf{g}(\mathsf{a})))
\end{aligned}$$

This situation is illustrated on the right, where the redexes in A are indicated in red and those in B in green.

If α is a rule symbol then $\mathsf{lhs}(\alpha)$ ($\mathsf{rhs}(\alpha)$) denotes the left-hand (right-hand) side of the rewrite rule represented by α. Furthermore $\mathsf{var}(\alpha)$ denotes the list $(x_1,\ldots,x_n)$ 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 $\mathsf{var}(\alpha) = (x_1,\ldots,x_n)$ and terms $t_1,\ldots,t_n$, we write $\langle t_1,\ldots,t_n\rangle_\alpha$ for the substitution $\{x_i \mapsto t_i \mid 1 \leqslant i \leqslant n\}$. A proof term A witnesses a multistep from its source $\mathsf{src}(A)$ to its target $\mathsf{tgt}(A)$, which are computed as follows:

$$\begin{aligned}
\mathsf{src}(x) = \mathsf{tgt}(x) &= x\\
\mathsf{src}(f(A_1,\ldots,A_n)) &= f(\mathsf{src}(A_1),\ldots,\mathsf{src}(A_n))\\
\mathsf{src}(\alpha(A_1,\ldots,A_n)) &= \mathsf{lhs}(\alpha)\langle\mathsf{src}(A_1),\ldots,\mathsf{src}(A_n)\rangle_\alpha\\
\mathsf{tgt}(f(A_1,\ldots,A_n)) &= f(\mathsf{tgt}(A_1),\ldots,\mathsf{tgt}(A_n))\\
\mathsf{tgt}(\alpha(A_1,\ldots,A_n)) &= \mathsf{rhs}(\alpha)\langle\mathsf{tgt}(A_1),\ldots,\mathsf{tgt}(A_n)\rangle_\alpha
\end{aligned}$$

Here f is an n-ary function symbol. Proof terms A and B are *co-initial* if they have the same source. We define the *orthogonality* predicate $A \perp B$ by the following clauses:

$$\begin{aligned} x &\perp x \\ f(A_1,\ldots,A_n) \perp f(B_1,\ldots,B_n) &\iff A_i \perp B_i \text{ for all } 1 \leqslant i \leqslant n \\ \alpha(A_1,\ldots,A_n) \perp \mathsf{lhs}(\alpha)\langle B_1,\ldots,B_n\rangle_\alpha &\iff A_i \perp B_i \text{ for all } 1 \leqslant i \leqslant n \\ \mathsf{lhs}(\alpha)\langle A_1,\ldots,A_n\rangle_\alpha \perp \alpha(B_1,\ldots,B_n) &\iff A_i \perp B_i \text{ for all } 1 \leqslant i \leqslant n \end{aligned}$$

In all other cases $A \perp B$ is false. Next we recall the join $(A \sqcup B)$ and residual $(A \,/\, B)$ operations on co-initial proof terms:

$$\begin{aligned} x \sqcup x = x \,/\, x &= x \\ f(A_1,\ldots,A_n) \sqcup f(B_1,\ldots,B_n) &= f(A_1 \sqcup B_1,\ldots,A_n \sqcup B_n) \\ \alpha(A_1,\ldots,A_n) \sqcup \alpha(B_1,\ldots,B_n) &= \alpha(A_1 \sqcup B_1,\ldots,A_n \sqcup B_n) \\ \alpha(A_1,\ldots,A_n) \sqcup \mathsf{lhs}(\alpha)\langle B_1,\ldots,B_n\rangle_\alpha &= \alpha(A_1 \sqcup B_1,\ldots,A_n \sqcup B_n) \\ \mathsf{lhs}(\alpha)\langle A_1,\ldots,A_n\rangle_\alpha \sqcup \alpha(B_1,\ldots,B_n) &= \alpha(A_1 \sqcup B_1,\ldots,A_n \sqcup B_n) \\ f(A_1,\ldots,A_n) \,/\, f(B_1,\ldots,B_n) &= f(A_1 \,/\, B_1,\ldots,A_n \,/\, B_n) \\ \alpha(A_1,\ldots,A_n) \,/\, \alpha(B_1,\ldots,B_n) &= \mathsf{rhs}(\alpha)\langle A_1 \,/\, B_1,\ldots,A_n \,/\, B_n\rangle_\alpha \\ \alpha(A_1,\ldots,A_n) \,/\, \mathsf{lhs}(\alpha)\langle B_1,\ldots,B_n\rangle_\alpha &= \alpha(A_1 \,/\, B_1,\ldots,A_n \,/\, B_n) \\ \mathsf{lhs}(\alpha)\langle A_1,\ldots,A_n\rangle_\alpha \,/\, \alpha(B_1,\ldots,B_n) &= \mathsf{rhs}(\alpha)\langle A_1 \,/\, B_1,\ldots,A_n \,/\, B_n\rangle_\alpha \end{aligned}$$

These are partial operations. The next operation that we define on proof terms is *deletion* $A - B$, which is used to remove steps from a multistep:

$$\begin{aligned} x - x &= x \\ f(A_1,\ldots,A_n) - f(B_1,\ldots,B_n) &= f(A_1 - B_1,\ldots,A_n - B_n) \\ \alpha(A_1,\ldots,A_n) - \alpha(B_1,\ldots,B_n) &= \mathsf{lhs}(\alpha)\langle A_1 - B_1,\ldots,A_n - B_n\rangle_\alpha \\ \alpha(A_1,\ldots,A_n) - \mathsf{lhs}(\alpha)\langle B_1,\ldots,B_n\rangle_\alpha &= \alpha(A_1 - B_1,\ldots,A_n - B_n) \end{aligned}$$

Like join and residual, deletion is a partial operation.

▶ **Example 2.** The proof terms A and B in Example 1 are not orthogonal. Let $C = B - \mathsf{f}(\beta(\mathsf{i}(\mathsf{a}),\mathsf{f}(\mathsf{i}(\mathsf{a})),\mathsf{g}(\mathsf{a}))) = \mathsf{f}(\mathsf{g}(\mathsf{h}(\mathsf{h}(\mathsf{i}(\mathsf{i}(\mathsf{a})),\mathsf{f}(\gamma(\mathsf{a})),\mathsf{f}(\mathsf{g}(\mathsf{a}))))))$. We have $A \perp C$. Moreover,

$$\begin{aligned} A \,/\, C &= \alpha(\mathsf{h}(\delta(\mathsf{i}(\gamma(\mathsf{a})),\mathsf{a}),\alpha(\mathsf{a}))) \\ C \,/\, A &= \mathsf{g}(\mathsf{h}(\mathsf{h}(\mathsf{h}(\mathsf{i}(\mathsf{f}(\gamma(\mathsf{a}))),\mathsf{f}(\gamma(\mathsf{a}))),\mathsf{g}(\mathsf{h}(\mathsf{a},\mathsf{i}(\mathsf{a})))),\mathsf{i}(\mathsf{a}))) \end{aligned}$$

An important concept in the correctness proof of the confluence theorem in [11] is the amount of overlap between two multisteps. Below we present an inductive definition for measuring the overlap between co-initial proof terms. It is based on a special labeling of the source of a proof term. We write $\mathsf{lhs}^\sharp(\alpha)$ for the result of labeling every function symbol in $\mathsf{lhs}(\alpha)$ with α as well as the distance to the root of α: $\mathsf{lhs}^\sharp(\alpha) = \varphi(\mathsf{lhs}(\alpha),\alpha,0)$ with

$$\varphi(t,\alpha,i) = \begin{cases} t & \text{if } t \in \mathcal{V} \\ f_{\alpha^i}(\varphi(t_1,\alpha,i+1),\ldots,\varphi(t_n,\alpha,i+1)) & \text{if } t = f(t_1,\ldots,t_n) \end{cases}$$

The mapping $\mathsf{src}^\sharp$ computes the labeled source of a proof term:

$$\begin{aligned} \mathsf{src}^\sharp(x) &= x \\ \mathsf{src}^\sharp(f(A_1,\ldots,A_n)) &= f(\mathsf{src}^\sharp(A_1),\ldots,\mathsf{src}^\sharp(A_n)) \\ \mathsf{src}^\sharp(\alpha(A_1,\ldots,A_n)) &= \mathsf{lhs}^\sharp(\alpha)\langle \mathsf{src}^\sharp(A_1),\ldots,\mathsf{src}^\sharp(A_n)\rangle_\alpha \end{aligned}$$

Given two co-initial proof terms A and B, the following function computes a single labeled term in which all function symbols corresponding to redex patterns in A and B are marked:

$$\mathsf{merge}(A,B) = \mathsf{merge}'(\mathsf{src}^\sharp(A), \mathsf{src}^\sharp(B))$$

with $\mathsf{merge}'(s,t) = s$ for $s,t \in \mathcal{V}$ and $\mathsf{merge}'(s,t) = f_{ab}(\mathsf{merge}'(s_1,t_1),\ldots,\mathsf{merge}'(s_n,t_n))$ if $s = f_a(s_1,\ldots,s_n)$ and $t = f_b(t_1,\ldots,t_n)$. Here we identify an unlabeled function symbol f with f_-. The merge function is used to measure the amount of overlap between co-initial proof terms: $\blacktriangle(A,B) = \mathsf{measure}(\mathsf{merge}(A,B))$ with $\mathsf{measure}(u) = 0$ if $u \in \mathcal{V}$ and

$$\mathsf{measure}(f_{a^k b^l}(u_1,\ldots,u_n)) = \begin{cases} 1 + \sum_{i=1}^{n} \mathsf{measure}(u_i) & \text{if } a^k \neq - \text{ and } b^l \neq - \\ \sum_{i=1}^{n} \mathsf{measure}(u_i) & \text{otherwise} \end{cases}$$

Finally the overlaps function collects all pairs of overlapping redexes in co-initial proof terms:

$$\mathsf{overlaps}(A,B) = \left\{ (p,\alpha,q,\beta) \;\middle|\; \begin{array}{l} p,q \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(u),\ \ell_1(u(p)) = \alpha^0,\ \ell_2(u(q)) = \beta^0, \text{ and either} \\ p \leqslant q \text{ and } \ell_1(u(q)) = \alpha^{|q\backslash p|} \text{ or } q < p \text{ and } \ell_2(u(p)) = \beta^{|p\backslash q|} \end{array} \right\}$$

Here $u = \mathsf{merge}(A,B)$ and $\mathcal{P}\mathsf{os}_{\mathcal{F}}(u)$ is the set of function positions in u. The functions ℓ_1 and ℓ_2 extract the first and second label of a labeled function symbol: $\ell_1(f_{ab}) = a$ and $\ell_2(f_{ab}) = b$. The condition $\ell_1(u(q)) = \alpha^{|q\backslash p|}$ in the first case of the definition of $\mathsf{overlaps}(A,B)$ ensures that $q\backslash p$ is a position in $\mathsf{lhs}(\alpha)$.

▶ **Example 3.** For the proof terms in Example 1 we have

$$\mathsf{merge}(A,B) = \mathsf{f}_{\boldsymbol{\alpha}^0 -}(\mathsf{g}_{\boldsymbol{\alpha}^1 \boldsymbol{\beta}^0}(\mathsf{h}_{-\boldsymbol{\beta}^1}(\mathsf{h}_{\boldsymbol{\delta}^0 \boldsymbol{\beta}^2}(\mathsf{i}_{-\boldsymbol{\beta}^3}(\mathsf{i}_{\boldsymbol{\gamma}^0 -}(\mathsf{a})), \mathsf{f}_{\boldsymbol{\delta}^1 -}(\mathsf{i}_{-\boldsymbol{\gamma}^0}(\mathsf{a}))), \mathsf{f}_{\boldsymbol{\alpha}^0 \boldsymbol{\beta}^2}(\mathsf{g}_{\boldsymbol{\alpha}^1 -}(\mathsf{a})))))$$

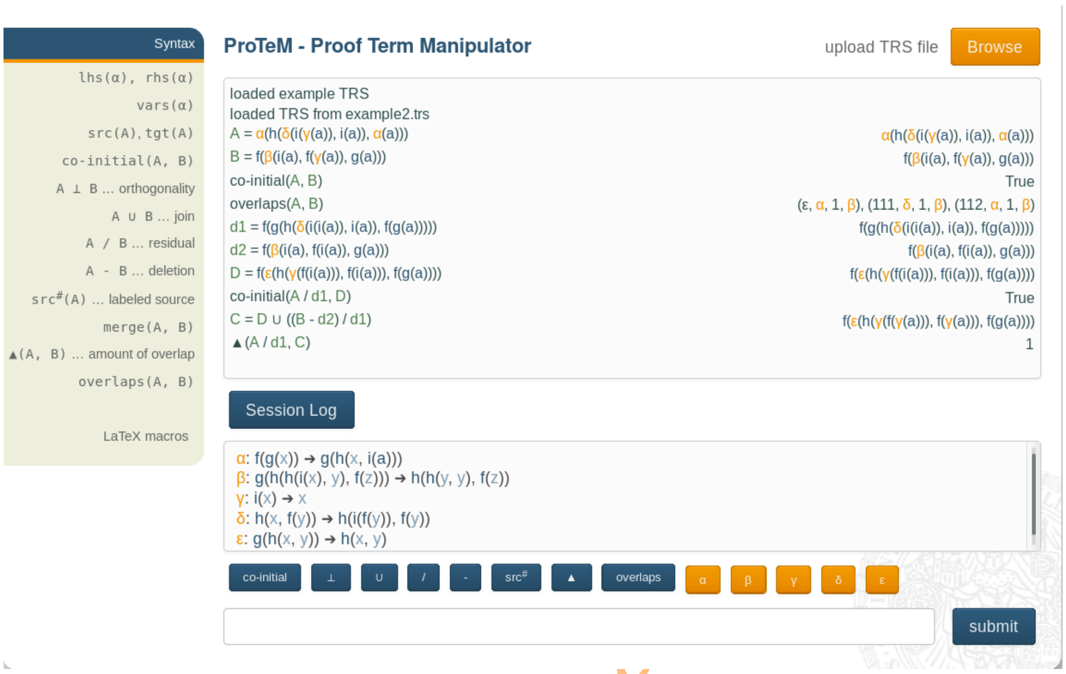
and $\blacktriangle(A,B) = 3$. Here a abbreviates a_{--}. Furthermore, $\mathsf{overlaps}(A,B)$ consists of the tuples $(\epsilon, \boldsymbol{\alpha}, 1, \boldsymbol{\beta})$, $(111, \boldsymbol{\delta}, 1, \boldsymbol{\beta})$, and $(112, \boldsymbol{\alpha}, 1, \boldsymbol{\beta})$.

3 Web Interface

In this section we will first give a brief overview of the main parts of ProTeM's user interface, subsequently we will describe all features in more detail. The web interface of ProTeM can be accessed at

`http://informatik-protem.uibk.ac.at/`

The layout of our application is displayed in Figure 1. At the center of the screen we have a large area for displaying the history of commands a user has entered (on the left), together with result output corresponding to these commands (on the right). We also offer the possibility to export the commands and results of the current session as a simple text file via the "Session Log" button underneath the output area. Below that there is a smaller panel where all rules of the currently loaded term rewrite system are displayed. At the bottom of the screen we have a command line with several buttons above it, that help users enter unusual symbols such as Greek letters for rule symbols or the $\perp$ symbol for the orthogonality predicate on proof terms. To the left of the screen we have a sidebar that gives an overview of the syntax that is used for commands. In the navigation bar at the top right corner of the screen we have a link leading to a help page with details about every component and feature of ProTeM.



```
loaded example TRS
loaded TRS from example2.trs
A = α(h(δ(i(γ(a)), i(a)), α(a)))                              α(h(δ(i(γ(a)), i(a)), α(a)))
B = f(β(i(a), f(γ(a)), g(a)))                                    f(β(i(a), f(γ(a)), g(a)))
co-initial(A, B)                                                                      True
overlaps(A, B)                                  (ε, α, 1, β), (111, δ, 1, β), (112, α, 1, β)
d1 = f(g(h(δ(i(i(a)), i(a)), f(g(a)))))                    f(g(h(δ(i(i(a)), i(a)), f(g(a)))))
d2 = f(β(i(a), f(i(a)), g(a)))                                  f(β(i(a), f(i(a)), g(a)))
D = f(ε(h(γ(f(i(a))), f(i(a))), f(g(a))))                f(ε(h(γ(f(i(a))), f(i(a))), f(g(a))))
co-initial(A / d1, D)                                                                 True
C = D ∪ ((B - d2) / d1)                                  f(ε(h(γ(f(γ(a))), f(γ(a))), f(g(a))))
▲(A / d1, C)                                                                             1
```

```
α: f(g(x)) → g(h(x, i(a)))
β: g(h(h(i(x), y), f(z))) → h(h(y, y), f(z))
γ: i(x) → x
δ: h(x, f(y)) → h(i(f(y)), f(y))
ε: g(h(x, y)) → h(x, y)
```

Figure 1 Screenshot of a ProTeM session.

3.1 Uploading a Term Rewrite System

When first opening the website, a simple example rewrite system is loaded per default. Users can upload their own rewrite systems from `.trs` files. The files need to correspond to a simplified form of the standard TRS-format as described in [6], where only the `VAR` and `RULES` sections are taken into account. Additionally the rule symbols ProTeM should use can be specified in the file by prepending each rule with its corresponding symbol followed by a colon. For reference, an example `.trs` file is available in the help section of the tool. If one or more rules have no specified rule symbols, ProTeM chooses a new Greek letter for each rule, starting from α. In cases where there are more than 24 rules, ProTeM begins to append digits to each Greek letter, e.g. $\alpha 1$, $\beta 1$, $\gamma 1$, ... When uploading a new rewrite system, the buttons above the command line will automatically change according to the new rule symbols.

3.2 Commands

There are two types of commands available, one are assignments, the other computations on proof terms. Assignments have syntax `id = proofterm` where `id` can be any string and `proofterm` any valid proof term. Notably it is possible to use nested expressions in assignments (e.g. $C = D \sqcup ((B - \mathrm{d2})/\mathrm{d1})$, see also Figure 1). Commands for rule symbols are `lhs`(α), `rhs`(α) and `vars`(α) where α can be any rule symbol used in the current TRS. Commands for proof terms include all operations described in Section 2 and their nested applications. The syntax of these operations is listed in the sidebar of our application.

Commands have to be entered into the text field at the bottom of the screen. The blue buttons above it can be used to enter special symbols that are used for some of the commands (like $\perp$ for orthogonality, or ▲ for measuring the amount of overlap between two proof terms). In addition there is one orange button for each currently used rule symbol.

When pressing one of the buttons, the corresponding symbol appears in the command line, with the focus returning immediately to the text field itself so that the user can carry on typing. A command can be submitted either by pressing enter or by using the "Submit" button. If the command line contains a valid command, it will be sent to the server and executed. The result will then be displayed in the output area above. If the command was not valid (e.g. trying to assign the result of an undefined operation), an error message will be displayed (see Figure 2 in Appendix A).

3.3 Export to LaTeX

A proof term or labeled proof term can be exported as a LaTeX string. To correctly insert proof terms from ProTeM into a LaTeX document it is first necessary to add the required macros. These define colors and provide support for UTF8 encoding of Greek letters. In particular we define three new commands `\pfun`, `\pvar`, `\prule` which define the representations of function symbols, variables and rule symbols respectively. The macros can be downloaded by clicking on the "LaTeX macros" entry in the sidebar. Clicking on any proof term in the output area will open a popup view, which contains a text field with the LaTeX representation of that proof term (see Figure 3 in Appendix A). It can then be copy-and-pasted into any document.

4 Implementation Details

The core functionality of ProTeM is written in Scala. For the web component we used the Vaadin framework [10]. Vaadin is a Java web application framework that makes it easier for developers who don't have much experience with web technologies, such as JavaScript, HTML and HTTP requests, to design responsive and interactive web applications. Vaadin allows developers to write all required code in pure Java (or any other language that runs on the JVM). Applications can also be extended with custom HTML or JavaScript and themed with CSS. From a technological point of view the UI logic of a Vaadin application runs as a Java Servlet in a Java application server. On the client side Vaadin uses JavaScript to render the user interface in the browser and communicate user events to the server. All communication is automated and makes heavy use of AJAX (Asynchronous JavaScript and XML) to make applications as responsive as possible. An additional benefit for our particular application was that Vaadin automatically stores the state of each user session (as long as the browser window is open), so that we can provide users with an interactive interface and still call our Scala functions on the server for all computations on proof terms.

5 Conclusion

In this paper we presented ProTeM, a tool that supports operations on proof terms that represent multisteps in first-order left-linear rewrite systems. There are several possibilities to extend the functionality of ProTeM. First of all, adding a composition operation to the language of proof terms allows to represent rewrite sequences that are not single multisteps. Equivalence testing and normalization become then interesting questions. Also one could ask the tool to compute proof terms that represent a given rewrite sequence. Another useful extension will be automatic support for visualizing co-initial proof terms, like the figure in Example 1. Dropping the left-linearity requirement will be a challenging task, which requires the development of new theory.

References

1 H.J. Sander Bruggink. *Equivalence of Reductions in Higher-Order Rewriting*. PhD thesis, Utrecht University, 2008.

2 Nao Hirokawa and Aart Middeldorp. Decreasing diagrams and relative termination. *J. Autom. Reasoning*, 47(4):481–501, 2011. doi:10.1007/s10817-011-9238-x.

3 Nao Hirokawa and Aart Middeldorp. Commutation via relative termination. In *Proc. 2nd International Workshop on Confluence*, pages 29–33, 2013.

4 Gérard P. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems. *J. ACM*, 27(4):797–821, 1980. doi:10.1145/322217.322230.

5 Carlos Lombardi, Alejandro Ríos, and Roel de Vrijer. Proof terms for infinitary rewriting. In Gilles Dowek, editor, *Rewriting and Typed Lambda Calculi - Joint International Conference, RTA-TLCA 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8560 of *Lecture Notes in Computer Science*, pages 303–318. Springer, 2014. doi:10.1007/978-3-319-08918-8_21.

6 Claude Marché, Albert Rubio, and Hans Zantema. Termination problem data base: Format of input files. https://www.lri.fr/~marche/tpdb/format.html. Accessed: 2018-17-01.

7 Julian Nagele. *Mechanizing Confluence*. PhD thesis, University of Innsbruck, 2017.

8 Julian Nagele and Aart Middeldorp. Certification of classical confluence results for left-linear term rewrite systems. In Jasmin Christian Blanchette and Stephan Merz, editors, *Interactive Theorem Proving - 7th International Conference, ITP 2016, Nancy, France, August 22-25, 2016, Proceedings*, volume 9807 of *Lecture Notes in Computer Science*, pages 290–306. Springer, 2016. doi:10.1007/978-3-319-43144-4_18.

9 Terese, editor. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

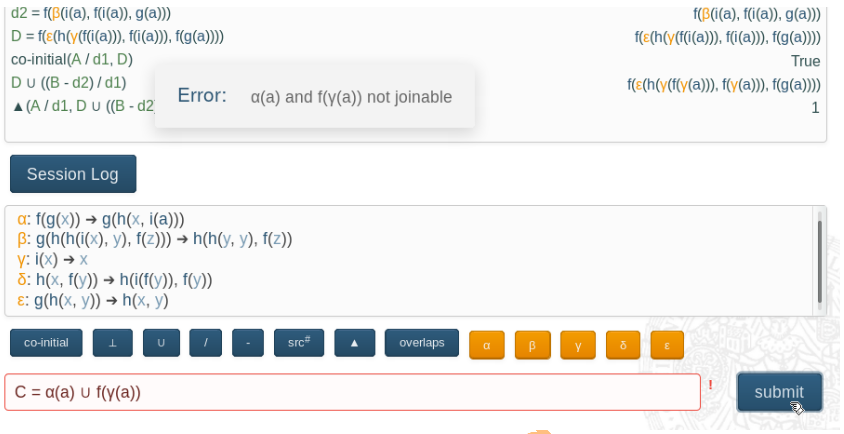
10 Vaadin framework 8. https://vaadin.com/docs/v8/framework/introduction/intro-overview.html. Accessed: 2018-17-01.

11 Vincent van Oostrom. Developing developments. *Theor. Comput. Sci.*, 175(1):159–181, 1997. doi:10.1016/S0304-3975(96)00173-9.

12 Vincent van Oostrom and Roel C. de Vrijer. Four equivalent equivalences of reductions. *Electr. Notes Theor. Comput. Sci.*, 70(6):21–61, 2002. doi:10.1016/S1571-0661(04)80599-1.

A Additional Screenshots

This appendix contains the screenshots referred to in Sections 3.2 and 3.3.



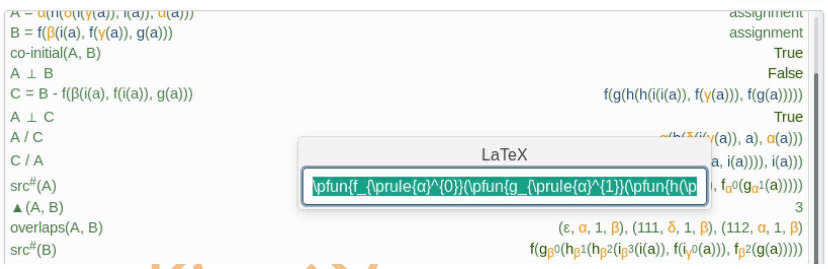
```
d2 = f(β(i(a), f(i(a)), g(a)))                    f(β(i(a), f(i(a)), g(a)))
D = f(ε(h(γ(f(i(a))), f(i(a))), f(g(a))))   f(ε(h(γ(f(i(a))), f(i(a))), f(g(a))))
co-initial(A / d1, D)                                                  True
D ∪ ((B - d2) / d1)                     f(ε(h(γ(f(γ(a))), f(γ(a))), f(g(a))))
▲(A / d1, D ∪ ((B - d2                                                    1

Error:   α(a) and f(γ(a)) not joinable
```

```
α: f(g(x)) → g(h(x, i(a)))
β: g(h(h(i(x), y), f(z))) → h(h(y, y), f(z))
γ: i(x) → x
δ: h(x, f(y)) → h(i(f(y)), f(y))
ε: g(h(x, y)) → h(x, y)
```

```
C = α(a) ∪ f(γ(a))                                   !    submit
```

Figure 2 An invalid assignment; the join operation of these two proof terms is not defined.



```
A = α(h(δ(i(γ(a)), i(a)), α(a)))                                  assignment
B = f(β(i(a), f(γ(a)), g(a)))                                     assignment
co-initial(A, B)                                                        True
A ⊥ B                                                                  False
C = B - f(β(i(a), f(i(a)), g(a)))       f(g(h(h(i(i(a)), f(γ(a))), f(g(a)))))
A ⊥ C                                                                   True
A / C                                                 (a)), a), α(a)))
C / A                                                 a, i(a)))), i(a)))
src#(A)                                               , fα0(gα1(a)))))
▲(A, B)                                                                    3
overlaps(A, B)             (ε, α, 1, β), (111, δ, 1, β), (112, α, 1, β)
src#(B)        f(gβ0(hβ1(hβ2(iβ3(i(a)), f(iγ0(a))), fβ2(g(a)))))

LaTeX
\pfun{f_{\prule{α}^{0}}(\pfun{g_{\prule{α}^{1}}(\pfun{h(\p
```

Figure 3 Popup view containing the LaTeX representation of a labeled proof term.