CSI – A Confluence Tool*

Harald Zankl, Bertram Felgenhauer, and Aart Middeldorp

Institute of Computer Science, University of Innsbruck, 6020 Innsbruck, Austria

Abstract. This paper describes a new confluence tool for term rewrite systems. Due to its modular design, the few techniques implemented so far can be combined flexibly. Methods developed for termination analysis are adapted to prove and disprove confluence. Preliminary experimental results show the potential of our tool.

Keywords: term rewriting, confluence, automation.

1 Introduction

We describe a new automatic tool for (dis)proving confluence of first-order rewrite systems (TRSs for short). Our tool is developed in Innsbruck, the city at the *c*onfluence of the two rivers Sill and Inn, and abbreviated CSI. It is available from

`http://cl-informatik.uibk.ac.at/software/csi`

and supports two new techniques for disproving confluence and very few but recent techniques for establishing confluence. CSI is open-source, equipped with a strategy language, and accessible via a simple web interface.

We assume familiarity with term rewriting and confluence [3, 15]. The remainder of this paper is organized as follows. In Section 2 the main techniques supported by CSI are summarized. Implementation issues are addressed in Section 3 and Section 4 concludes with preliminary experimental results.

2 Techniques

Besides Knuth and Bendix' criterion [9] (joinability of critical pairs for terminating systems), CSI supports the techniques described below.

Non-Confluence To disprove confluence of a TRS $\mathcal{R}$ we consider peaks

$$t \,{}^{\leqslant m}\!\leftarrow t_1 \leftarrow s \rightarrow u_1 \rightarrow^{\leqslant n} u \tag{1}$$

such that $t_1 = s[r_1\sigma]_p \leftarrow s[\ell_1\sigma]_p = s = s[\ell_2\sigma]_q \rightarrow s[r_2\sigma]_q = u_1$ with $\ell_1 \rightarrow r_1$, $\ell_2 \rightarrow r_2 \in \mathcal{R}$, $q \leqslant p$, and $p \in \mathcal{P}\mathsf{os}(s[\ell_2]_q)$. This includes critical overlaps and some variable overlaps. In order to test non-joinability of t and u we consider ground

* This research is supported by FWF (Austrian Science Fund) project P22467.

N. Bjørner and V. Sofronie-Stokkermans (Eds.): CADE 2011, LNAI 6803, pp. 499–505, 2011.

instances of t and u. Let c_x be a fresh constant for every variable x and let $\hat{t}$ denote the result of replacing every variable in a term t by the corresponding constant. Since for terms s and w we have $s \to_{\mathcal{R}} w$ if and only if $\hat{s} \to_{\mathcal{R}} \hat{w}$, it follows that terms t and u are joinable if and only if $\hat{t}$ and $\hat{u}$ are joinable. In order to test non-joinability of $\hat{t}$ and $\hat{u}$ we overapproximate the sets of reducts for $\hat{t}$ and $\hat{u}$ and check if the intersection is empty.

The first approach is based on TCAP, which was introduced to obtain a better approximation of dependency graphs [7]. Let t be a term. The term $\mathrm{TCAP}(t)$ is inductively defined as follows. If t is a variable, $\mathrm{TCAP}(t)$ is a fresh variable. If $t = f(t_1, \ldots, t_n)$ then we let $u = f(\mathrm{TCAP}(t_1), \ldots, \mathrm{TCAP}(t_n))$ and define $\mathrm{TCAP}(t)$ to be u if u does not unify with the left-hand side of a rule in $\mathcal{R}$, and a fresh variable otherwise.

Lemma 1. *If $\hat{t}$ and $\hat{u}$ are joinable then $\mathrm{TCAP}(\hat{t})$ and $\mathrm{TCAP}(\hat{u})$ unify.* □

In the sequel we use the result in its contrapositive form, i.e., whenever $\mathrm{TCAP}(\hat{t})$ and $\mathrm{TCAP}(\hat{u})$ are not unifiable then $\hat{t}$ and $\hat{u}$ are not joinable.

The following example motivates why replacing variables by constants is beneficial.

Example 2. Consider the TRS $\mathcal{R}$ consisting of the rules $\mathsf{f}(x, y) \to \mathsf{g}(x)$ and $\mathsf{f}(x, y) \to \mathsf{g}(y)$. Note that $\mathrm{TCAP}(\mathsf{g}(x)) = \mathsf{g}(x')$ and $\mathrm{TCAP}(\mathsf{g}(y)) = \mathsf{g}(y')$ are unifiable but since x and y are different normal forms it is beneficial to replace them by fresh constants such that unification fails. We have $\mathrm{TCAP}(\mathsf{g}(\mathsf{c}_x)) = \mathsf{g}(\mathsf{c}_x)$ is not unifiable with $\mathsf{g}(\mathsf{c}_y) = \mathrm{TCAP}(\mathsf{g}(\mathsf{c}_y))$.

The next example illustrates Lemma 1.

Example 3. Consider the TRS $\mathcal{R} = \{\mathsf{a} \to \mathsf{f}(\mathsf{a}, \mathsf{b}), \mathsf{f}(\mathsf{a}, \mathsf{b}) \to \mathsf{f}(\mathsf{b}, \mathsf{a})\}$ from [16] and the peak $\hat{t} = \mathsf{f}(\mathsf{f}(\mathsf{b}, \mathsf{a}), \mathsf{b}) \;{}^{2}\!\leftarrow \mathsf{f}(\mathsf{a}, \mathsf{b}) \to \mathsf{f}(\mathsf{b}, \mathsf{a}) = \hat{u}$. Since $\mathrm{TCAP}(\hat{t}) = \mathsf{f}(\mathsf{f}(\mathsf{b}, x), \mathsf{b})$ and $\mathrm{TCAP}(\hat{u}) = \mathsf{f}(\mathsf{b}, y)$ are not unifiable $\mathcal{R}$ is not confluent.

We remark that Lemma 1 subsumes the case that t and u are different normal forms or that t and u have different root symbols which do not occur at the root of any left-hand side in $\mathcal{R}$. The latter amounts to $t(\epsilon) \neq u(\epsilon)$ and $t(\epsilon) \neq \ell(\epsilon) \neq u(\epsilon)$ for all $\ell \to r \in \mathcal{R}$, which is the test performed in [2].

Our second approach is based on tree automata. Let $\mathcal{R}$ be a left-linear TRS and L a set of ground terms. A tree automaton $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ is *compatible* [6] with $\mathcal{R}$ and L if $L \subseteq \mathcal{L}(\mathcal{A})$ and for each $\ell \to r \in \mathcal{R}$ and state substitution $\sigma \colon \mathcal{V}\mathsf{ar}(\ell) \to Q$, $r\sigma \to^*_\Delta q$ whenever $\ell\sigma \to^*_\Delta q$. The extension to arbitrary TRSs that we use in our implementation is described in [10]. Here $\mathcal{L}(\mathcal{A})$ is the language accepted by a tree automaton $\mathcal{A}$. In the following $\to^*_{\mathcal{R}}(L)$ denotes the set $\{t \mid s \to^*_{\mathcal{R}} t \text{ for some } s \in L\}$.

Theorem 4. *Let $\mathcal{R}$ be a TRS, $\mathcal{A}$ a tree automaton, and L a set of ground terms. If $\mathcal{A}$ is compatible with $\mathcal{R}$ and L then $\to^*_{\mathcal{R}}(L) \subseteq \mathcal{L}(\mathcal{A})$.* □

We overapproximate the sets of terms reachable from $\hat{t}$ and $\hat{u}$ using tree automata, i.e., we construct tree automata $\mathcal{A}_1$ and $\mathcal{A}_2$ (by *tree automata completion* [10]) such that $\to^*_{\mathcal{R}}(\{\hat{t}\}) \subseteq \mathcal{L}(\mathcal{A}_1)$ and $\to^*_{\mathcal{R}}(\{\hat{u}\}) \subseteq \mathcal{L}(\mathcal{A}_2)$ and conclude non-joinability of $\hat{t}$ and $\hat{u}$ if $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2) = \varnothing$, which is decidable.

Example 5. Consider Lévy's TRS $\mathcal{R}$ from [8]

$$\begin{array}{llll} \mathsf{f}(\mathsf{a},\mathsf{a}) \to \mathsf{g}(\mathsf{b},\mathsf{b}) & \mathsf{a} \to \mathsf{a}' & \mathsf{f}(\mathsf{a}',x) \to \mathsf{f}(x,x) & \mathsf{f}(x,\mathsf{a}') \to \mathsf{f}(x,x) \\ \mathsf{g}(\mathsf{b},\mathsf{b}) \to \mathsf{f}(\mathsf{a},\mathsf{a}) & \mathsf{b} \to \mathsf{b}' & \mathsf{g}(\mathsf{b}',x) \to \mathsf{g}(x,x) & \mathsf{g}(x,\mathsf{b}') \to \mathsf{g}(x,x) \end{array}$$

and $\hat{t} = \mathsf{f}(\mathsf{a}',\mathsf{a}') \;{}^*\!\leftarrow \mathsf{f}(\mathsf{a},\mathsf{a}) \to^* \mathsf{g}(\mathsf{b}',\mathsf{b}') = \hat{u}$. We have $\to^*_{\mathcal{R}}(\{\hat{t}\}) = \{\hat{t}\}$ and $\to^*_{\mathcal{R}}(\{\hat{u}\}) = \{\hat{u}\}$. Consequently $\to^*_{\mathcal{R}}(\{\hat{t}\}) \cap \to^*_{\mathcal{R}}(\{\hat{u}\}) = \varnothing$ and hence we conclude non-joinability of $\hat{t}$ and $\hat{u}$ which yields the non-confluence of $\mathcal{R}$. Note that $\mathrm{TCAP}(\hat{t}) = x$ and $\mathrm{TCAP}(\hat{u}) = y$ unify.

Order-Sorted Decomposition. Next we focus on a criterion that allows to decompose a TRS $\mathcal{R}$ into TRSs $\mathcal{R}_1 \cup \cdots \cup \mathcal{R}_n$ where $\mathcal{R}$ is confluent whenever all $\mathcal{R}_i$ are confluent. Order-sorted decomposition is a generalization of persistent decomposition [2, Definition 2] to ordered sorts. It is based on a result in [5].

Theorem 6. *Let $\mathcal{R}$ be a TRS and $\langle \mathcal{F}, \mathcal{V} \rangle$ an order-sorted signature with sorts $\mathcal{S}$ equipped with a strict order $\succ$. Assume that the following conditions hold:*

1. *$\mathcal{R}$ is compatible with $\mathcal{S}$, i.e., rules $\ell \to r \in \mathcal{R}$ are well-sorted, with variables bound strictly in ℓ and the sort of ℓ is $\succeq$ that of r.*
2. *If $\mathcal{R}$ is non-left-linear and duplicating then for $\ell \to r \in \mathcal{R}$, variables in r are bound strictly as well. Furthermore, if $r \in \mathcal{V}$ the sort of r must be maximal.*

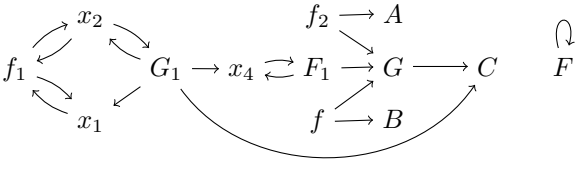
If $\mathcal{R}$ is confluent on well-sorted terms then $\mathcal{R}$ is confluent on all terms. □

Each sort attachment satisfying the conditions of Theorem 6 gives rise to a decomposition of $\mathcal{R}$ into $\max\{\mathcal{R} \cap \mathcal{T}_{\trianglelefteq\alpha}(\mathcal{F},\mathcal{V}) \times \mathcal{T}_{\trianglelefteq\alpha}(\mathcal{F},\mathcal{V}) \mid \alpha \in \mathcal{S}\}$, where $\mathcal{T}_{\trianglelefteq\alpha}(\mathcal{F},\mathcal{V})$ denotes the subterms of terms of sort $\preceq \alpha$. Note that we can replace proper subterms $t|_p : \beta$ by any other terms with sort $\preceq \beta$. Hence $\mathcal{T}_{\trianglelefteq\alpha}(\mathcal{F},\mathcal{V})$ is closed under adding terms of sort $\preceq$ that of any terms in $\mathcal{T}_{\trianglelefteq\alpha}(\mathcal{F},\mathcal{V})$. As in the many-sorted persistence case, we can find a most general ordered sort attachment consistent with any given TRS efficiently. Start by assigning sort variables to the argument and result types of all function symbols and to the variables occurring in the rules, after renaming them to ensure that no two rules share any variables. The consistency conditions, except for the maximality condition for collapsing rules, translate to inequalities $\alpha \succeq \beta$ between these type variables. To solve a system of such constraints, consider the graph with sort variables as nodes and edges from α to β whenever there is a constraint $\alpha \succeq \beta$. Then assign a distinct sort to the variables of each strongly connected component of the graph, ordered strictly by the edges between the components. A maximality constraint on β can be enforced in a second pass that equates α and β whenever $\alpha \succ \beta$. This process is demonstrated in the example below.

Example 7. Consider the TRS

$$1\colon \mathsf{f}(x,\mathsf{A}) \to \mathsf{G}(x) \quad 2\colon \mathsf{f}(x,\mathsf{G}(x)) \to \mathsf{B} \quad 3\colon \mathsf{G}(\mathsf{C}) \to \mathsf{C} \quad 4\colon \mathsf{F}(x) \to \mathsf{F}(\mathsf{G}(x))$$

We start by assigning variables to the various sorts. Let x_i be the sort of x in rule i. Furthermore let $\mathsf{A} : A$, $\mathsf{B} : B$, $\mathsf{C} : C$, $\mathsf{f} : f_1 \times f_2 \to f$, $\mathsf{F} : F_1 \to F$ and $\mathsf{G} : G_1 \to G$. By well-sortedness we get constraints $f_1 \succeq x_1$, $f_2 \succeq A$ from the left-hand side of the first rule. By strictness of left-hand sides, we require that $x_1 \succeq f_1$. We get similar constraints from the other rules, noting that since the TRS is non-duplicating, we do not have strictness constraints on the right-hand sides. By relating the sorts of left-hand sides and right-hand sides, we obtain further constraints, namely $f \succeq G$, $f \succeq B$, $G \succeq C$ and $F \succeq F$. Denoting $\alpha \succeq \beta$ by an edge $\alpha \to \beta$, we obtain the following graph:



The strongly connected components are $8 = \{f_2\}$, $7 = \{A\}$, $6 = \{f\}$, $5 = \{B\}$, $4 = \{G_1, x_1, f_1, x_2\}$, $3 = \{F_1, x_4\}$, $2 = \{G\}$, $1 = \{C\}$, and $0 = \{F\}$, ordered by $8 \succ 7, 2$, $6 \succ 5, 2$, and $4 \succ 3 \succ 2 \succ 1$. The resulting signature is $\mathsf{A} : 7$, $\mathsf{B} : 5$, $\mathsf{C} : 1$, $\mathsf{f} : 4 \times 8 \to 6$, $\mathsf{F} : 3 \to 0$, and $\mathsf{G} : 4 \to 2$ giving rise to the decomposition into the TRSs $\{(1), (2), (3)\}$ and $\{(3), (4)\}$.

If we required maximality of the sort $2 = \{G\}$, then we would equate 2 and 3 (since $3 \succ 2$), and further with 4 (as $4 \succ 3$), 6 (as $6 \succ 2$) and 8 (as $8 \succ 2$), obtaining $8' = \{G, F_1, x_4, G_1, x_1, f_1, x_2, f, f_2\}$, ordered by $8' \succ 7, 5, 1$. The resulting signature is $\mathsf{A} : 7$, $\mathsf{B} : 5$, $\mathsf{C} : 1$, $\mathsf{f} : 8' \times 8' \to 8'$, $\mathsf{F} : 8' \to 0$, and $\mathsf{G} : 8' \to 8'$. Note that here no (non-trivial) decomposition is possible.

Decreasing Diagrams. The decreasing diagrams technique [12, 14] is a complete method for confluence on countable abstract rewrite systems. The next result employs decreasing diagrams for TRSs and follows immediately from [17, Corollary 3.16]. It also serves to demonstrate the design of our tool which typically implements one criterion by combining smaller pieces via a strategy language (cf. Section 3). Here $\mathcal{R}_{\mathsf{d}}$ ($\mathcal{R}_{\mathsf{nd}}$) denotes the (non)duplicating rules in a TRS $\mathcal{R}$.

Theorem 8. *A left-linear TRS $\mathcal{R}$ is confluent if $\mathcal{R}_{\mathsf{d}}$ is terminating relative to $\mathcal{R}_{\mathsf{nd}}$ and all critical peaks of $\mathcal{R}$ are decreasing with respect to the rule labeling.* □

To exploit this theorem we need to solve relative termination problems. In [17] we show that relative termination techniques can additionally be used for labeling diagrams (also in combination with the rule labeling).

3 Implementation

CSI is implemented based on the open source termination tool $\mathsf{T_TT_2}$ [11] and written in OCaml. As explained in the preceding section, several criteria from termination analysis are useful for confluence. Our tool is based on few techniques, but a *strategy language* (akin to the one to control $\mathsf{T_TT_2}$) allows one to combine different criteria flexibly and to obtain a powerful tool. For a grammar of this strategy language, consult [11] or pass the option `-h` to the tool.

Automatic Mode. In its automatic mode CSI executes the strategy

```
(KB || NOTCR || (((CLOSED || DD) | add)2*)! || sorted -order)*
```

Here identifiers in capital letters abbreviate combinations of techniques. We skip details for brevity. The command `KB` refers to Knuth and Bendix' criterion [9], `NOTCR` is a test for non-confluence as described in Section 2, and `sorted -order` aims for an order-sorted decomposition (cf. Section 2 and [5]). The operator `||` executes all those criteria in parallel—to make use of modern multi-core architectures—and the first substrategy that succeeds is used to make progress on the given problem. Since a successful call to `sorted -order` returns a list of problems, the trailing `*` ensures that the above strategy is iterated on all subproblems until no further progress can be achieved. Finally we describe the part that is still missing. `CLOSED` tests whether the critical pairs of a left-linear system are development closed [13] and `DD` implements decreasing diagrams (Section 2 and [17]). If these methods do not succeed the alternative `|` executes `add`, which adds new rules that might enable the other criteria to succeed ([17, Lemma 4.3 and Example 4.4]) while the postfix `2*` executes the strategy inside parentheses at most two times, i.e., `CLOSED || DD` is run again, if some rules have been added. The outermost `!` ensures that the strategy inside only succeeds if confluence could be (dis)proved.

Strategy Language. We elaborate on the strategy language to show the flexibility and modularity of our tool. In the strategy `nonconfluence -steps 2 -tcap` the flag `-tcap` tests non-joinability of terms with TCAP, as outlined in Section 2. With `-steps` values for m and n in the peak (1) on page 499 are set.

The criterion from Theorem 8 allows one to use the decreasing diagrams technique, provided some precondition is satisfied. To this end the composition operator `;` is employed, where `A;B` executes `B` only if `A` succeeds. Given an input TRS $\mathcal{R}$, in the strategy

```
cr -dup;matrix -dim 2*;rule_labeling;decreasing
```

the expression `cr -dup` generates the relative TRS $\mathcal{R}_{\mathsf{d}}/\mathcal{R}_{\mathsf{nd}}$, termination of which is attempted with matrix interpretations of dimension 2. If this succeeds the critical diagrams are labeled with the rule labeling [14], before a test for decreasingness is performed. We note that the strategy language allows to label incrementally combining different (relative termination) criteria [17]. Here a critical diagram is a critical peak $t \leftarrow s \rightarrow u$ together with joining sequences $t \rightarrow^* v \ {}^*\!\leftarrow u$. In the implementation for every critical peak we consider all joining sequences $t \rightarrow^{\leqslant n} \cdot {}^{\leqslant n}\!\leftarrow u$ for which there is no smaller n that admits a common reduct.

Table 1. Experiments

(a) 106 TRSs.

	CSI	ACP	$\bigcup$
CR	61	64	67
not CR	20	18	21

(b) 99 TRSs.

	CSI	ACP	$\bigcup$
CR	43	42	43
not CR	47	47	47

(c) 9 TRSs.

	CSI	ACP	$\bigcup$
CR	6	2	6
not CR	2	2	2

Table 2. Performance difference on the three testbenches

system	CSI	ACP	status
BN98/ex6.5f	$\times(\infty)$	✓(0.4)	¬CR
Der97/p204	$\times$(6.6)	✓(0.1)	CR
GL06/ex3	✓(0.6)	$\times$(0.2)	CR
GOO96/R2p	$\times$(6.3)	✓(0.1)	CR
Gra96caap/ex2	$\times$(3.0)	✓(0.1)	CR
OO03/ex1	✓(4.0)	$\times$(7.0)	CR
OO03/ex2	$\times$(6.3)	✓(0.1)	CR
Ohl94caap/ex5.12	✓(0.4)	$\times$(0.1)	¬CR
TO01/ex6	$\times$(6.3)	✓(0.1)	CR

system	CSI	ACP	status
Tiw02/ex1	✓(0.3)	$\times$(0.1)	¬CR
Toy98/ex1	$\times$(6.1)	✓(0.1)	CR
standards/AC	✓(2.7)	$\times$(0.1)	CR
standards/add_C	✓(4.0)	$\times$(0.1)	CR
Transformed_CS[a]	✓(4.6)	$\times(\infty)$	CR
ZFM11/ex1.1	✓(4.9)	$\times$(7.0)	CR
ZFM11/ex3.18	✓(1.0)	$\times$(0.1)	CR
ZFM11/ex3.20	✓(2.1)	$\times$(0.5)	CR
ZFM11/ex4.1	✓(0.9)	$\times$(0.1)	CR

[a] Transformed_CSR_04_PALINDROME_nokinds-noand_⊥

4 Evaluation

For experiments[1] we used the collection from [1] which consists of 106 TRSs from the rewriting literature dealing with confluence (Table 1(a)), the 99 TRSs from the 2010 edition of the termination competition which are non-terminating or not known to be terminating (Table 1(b)), and the TRSs from [5,17] (Table 1(c)). The time limit of 60 seconds was hardly ever reached.

In Table 1 we compare the automatic mode of our tool with ACP [2], a confluence prover that implements various techniques from the literature. On the testbench in Table 1(a) ACP can show more systems confluent than CSI but our tool is superior for non-confluence. The last column shows that on this testbench no tool subsumes the other one which is not the case for Tables 1(b)(c).

Table 2 elaborates on the differences of the tools' performance. Here a $\times$ indicates that the corresponding tool failed to analyze the status of the given TRS while a ✓ means that confluence (or non-confluence) could be determined. The numbers in parentheses refer to the time spent on this problem in seconds. The different blocks in Table 2 correspond to the different testbeds employed.

The rewriting toolkit C*i*ME3 [4] also supports confluence analysis as one of its many features. This tool exploits Newman's Lemma, i.e., for a terminating TRS confluence coincides with local confluence (the latter can then effectively be

[1] Details are available from the CSI website.

checked [9]). While this test is also contained in ACP and CSI, the novel feature of C*i*ME3 is that it can (automatically) certify such confluence proofs in the proof assistant Coq.

To conclude we stress the main attractions of CSI: To the best of our knowledge it is the only tool that implements order-sorted decomposition of rewrite systems, it employs powerful criteria for disproving confluence, and due to the modular design it allows to combine different labeling functions for the decreasing diagrams technique.

References

1. Aoto, T.: Automated confluence proof by decreasing diagrams based on rule-labelling. In: Lynch, C. (ed.) RTA 2010. LIPIcs, vol. 6, pp. 7–16. Schloss Dagstuhl, Dagstuhl (2010)
2. Aoto, T., Yoshida, J., Toyama, Y.: Proving confluence of term rewriting systems automatically. In: Treinen, R. (ed.) RTA 2009. LNCS, vol. 5595, pp. 93–102. Springer, Heidelberg (2009)
3. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
4. Contejean, E., Courtieu, P., Forest, J., Pons, O., Urbain, X.: Automated certified proofs with CiME3. In: Schmidt-Schauß, M. (ed.) RTA 2011. LIPIcs, vol. 10, pp. 21–30. Schloss Dagstuhl, Dagstuhl (2011)
5. Felgenhauer, B., Zankl, H., Middeldorp, A.: Proving confluence with layer systems (2011); submitted for publication
6. Genet, T.: Decidable approximations of sets of descendants and sets of normal forms. In: Nipkow, T. (ed.) RTA 1998. LNCS, vol. 1379, pp. 151–165. Springer, Heidelberg (1998)
7. Giesl, J., Thiemann, R., Schneider-Kamp, P.: Proving and disproving termination of higher-order functions. In: Gramlich, B. (ed.) FroCoS 2005. LNCS (LNAI), vol. 3717, pp. 216–231. Springer, Heidelberg (2005)
8. Huet, G.: Confluent reductions: Abstract properties and applications to term rewriting systems. JACM 27(4), 797–821 (1980)
9. Knuth, D., Bendix, P.: Simple word problems in universal algebras. In: Leech, J. (ed.) Computational Problems in Abstract Algebra, pp. 263–297. Pergamon Press, Oxford (1970)
10. Korp, M., Middeldorp, A.: Match-bounds revisited. I&C 207(11), 1259–1283 (2009)
11. Korp, M., Sternagel, C., Zankl, H., Middeldorp, A.: Tyrolean termination tool 2. In: Treinen, R. (ed.) RTA 2009. LNCS, vol. 5595, pp. 295–304. Springer, Heidelberg (2009)
12. van Oostrom, V.: Confluence by decreasing diagrams. TCS 126(2), 259–280 (1994)
13. van Oostrom, V.: Developing developments. TCS 175(1), 159–181 (1997)
14. van Oostrom, V.: Confluence by decreasing diagrams. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 306–320. Springer, Heidelberg (2008)
15. Terese: Term Rewriting Systems, vol. 55. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge (2003)
16. Tiwari, A.: Deciding confluence of certain term rewriting systems in polynomial time. In: LICS 2002, pp. 447–457 (2002)
17. Zankl, H., Felgenhauer, B., Middeldorp, A.: Labelings for decreasing diagrams. In: Schmidt-Schauß, M. (ed.) RTA 2011. LIPIcs, vol. 10, pp. 377–392. Schloss Dagstuhl, Dagstuhl (2011)