CSI: New Evidence – A Progress Report

Julian Nagele(✉), Bertram Felgenhauer, and Aart Middeldorp

Department of Computer Science, University of Innsbruck, Innsbruck, Austria
{julian.nagele,bertram.felgenhauer,aart.middeldorp}@uibk.ac.at

Abstract. CSI is a strong automated confluence prover for rewrite systems which has been in development since 2010. In this paper we report on recent extensions that make CSI more powerful, secure, and useful. These extensions include improved confluence criteria but also support for uniqueness of normal forms. Most of the implemented techniques produce machine-readable proof output that can be independently verified by an external tool, thus increasing the trust in CSI. We also report on CSIˆho, a tool built on the same framework and similar ideas as CSI that automatically checks confluence of higher-order rewrite systems.

1 Introduction

CSI [44] is an automatic confluence prover for rewrite systems, which participates in the annual confluence competition (CoCo) [1].

In this paper we report on recent additions to CSI, in particular support for higher-order rewrite systems, efficient decision procedures for the unique normal form properties for ground rewrite systems, support for first-order systems with associative and commutative symbols, and more refined non-confluence techniques. Several techniques have been formalized to enable certification of the output of CSI, making it the most trustworthy confluence tool.

We assume familiarity with rewriting [7]. Here we only recall notions that will be used in Sect. 2. We consider terms built from a signature $\mathcal{F}$ and a disjoint set of variables $\mathcal{V}$. Given a subset $\mathcal{F}_{\mathsf{AC}} \subseteq \mathcal{F}$ of binary function symbols, the term rewrite system (TRS for short) AC consists of the AC rules $f(x,y) \to f(y,x)$ and $f(f(x,y),z) \to f(x,f(y,z))$ for every $f \in \mathcal{F}_{\mathsf{AC}}$. We write $\sim_{\mathsf{AC}}$ for the congruence induced by AC. Given a TRS $\mathcal{R}$ over the signature $\mathcal{F}$, we write $\mathcal{R}^e$ for the union of $\mathcal{R}$ and the extended rules $f(\ell,x) \to f(r,x)$ for all $\ell \to r \in \mathcal{R}$ such that $\mathsf{root}(\ell) = f \in \mathcal{F}_{\mathsf{AC}}$. We write $\to_{\mathcal{R}/\mathsf{AC}}$ for the relation $\sim_{\mathsf{AC}} \cdot \to_{\mathcal{R}} \cdot \sim_{\mathsf{AC}}$. The relation $\to_{\mathcal{R},\mathsf{AC}}$ is defined as follows: $s \to_{\mathcal{R},\mathsf{AC}} t$ if there exists a position p in s, a rewrite rule $\ell \to r$ in $\mathcal{R}$, and a substitution σ such that $s|p \sim_{\mathsf{AC}} \ell\sigma$ and $t = s[r\sigma]_p$. The relations $\to_{\mathcal{R}/\mathsf{AC}}$ and $\to_{\mathcal{R}^e,\mathsf{AC}} \cdot \sim_{\mathsf{AC}}$ coincide.

Consider two rewrite rules $\ell_1 \to r_1$ and $\ell_2 \to r_2$ without common variables and a function position p in ℓ_2 such that ℓ_l and $\ell_2|_p$ are unifiable modulo AC. Given a complete set S of AC unifiers of ℓ_l and $\ell_2|_p$, the pair $\ell_2[r_1]_p\sigma \approx r_2\sigma$ with $\sigma \in S$ is called an AC critical pair. The set of all AC critical pairs between rules of a TRS $\mathcal{R}$ and a TRS $\mathcal{S}$ is denoted by $\mathsf{CP}_{\mathsf{AC}}(\mathcal{R},\mathcal{S})$.

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

© Springer International Publishing AG 2017
L. de Moura (Ed.): CADE 2017, LNAI 10395, pp. 385–397, 2017.
DOI: 10.1007/978-3-319-63046-5_24

The remainder of the paper is organized as follows. In the next section we report on the main extensions to CSI for (non-)confluence proving of TRSs. Section 3 is devoted to the support of CSI for the unique normal form properties. The extension to higher-order systems is covered in Sect. 4. An overview of the certified techniques in CSI is presented in Sect. 5. Some implementation details are given in Sect. 6 before we conclude in Sect. 7 with experimental data.

2 Extensions

In this section we describe the two features for (non-)confluence proving of TRSs that were added to CSI after CoCo 2016. Other extensions are briefly described in the one-page tool descriptions accompanying CoCo.[1]

TRSs that contain AC rules pose a challenge for confluence provers. The confluence problems database (Cops)[2] contains several such systems whose status is open. Aoto and Toyama [2] developed a special confluence technique for rewrite systems with AC rules and more general non-terminating rewrite systems, which is incorporated in the confluence prover ACP [5]. A key idea in [2] is that AC rules are *reversible*. This idea was combined with the extended critical pair lemma of Jouannaud and Kirchner [13] in Saigawa [15] and more recently in CoLL [38], where the technique is extended to handle associative rules in the absence of commutation rules.

Theorem 1 (Jouannaud and Kirchner [13], Shintani and Hirokawa [38]). *If $\mathcal{R} = \mathcal{S} \uplus \mathsf{AC}$ such that $s \rightarrow^*_{\mathcal{S},\mathsf{AC}} \cdot \sim_{\mathsf{AC}} \cdot {}_{\mathcal{S},\mathsf{AC}}^{*}\!\leftarrow t$ for all $s \approx t \in \mathsf{CP}_{\mathsf{AC}}(\mathcal{S}, \mathcal{S} \cup \mathsf{AC} \cup \mathsf{AC}^{-1})$ and $\mathcal{S}/\mathsf{AC}$ is terminating then $\mathcal{R}$ is confluent.*

In CSI we incorporated the version of the AC critical pair lemma based on extended rules [32], which is used in the modern completion tool mkbtt.[3]

Theorem 2. *If $\mathcal{R} = \mathcal{S} \uplus \mathsf{AC}$ such that $s \rightarrow^*_{\mathcal{S}/\mathsf{AC}} \cdot \sim_{\mathsf{AC}} \cdot {}_{\mathcal{S}/\mathsf{AC}}^{*}\!\leftarrow t$ for all $s \approx t \in \mathsf{CP}_{\mathsf{AC}}(\mathcal{S}^e, \mathcal{S}^e)$ and $\mathcal{S}/\mathsf{AC}$ is terminating then $\mathcal{R}$ is confluent.*

We illustrate the use of Theorem 2 on two examples.

Example 3. The rewrite system (Cops 183)

$$\begin{array}{rrr}
+(0,x) \to x & +(x,0) \to x & -(+(x,y)) \to +(-(x),-(y)) \\
+(1,-(1)) \to 0 & +(-(1),1) \to 0 & +(x,y) \to +(y,x) \\
-(0) \to 0 & -(-(x)) \to x & +(+(x,y),z) \to +(x,+(y,z))
\end{array}$$

cannot be handled by the recent *ground* confluence prover AGCP [3, Example 25]. After removing the AC rules $+(+(x,y),z) \to +(x,+(y,z))$ and $+(x,y) \to$

[1] Available from http://coco.nue.riec.tohoku.ac.jp/2013, http://coco.nue.riec.tohoku.ac.jp/2014, http://coco.nue.riec.tohoku.ac.jp/2015, http://coco.nue.riec.tohoku.ac.jp/2016, under Entrants.

[2] http://cops.uibk.ac.at.

[3] http://cl-informatik.uibk.ac.at/software/mkbtt.

$+(y,x)$ we obtain a rewrite system $\mathcal{S}$ such that the 18 AC critical pairs of $\mathcal{S}^e$ are joinable modulo AC[4] (arising from 27 AC critical peaks). Since $\mathcal{S}$ is easily shown to be AC terminating (e.g. ACRPO [36] applies), we conclude by Theorem 2 that the original rewrite system is confluent. In particular, it is ground confluent, answering the open problem in [3].

Interestingly, the CoCo 2015 version of CoLL-Saigawa, using Theorem 1, could already show confluence of the TRS of Example 3. In light of the next example, it remains to be seen whether this answer can be trusted.

Example 4. Consider the rewrite system

$$\mathsf{a}+\mathsf{b} \to \mathsf{b} \qquad \mathsf{c}+\mathsf{a} \to \mathsf{a} \qquad x+y \to y+x \qquad (x+y)+z \to x+(y+z)$$

consisting of $\mathcal{S} = \{\mathsf{a}+\mathsf{b} \to \mathsf{b}, \mathsf{c}+\mathsf{a} \to \mathsf{a}\}$ and the AC rules for $+$. The two extended rules

$$(\mathsf{a}+\mathsf{b})+z \to \mathsf{b}+z \qquad\qquad (\mathsf{c}+\mathsf{a})+z \to \mathsf{a}+z$$

admit the AC critical peak

$$\mathsf{c}+\mathsf{b} \leftarrow (\mathsf{c}+\mathsf{a})+\mathsf{b} \to \mathsf{a}+\mathsf{b}$$

where $\mathsf{c}+\mathsf{b}$ is in normal form and $\mathsf{a}+\mathsf{b}$ rewrites in one step to the normal form b. These normal forms are obviously not AC equivalent. Hence Theorem 2 does not apply and CSI correctly reports that the system is not confluent. Surprisingly, CoLL-Saigawa wrongly reports the opposite. The reason could be that the peak

$$\mathsf{b}+z \;{}_{\mathcal{S}}\!\leftarrow (\mathsf{a}+\mathsf{b})+z \to_{\mathsf{AC}} \mathsf{a}+(\mathsf{b}+z)$$

is joinable modulo AC (as $\mathsf{a}+(\mathsf{b}+z) \to_{\mathcal{S}/\mathsf{AC}} \mathsf{b}+z$) but $\mathsf{a}+(\mathsf{b}+z)$ is a normal form with respect to $\to_{\mathcal{S},\mathsf{AC}}$ and hence Theorem 1 does not apply.

The second extension we describe is a non-confluence technique, or more precisely, a technique for finding non-joinable conversions. Let us first consider an example.

Example 5. Consider the TRS $\mathcal{R}$ due to Klop [16] consisting of the three rules

$$\mathsf{f}(x,x) \to \mathsf{a} \qquad\qquad \mathsf{g}(x) \to \mathsf{f}(x,\mathsf{g}(x)) \qquad\qquad \mathsf{c} \to \mathsf{g}(\mathsf{c})$$

Because of the rewrite sequence $\mathsf{c} \to \mathsf{g}(\mathsf{c}) \to \mathsf{f}(\mathsf{c},\mathsf{g}(\mathsf{c})) \to \mathsf{f}(\mathsf{g}(\mathsf{c}),\mathsf{g}(\mathsf{c})) \to \mathsf{a}$ we also have $\mathsf{c} \to \mathsf{g}(\mathsf{c}) \to^* \mathsf{g}(\mathsf{a})$ and since a and $\mathsf{g}(\mathsf{a})$ are not joinable,[5] $\mathcal{R}$ is not confluent.

[4] In fact, CSI uses a modified definition of $\mathcal{S}^e$ that avoids adding extended rules for rules where the same linear variable appears as an argument of the two top-flattenings of the left-hand and right-hand sides of the rule, using the same AC symbol. In the example, this applies to the first two rules. We still have $\to_{\mathcal{S}/\mathsf{AC}} = \to_{\mathcal{S}^e,\mathsf{AC}} \cdot \sim_{\mathsf{AC}}$.

[5] This can be shown using tree automata techniques [11].

However, finding the above conversion is non-trivial and indeed none of the participants of CoCo 2016 can show non-confluence of this system. The technique of redundant rules [22] can be used to strengthen other criteria in such situations. The basic idea is to add or remove rules that can be simulated by the other rules, thus reflecting (non-)confluence. A systematic method for finding redundant rules is presented below.

Definition 6. *Given two variable disjoint rewrite rules $\ell_1 \to r_1$ and $\ell_2 \to r_2$ of a TRS $\mathcal{R}$, a function position p in r_1, and an mgu σ of $r_1|_p$ and ℓ_2, the rewrite rule $\ell_1\sigma \to r_1\sigma[r_2\sigma]_p$ is a* forward closure *of $\mathcal{R}$. We write $\mathsf{FC}(\mathcal{R})$ for the extension of $\mathcal{R}$ with all its forward closures.*

Since the rules in $\mathsf{FC}(\mathcal{R}) \setminus \mathcal{R}$ are redundant (as they can be simulated using the rules of the original TRS $\mathcal{R}$), the following result is obvious.

Lemma 7. *A TRS $\mathcal{R}$ is confluent if and only if the TRS $\mathsf{FC}(\mathcal{R}) \cup \mathsf{FC}(\mathcal{R}^{-1})^{-1}$ is confluent* □

The reason for including $\mathsf{FC}(\mathcal{R}^{-1})^{-1}$ will become clear in the next section. Returning to Example 5 we find

$$\begin{aligned} &\mathsf{c} \to \mathsf{f}(\mathsf{c},\mathsf{g}(\mathsf{c})) \in \mathsf{FC}(\mathcal{R}) && \mathsf{c} \to \mathsf{a} \in \mathsf{FC}^3(\mathcal{R}) \\ &\mathsf{c} \to \mathsf{f}(\mathsf{g}(\mathsf{c}),\mathsf{g}(\mathsf{c})) \in \mathsf{FC}^2(\mathcal{R}) && \mathsf{c} \to \mathsf{g}(\mathsf{a}) \in \mathsf{FC}^4(\mathcal{R}) \end{aligned}$$

and hence we obtain the non-joinable critical pair $\mathsf{a} \approx \mathsf{g}(\mathsf{a}) \in \mathsf{CP}(\mathsf{FC}^4(\mathcal{R}))$. By Lemma 7 this implies non-confluence of $\mathcal{R}$.

3 Unique Normal Forms

In addition to confluence, CSI includes preliminary support for two unique normal form properties, namely UNC (any two convertible normal forms are equal; two terms s and t are convertible if $s \leftrightarrow^* t$) and UNR (any term reaches at most one normal form). Note that in contrast to CSIˆho (described in the next section), which has to deal with a whole new rewriting mechanism, adding support for UNR and UNC is similar to adding a new confluence criterion, which is why there is no separate CSIˆun tool. Furthermore, the implications

$$\mathsf{CR} \implies \mathsf{UNC} \implies \mathsf{UNR} \tag{1}$$

mean that any confluence criterion can also serve as a criterion for UNC and UNR, which is another reason for using a confluence tool as the basis of a tool for unique normal forms. (The reverse implications do not hold, as witnessed by the well-known TRSs $\mathcal{R}_1 = \{\mathsf{b} \to \mathsf{a}, \mathsf{b} \to \mathsf{c}, \mathsf{c} \to \mathsf{c}\}$ and $\mathcal{R}_2 = \mathcal{R}_1 \cup \{\mathsf{d} \to \mathsf{c}, \mathsf{d} \to \mathsf{e}\}$. The first one is UNC but not CR because a and c do not have a common reduct; the second one is UNR but not UNC because the normal forms a and e are convertible.)

CSI incorporates efficient decision procedures for both UNC and UNR for *ground* term rewrite systems, which are TRSs without variables. The former property can be decided in $O(n \log n)$ time, where n is the size of the input TRS, based on currying, the congruence closure algorithm by Nelson and Oppen [26], and an ad hoc enumeration of runs of a tree automaton that accepts convertible normal forms. The latter property is decided in $O(n^3)$ time, using currying and ground tree transducers that accept normal forms which are related by a peak. For details of these two algorithms, see [9]. Furthermore, CSI implements the following criterion for UNC for non-ground systems.

Theorem 8 (Kahrs and Smith [14]). *Every non-ω-overlapping TRS has unique normal forms with respect to conversions* (UNC)*.*

A TRS is *ω-overlapping* if it has overlaps that may be infinite terms. In order to check for ω-overlaps, CSI implements a unification algorithm without occurs-check.

Example 9. The TRS consisting of the rules

$$\mathsf{f}(x,x) \to \mathsf{a} \qquad \mathsf{f}(x,\mathsf{g}(x)) \to \mathsf{b} \qquad \mathsf{c} \to \mathsf{g}(\mathsf{c})$$

of [12] is not UNR because $\mathsf{a} \leftarrow \mathsf{f}(\mathsf{c},\mathsf{c}) \to \mathsf{f}(\mathsf{c},\mathsf{g}(\mathsf{c})) \to \mathsf{b}$ is a peak connecting two distinct normal forms. This TRS is non-overlapping but ω-overlapping because $\mathsf{f}(\mathsf{g}^\omega,\mathsf{g}^\omega)$ is an instance of both $\mathsf{f}(x,x)$ and $\mathsf{f}(y,\mathsf{g}(y))$ by substituting $\{x \mapsto \mathsf{g}^\omega, y \mapsto \mathsf{g}^\omega\}$. The TRS from Example 5 on the other hand is non-ω-overlapping and hence UNC by Theorem 8.

Finally, there is a simple check for non-UNR, where CSI attempts to find two distinct normal forms reachable from the same term by starting from critical peaks and overlaps at variables. Including variable overlaps enables CSI to find the peak in Example 9. Note that Lemma 7 also holds for UNR. This enables an alternative approach to finding a suitable peak for Example 9: There is an overlap between $\mathsf{f}(\mathsf{c},\mathsf{c}) \to \mathsf{b} \in \mathsf{FC}(\mathcal{R}^{-1})^{-1}$ and $\mathsf{f}(x,x) \to \mathsf{a} \in \mathcal{R}$, resulting in the critical pair $\mathsf{a} \approx \mathsf{b}$. Note that considering $\mathsf{FC}(\mathcal{R})$ alone does not yield any progress in this example.

We aim for having a single tool that simultaneously attempts to prove and disprove all three properties UNR, UNC and CR, fully exploiting the chain of implications (1) for optimization. For example, if UNC has been established, any effort spent on proving UNR would be wasted, but the current implementation cannot use this information. For the time being, however, there are separate tool invocations for each of these properties.

4 Higher-Order Confluence

CSI^ho is an extension of CSI for proving confluence of higher-order rewrite systems. Higher-order rewriting combines first-order rewriting with notions and concepts from (typed) λ-calculus, resulting in rewriting systems with higher-order functions and bound variables. More precisely we consider pattern rewrite systems (PRSs) as introduced by Nipkow [20,27], i.e., terms are simply typed lambda terms with constants modulo $\lambda\beta\eta$ and rewriting uses higher-order matching. Additionally left-hand sides of rewrite rules are required to be patterns [21].[6] This restriction is essential for obtaining decidability of unification and thus makes it possible to compute critical pairs. To this end CSIˆho implements a version of Nipkow's algorithm for higher-order pattern unification [28].

Example 10. The untyped lambda calculus with β and η-reduction can be encoded as a PRS as follows:

$$\mathsf{abs}\colon (\mathsf{term} \to \mathsf{term}) \to \mathsf{term} \qquad \mathsf{app}\colon \mathsf{term} \to \mathsf{term} \to \mathsf{term}$$
$$\mathsf{app}(\mathsf{abs}(\lambda x.\, M(x)), N) \to M(N) \qquad \mathsf{abs}(\lambda x.\, \mathsf{app}(M, x)) \to M$$

Next we briefly explain the confluence criteria supported by CSIˆho. The first criterion is based on a higher-order version of the critical pair lemma.

Lemma 11 (Nipkow [27]). *A PRS $\mathcal{R}$ is locally confluent if and only if $s \downarrow t$ for all critical pairs $s \approx t$ of $\mathcal{R}$.*

The definition of critical pairs is essentially the same as in the first-order setting, with some additional technicalities to account for the presence of bound variables, see e.g. [20] for a formal definition. Together with Newman's Lemma this yields a confluence criterion for PRSs.

Corollary 12. *A terminating PRS $\mathcal{R}$ is confluent if and only if $s \downarrow t$ for all critical pairs $s \approx t$ of $\mathcal{R}$.*

For showing termination CSIˆho uses a basic higher-order recursive path ordering [33] and static dependency pairs with dependency graph decomposition and the subterm criterion [19]. Alternatively, one can also use an external termination tool like WANDA [18] as an oracle.

For potentially non-terminating systems CSIˆho supports two more classical criteria based on critical pairs. The first states that *weakly orthogonal* systems are confluent.

Theorem 13 (van Oostrom and van Raamsdonk [30]). *A left-linear PRS $\mathcal{R}$ is confluent if $s = t$ for all critical pairs $s \approx t$ of $\mathcal{R}$.*

The PRS from Example 10 has two trivial critical pairs and hence is confluent. This result was extended by van Oostrom to allow for non-trivial critical pairs that are connected by a development step.[7]

Theorem 14 (van Oostrom [29]). *A left-linear PRS $\mathcal{R}$ is confluent if $s \mathrel{\circ\!\!\!\to} t$ for all critical pairs $s \approx t$ of $\mathcal{R}$.*

[6] A term is a pattern if free variables only have distinct bound variables as arguments.

[7] A development step $\mathrel{\circ\!\!\!\to}$ contracts multiple, non-overlapping but possibly nested redexes at once.

As a divide-and-conquer technique CSI^ho implements modularity, i.e., decomposing a PRS into parts with disjoint signatures, for left-linear PRSs [6]. Note that the restriction to left-linear systems is essential—unlike for the first-order setting confluence is not modular in general. The following example illustrates the problem.

Example 15. Consider the PRS $\mathcal{R}$ from [6] consisting of the three rules

$$\mathsf{f}(x,x) \to \mathsf{a} \qquad \mathsf{f}(x,\mathsf{g}(x)) \to \mathsf{b} \qquad \mu(\lambda x.\, Z(x)) \to Z(\mu(\lambda x.\, Z(x)))$$

The first two rules and the third rule on their own are confluent, e.g. by Corollary 12 and Theorem 13 respectively. However, because of the peak

$$\mathsf{a} \leftarrow \mathsf{f}(\mu(\lambda x.\, \mathsf{g}(x)), \mu(\lambda x.\, \mathsf{g}(x))) \to \mathsf{f}(\mu(\lambda x.\, \mathsf{g}(x)), \mathsf{g}(\mu(\lambda x.\, \mathsf{g}(x)))) \to \mathsf{b}$$

$\mathcal{R}$ is not confluent. Note that $\mathcal{R}$ does not have critical pairs, making it non-trivial to find this peak.

As described in Sect. 2 redundant rules can used to find such peaks. Implementing transformations based on redundant rules for PRSs is straightforward, one just has to take care to only add rules that do not violate the pattern restriction.

Example 16. Consider the PRS from Example 15. After adding the redundant rule $\mathsf{f}(\mu(\lambda x.\, \mathsf{g}(x)), \mu(\lambda x.\, \mathsf{g}(x))) \to \mathsf{b}$ there is a critical pair $\mathsf{a} \approx \mathsf{b}$ and non-confluence is obvious.

To find new rules like the one above we again use narrowing, applying rules in both directions. In Example 16 unifying $Z(\mu(\lambda x.\, Z(x)))$ with $\mathsf{g}(x)$ and applying the reversed third rule to the left-hand side of the second rule yields the desired new rule. The following example illustrates removal of redundant rules.

Example 17. Consider the following encoding of lambda-calculus with Regnier's σ-reduction [34]:

$$\begin{aligned}
\mathsf{app}(\mathsf{abs}(\lambda x.\, T(x)), S) &\to T(S) \\
\mathsf{app}(\mathsf{abs}(\lambda y.\, \mathsf{abs}(\lambda x.\, M(y,x))), S) &\to \mathsf{abs}(\lambda x.\, \mathsf{app}(\mathsf{abs}(\lambda y.\, M(y,x)), S)) \\
\mathsf{app}(\mathsf{app}(\mathsf{abs}(\lambda x.\, T(x)), S), U) &\to \mathsf{app}(\mathsf{abs}(\lambda x.\, \mathsf{app}(T(x), U)), S)
\end{aligned}$$

Since the left- and right-hand side of the second and third rule are convertible using the first rule, they can be removed and confluence of the first rule alone can be established by Theorem 13.

5 Certification

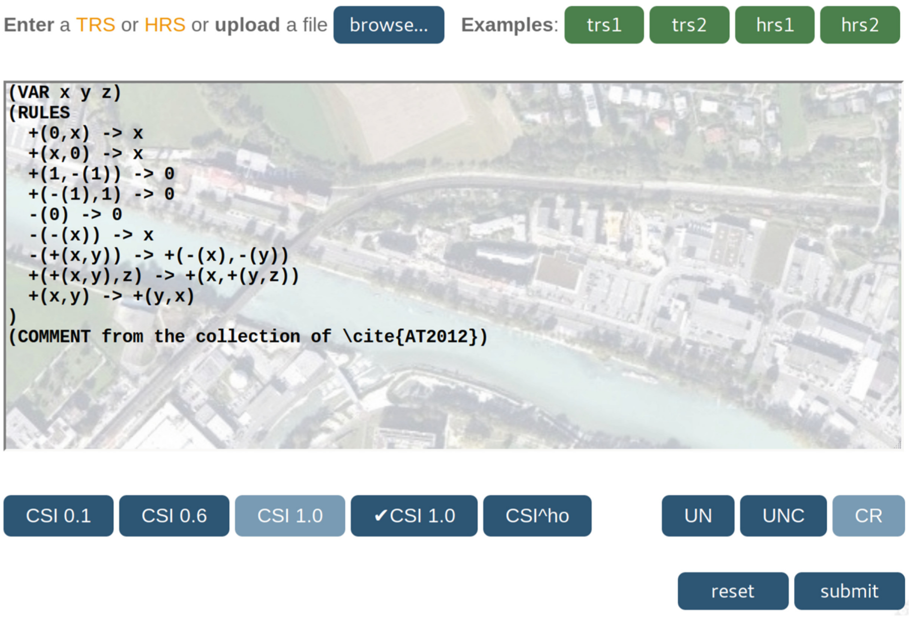
Due to the increasing interest in automatic analysis of rewrite systems in recent years, it is of great importance whether a proof, generated by an automatic tool, is indeed correct (cf. Example 4). Since the proofs produced by such tools are often complex and large, checking correctness is impractical for humans. Hence there is strong interest in verifying them using an independent certifier. A certifier is a tool that reads proof certificates, and either accepts them as correct or rejects them as erroneous. To ensure correctness of the certifier, the predominant solution is to use proof assistants like Coq or Isabelle to first formalize the underlying theory in the proof assistant and then use the formalization to obtain verified functions for inspecting the certificates.

As certifier we use CeTA [41], which reads certificates in CPF (certification problem format) [40]. Given a certificate CeTA will either answer `CERTIFIED`, or return a detailed error message why the proof was `REJECTED`. Its correctness is formally proved as part of IsaFoR, the Isabelle Formalization of Rewriting. IsaFoR contains executable check-functions for each formalized proof technique together with formal proofs that whenever such a check succeeds, the technique was indeed applied correctly. Isabelle's code-generation facility is used to obtain a trusted Haskell program from these check functions: the certifier CeTA.[8] Since 2012 CeTA supports checking (non-)confluence certificates. CSI supports certifiable output for the following criteria checkable by CeTA: Knuth and Bendix' criterion [17,39], (weak) orthogonality [25,35], Huet's results on strongly closed and parallel closed critical pairs and Toyama's extenson of the latter [12,24,42], the rule labeling heuristic for decreasing diagrams [23,45], and transformations based on redundant rules [22]. For non-confluence CeTA can check that, given derivations $s \to^* t_1$ and $s \to^* t_2$, t_1 and t_2 cannot be joined. Here the justifications used by CSI are: using tcap [44] (i.e., test that $\mathsf{tcap}(t_1\sigma)$ and $\mathsf{tcap}(t_2\sigma)$ are not unifiable), and reachability analysis using tree automata [11]. Experimental results for certified confluence analysis are presented in Sect. 7.

6 Implementation Details

CSI is open source and available as pre-compiled binary or via the web-interface shown in Fig. 1 from http://cl-informatik.uibk.ac.at/software/csi, CSI^ho can be obtained from http://cl-informatik.uibk.ac.at/software/csi/ho. Since its first release one of CSI's defining features has been its *strategy language*, which enables the combination techniques in a flexible manner and facilitates integration of new criteria. Some of the combinators provided to combine methods that we will use below are: sequential composition of strategies `;`, alternative composition `|` (which executes its second argument if the first fails), parallel execution `||`, and iteration `*`. A postfix `n*` executes a strategy at most `n` times while `[n]` executes its argument for at most `n` seconds. Finally `?` applies a strategy optionally (i.e., only if it makes progress), and `!` ensures that its argument only succeeds if confluence could be (dis)proved. For a full grammar of the strategy language pass the option `-h` to CSI. To illustrate its power we briefly compare the strategy used in CSI 0.1 with the one from CSI 1.0. The original strategy was

[8] IsaFoR/CeTA and CPF are available at http://cl-informatik.uibk.ac.at/software/ceta.



```
(VAR x y z)
(RULES
  +(0,x) -> x
  +(x,0) -> x
  +(1,-(1)) -> 0
  +(-(1),1) -> 0
  -(0) -> 0
  -(-(x)) -> x
  -(+(x,y)) -> +(-(x),-(y))
  +(+(x,y),z) -> +(x,+(y,z))
  +(x,y) -> +(y,x)
)
(COMMENT from the collection of \cite{AT2012})
```

Fig. 1. The new web-interface of CSI.

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

where `sorted -order` applies order-sorted decomposition and methods written in capitals are abbreviations for sub-strategies: `KB` applies Knuth-Bendix' criterion, `CLOSED` tests whether the critical pairs of a TRS are strongly or development closed, `DD` implements decreasing diagrams, and `NOTCR` tries to establish non-confluence. The current strategy is

```
(if trs then (sorted -order*;
(((GROUND || KB || AC || KH || AT || SIMPLE || CPCS2 ||
(REDUNDANT_DEL?; (CLOSED || DD || SIMPLE || KB || AC ||
GROUND))3*! || ((CLOSED || DD) | REDUNDANT_RHS)3*! ||
((CLOSED || DD) | REDUNDANT_JS)3*! || fail)[30] | CPCS[5]2*)2* ||
(NOTCR | REDUNDANT_FC)3*!)
) else fail)
```

which illustrates how to integrate new techniques independently or in combination with others, for instance the `REDUNDANT_X` strategies, which are different heuristics for finding redundant rules. The features described in Sect. 2 are reflected in `AC` and `REDUNDANT_FC`. The `AC` substrategy is simply tried in parallel to the existing methods for confluence and non-confluence. For the `REDUNDANT_FC` method, which *modifies* a problem, a different approach is used: first, a non-confluence proof (`NOTCR`) is attempted. If that fails, then rules from the forward closure are added, and the process is repeated, starting with another attempt at proving non-confluence. After 3 iterations, CSI gives up on the non-confluence check. Other additions are a decision procedure for ground systems [8] (`GROUND`), criteria by Klein and Hirokawa [15] (`KH`) and by Aoto and Toyama [2] (`AT`), simple to test syntactic criteria by Sakai, Oyamaguchi, and Ogawa [37], and Toyama and Oyamaguchi [43] (`SIMPLE`), and techniques based on critical pair closing systems [31] (`CPCS`). The full strategy configuration file (which consists of definitions of abbreviations like `AC`) grew from 76 to 233 lines since the initial release.

7 Experimental Results

For experiments[9] we considered all 291 TRSs in the Cops database. Table 1 compares the power of the current version of CSI (1.0) to its initial release (CSI 0.1 [44]) and to the version used in CoCo 2016 (0.6). For each problem, a tool may establish confluence (*yes*), non-confluence (*no*), or fail to give a conclusive answer (*maybe*), corresponding to the rows of the table. The progress achieved in the past few months is obvious. Of the 24 systems which CSI cannot handle, its main weakness is lack of special support for non-left-linear rules. Here for instance criteria based on quasi-linearity [4] and implemented in ACP are missing in CSI's repertoire. Some of the 24 systems are (currently) out of reach for all automatic confluence tools, like extensions of combinatory logic or self-distributivity.

The fourth column shows the results when using CSI's certifiable strategy, i.e., only criteria that can be checked by CeTA. Note that the *maybe* answers, in principle, include proofs produced by CSI that are not accepted by CeTA. However, because we also use Cops for testing the tools, this case does not occur for this set of problems. While all non-confluence proofs produced by CSI are certifiable there is still a gap in confluence analysis. The main missing techniques are a criterion to deal with AC rules, e.g. the ones from Sect. 2 or the one by Aoto and Toyama [2], advanced decomposition techniques based on layer systems [10], and techniques for dealing with non-left-linear systems, in particular the criteria by Klein and Hirokawa [15] and by Sakai, Oyamaguchi, and Ogawa [37]. The formalization and subsequent certification of most of these techniques requires serious effort, which we leave as future work.

Table 2 summarizes the results for UNR and UNC. We include CR in the table because proving confluence is a common way of establishing UNR or UNC.

Table 1. Confluence results.

	CSI 0.1	CSI 0.6	CSI 1.0	✓CSI 1.0
yes	115	179	206	116
no	46	55	61	61
maybe	130	57	24	114

	CSI^ho
yes	50
no	10
maybe	9

[9] Full details are available from CSI's website.

Table 2. Unique normal form results.

	$\top$	UNR	UNC	CR
$\top$	120	43	37	17
$\neg$CR	81	20	14	–
$\neg$UNC	33	6	–	–
$\neg$UNR	27	–	–	–

The $\top$ row (where $\top$ stands for *true*) represents the positive (yes) results for the corresponding properties, whereas the $\top$ column represents the negative results. For these experiments we used 120 TRSs which are comprised of the 100 Cops that at most one of the tools ACP, CoLL-Saigawa, or CSI could show confluent in the respective version used in CoCo 2016, and an additional 20 TRSs that were used in the UNR demonstration category in CoCo 2016. Note that the table entries overlap. For example, there are 20 problems for which CR has been disproved and UNR has been established; these 20 problems include the 14 problems which have been shown to satisfy UNC but not CR. The number of problems for which none of the properties UNR, UNC, or CR was proved or disproved is $120 + 20 - 81 - 43 = 16$.

For experiments in the higher-order setting we again used Cops, which contains 69 PRSs. CSIˆho can show confluence of 50 and non-confluence of 10 of these. Solving the remaining 9 systems will require serious effort—they contain e.g. lambda calculus with surjective pairing and self-distributivity of explicit substitution.

Acknowledgments. We thank Sarah Winkler for contributing code and expertise related to AC termination and AC critical pairs.

References

1. Aoto, T., Hirokawa, N., Nagele, J., Nishida, N., Zankl, H.: Confluence competition 2015. In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS, vol. 9195, pp. 101–104. Springer, Cham (2015). doi:10.1007/978-3-319-21401-6_5
2. Aoto, T., Toyama, Y.: A reduction-preserving completion for proving confluence of non-terminating term rewriting systems. LMCS **8**(1: 31), 1–29 (2012). doi:10.2168/LMCS-8(1:31)2012
3. Aoto, T., Toyama, Y.: Ground confluence prover based on rewriting induction. In: Proceedings of 1st FSCD. LIPIcs, vol. 52, pp. 33: 1–33: 12 (2016). doi:10.4230/LIPIcs.FSCD.2016.33
4. Aoto, T., Toyama, Y., Uchida, K.: Proving confluence of term rewriting systems via persistency and decreasing diagrams. In: Dowek, G. (ed.) RTA 2014. LNCS, vol. 8560, pp. 46–60. Springer, Cham (2014). doi:10.1007/978-3-319-08918-8_4
5. 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). doi:10.1007/978-3-642-02348-4_7

6. Appel, C., van Oostrom, V., Simonsen, J.G.: Higher-order (non-)modularity. In: Proceedings of 21st RTA. LIPIcs, vol. 6, pp. 17–32 (2010). doi:10.4230/LIPIcs.RTA.2010.17
7. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, New York (1998)
8. Felgenhauer, B.: Deciding confluence of ground term rewrite systems in cubic time. In: Proceedings of 23rd RTA. LIPIcs, vol. 15, pp. 165–175 (2012). doi:10.4230/LIPIcs.RTA.2012.165
9. Felgenhauer, B.: Efficiently deciding uniqueness of normal forms and unique normalization for ground TRSs. In: Proceedings of 5th IWC, pp. 16–20 (2016)
10. Felgenhauer, B., Middeldorp, A., Zankl, H., Oostrom, V.O.: Layer systems for proving confluence. ACM TOCL **16**(2: 14), 1–32 (2015). doi:10.1145/2710017
11. Felgenhauer, B., Thiemann, R.: Reachability analysis with state-compatible automata. In: Dediu, A.-H., Martín-Vide, C., Sierra-Rodríguez, J.-L., Truthe, B. (eds.) LATA 2014. LNCS, vol. 8370, pp. 347–359. Springer, Cham (2014). doi:10.1007/978-3-319-04921-2_28
12. Huet, G.: Confluent reductions: Abstract properties and applications to term rewriting systems. JACM **27**(4), 797–821 (1980). doi:10.23638/LMCS-13(2:4)2017
13. Jouannaud, J.P., Kirchner, H.: Completion of a set of rules modulo a set of equations. SIAM J. Comput. **15**(4), 1155–1194 (1986). doi:10.1137/0215084
14. Kahrs, S., Smith, C.: Non-ω-overlapping TRSs are UN. In: Proceedings of 1st FSCD. LIPIcs, vol. 52, pp. 22: 1–22: 17 (2016). doi:10.4230/LIPIcs.FSCD.2016.22
15. Klein, D., Hirokawa, N.: Confluence of non-left-linear TRSs via relative termination. In: Bjørner, N., Voronkov, A. (eds.) LPAR 2012. LNCS, vol. 7180, pp. 258–273. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28717-6_21
16. Klop, J.: Combinatory reduction systems. Ph.D. thesis, Utrecht University (1980)
17. 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)
18. Kop, C.: Higher order termination. Ph.D. thesis, Vrije Universiteit, Amsterdam (2012)
19. Kusakari, K., Isogai, Y., Sakai, M., Blanqui, F.: Static dependency pair method based on strong computability for higher-order rewrite systems. IEICE TIS **92–D**(10), 2007–2015 (2009)
20. Mayr, R., Nipkow, T.: Higher-order rewrite systems and their confluence. TCS **192**(1), 3–29 (1998). doi:10.1016/S0304-3975(97)00143--6
21. Miller, D.: A logic programming language with lambda-abstraction, function variables, and simple unification. JLP **1**(4), 497–536 (1991). doi:10.1093/logcom/1.4.497
22. Nagele, J., Felgenhauer, B., Middeldorp, A.: Improving automatic confluence analysis of rewrite systems by redundant rules. In: Proceedings of 26th RTA. LIPIcs, vol. 36, pp. 257–268 (2015). doi:10.4230/LIPIcs.RTA.2015.257
23. Nagele, J., Felgenhauer, B., Zankl, H.: Certifying confluence proofs via relative termination and rule labeling. LMCS (to appear) (2017)
24. Nagele, J., Middeldorp, A.: Certification of classical confluence results for left-linear term rewrite systems. In: Blanchette, J.C., Merz, S. (eds.) ITP 2016. LNCS, vol. 9807, pp. 290–306. Springer, Cham (2016). doi:10.1007/978-3-319-43144-4_18
25. Nagele, J., Thiemann, R.: Certification of confluence proofs using CeTA. In: Proceedings of 3rd IWC, pp. 19–23 (2014)
26. Nelson, G., Oppen, D.: Fast decision procedures based on congruence closure. JACM **27**(2), 356–364 (1980). doi:10.1145/322186.322198

27. Nipkow, T.: Higher-order critical pairs. In: Proceedings of 6th LICS, pp. 342–349 (1991). doi:10.1109/LICS.1991.151658
28. Nipkow, T.: Functional unification of higher-order patterns. In: Proceedings of 8th LICS, pp. 64–74 (1993). doi:10.1109/LICS.1993.287599
29. van Oostrom, V.: Developing developments. TCS **175**(1), 159–181 (1997). doi:10.1016/S0304-3975(96)00173-9
30. van Oostrom, V., Raamsdonk, F.: Weak orthogonality implies confluence: the higher-order case. In: Nerode, A., Matiyasevich, Y.V. (eds.) LFCS 1994. LNCS, vol. 813, pp. 379–392. Springer, Heidelberg (1994). doi:10.1007/3-540-58140-5_35
31. Oyamaguchi, M., Hirokawa, N.: Confluence and critical-pair-closing systems. In: Proceedings of 3rd IWC, pp. 29–33 (2014)
32. Peterson, G.E., Stickel, M.E.: Complete sets of reductions for some equational theories. JACM **28**(2), 233–264 (1981). doi:10.1145/322248.322251
33. van Raamsdonk, F.: On termination of higher-order rewriting. In: Middeldorp, A. (ed.) RTA 2001. LNCS, vol. 2051, pp. 261–275. Springer, Heidelberg (2001). doi:10.1007/3-540-45127-7_20
34. Regnier, L.: Une équivalence sur les lambda-termes. TCS **126**(2), 281–292 (1994). doi:10.1016/0304-3975(94)90012--4
35. Rosen, B.: Tree-manipulating systems and Church-Rosser theorems. JACM **20**(1), 160–187 (1973). doi:10.1145/321738.321750
36. Rubio, A.: A fully syntactic AC-RPO. I&C **178**(2), 515–533 (2002). doi:10.1006/inco.2002.3158
37. Sakai, M., Oyamaguchi, M., Ogawa, M.: Non-E-overlapping, weakly shallow, and non-collapsing TRSs are confluent. In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS (LNAI), vol. 9195, pp. 111–126. Springer, Cham (2015). doi:10.1007/978-3-319-21401-6_7
38. Shintani, K., Hirokawa, N.: CoLL: A confluence tool for left-linear term rewrite systems. In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS (LNAI), vol. 9195, pp. 127–136. Springer, Cham (2015). doi:10.1007/978-3-319-21401-6_8
39. Sternagel, C., Thiemann, R.: Formalizing Knuth-Bendix orders and Knuth-Bendix completion. In: Proceedings of 24th RTA. LIPIcs, vol. 21, pp. 287–302 (2013).doi:10.4230/LIPIcs.RTA.2013.287
40. Sternagel, C., Thiemann, R.: The certification problem format. In: Proceedings of 11th UITP. EPTCS, vol. 167, pp. 61–72 (2014). doi:10.4204/EPTCS.167.8
41. Thiemann, R., Sternagel, C.: Certification of termination proofs using CeTA. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 452–468. Springer, Heidelberg (2009). doi:10.1007/978-3-642-03359-9_31
42. Toyama, Y.: Commutativity of term rewriting systems. In: Fuchi, K., Kott, L. (eds.) Programming of Future Generation Computers II, pp. 393–407. North-Holland Publishing, North Holland (1988)
43. Toyama, Y., Oyamaguchi, M.: Church-Rosser property and unique normal form property of non-duplicating term rewriting systems. In: Proceedings of the 4th CTRS withDershowitz N., Lindenstrauss N. (eds.) CTRS 1994. LNCS, vol. 968 (1995). doi:10.1007/3-540-60381-6_19
44. Zankl, H., Felgenhauer, B., Middeldorp, A.: CSI – a confluence tool. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS, vol. 6803, pp. 499–505. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22438-6_38
45. Zankl, H., Felgenhauer, B., Middeldorp, A.: Labelings for decreasing diagrams. JAR **54**(2), 101–133 (2015). doi:10.1007/s10817-014-9316-y