Proving Confluence of Term Rewriting Systems Automatically

Takahito Aoto, Junichi Yoshida, and Yoshihito Toyama

RIEC, Tohoku University
{aoto,yoshida,toyama}@nue.riec.tohoku.ac.jp

Abstract. We have developed an automated confluence prover for term rewriting systems (TRSs). This paper presents theoretical and technical ingredients that have been used in our prover. A distinctive feature of our prover is incorporation of several divide–and–conquer criteria such as those for commutative (Toyama, 1988), layer-preserving (Ohlebusch, 1994) and persistent (Aoto & Toyama, 1997) combinations. For a TRS to which direct confluence criteria do not apply, the prover decomposes it into components and tries to apply direct confluence criteria to each component. Then the prover combines these results to infer the (non-)confluence of the whole system. To the best of our knowledge, an automated confluence prover based on such an approach has been unknown.

1 Introduction

Termination and confluence are considered to be central properties of term rewriting systems. Recently, automation of termination proving has been widely investigated in the literature. On the other hand, automation of confluence proving has not been well-known. Numerous results have been obtained on proving the confluence of term rewriting systems [5, 7, 11, 12, 14, 16–18, 20], but little work is reported on automation or an integration of these results on a confluence prover. This motivates us to develop a fully-automated confluence prover. A distinctive feature of our prover is incorporation of several divide–and–conquer criteria such as those for commutative [16], layer-preserving [9] and persistent [2] combinations. We present theoretical and technical ingredients that have been used in our prover, design of the system, and some experimental results.

2 Preliminaries

This section fixes some notions and notations used in this paper. We refer to [3] for omitted definitions.

The sets of *function symbols* and *variables* are denoted by $\mathcal{F}$ and V. We denote by $\mathcal{F}(t)$ and $\mathrm{V}(t)$ the sets of function symbols and variables occurring in a term t. The symbol in t at a position p is written as $t(p)$. The root position is denoted by ϵ. The (proper) subterm relation is denoted by $\trianglelefteq$ ($\triangleleft$). A *rewrite rule* $l \to r$ is a pair of terms l and r such that $l \notin V$ and $\mathrm{V}(l) \supseteq \mathrm{V}(r)$. It is

collapsing if $r \in V$ and *left-linear* if no variable occurs more than twice in l. Rewrite rules are identified modulo variable renaming. A *term rewriting system* (*TRS* for short) is a finite set of rewrite rules. We put $\mathcal{F}(l \to r) = \mathcal{F}(l) \cup \mathcal{F}(r)$, $\mathrm{V}(l \to r) = \mathrm{V}(l) \cup \mathrm{V}(r)$ and $\mathcal{F}(\mathcal{R}) = \bigcup_{l \to r \in \mathcal{R}} \mathcal{F}(l \to r)$.

A rewrite step $s \to_{\mathcal{R},p} t$ is defined by $s/p = l\sigma$ and $t = s[r\sigma]_p$ for some $l \to r \in \mathcal{R}$ and substitution σ. We omit subscripts $_{\mathcal{R}}$ and/or $_p$ if they are not important. The subterm s/p of s is the *redex* of the rewrite step $s \to_p t$. A term s is in *normal form* if $s \to t$ for no term t. The set of normal forms is denoted by $\mathrm{NF}(\mathcal{R})$. A rewrite step $s \to_p t$ is said to be *innermost* when any $u \lhd s/p$ is in normal form. An innermost rewrite step is written as $s \to^i t$. The reflexive transitive closure (reflexive closure) of a relation $\to$ is denoted by $\xrightarrow{*}$ (resp. $\xrightarrow{=}$). We define $\xrightarrow{*}{}^i$ and $\xrightarrow{=}{}^i$ similarly. A term s is in *head normal form* if there exists no redex t such that $s \xrightarrow{*} t$. The set of head normal forms is denoted by $\mathrm{HNF}(\mathcal{R})$. A *normal form* (*head normal form*) of s is a term $t \in \mathrm{NF}(\mathcal{R})$ (resp. $t \in \mathrm{HNF}(\mathcal{R})$) such that $s \xrightarrow{*} t$. A normal form of s is denoted by $s{\downarrow}$. Similarly, an *innermost normal form* of s is denoted by $s{\downarrow^i}$ which is obtained by replacing $\to$ by $\to^i$. We use $\circ$ for the composition of relations. Terms s and t are *joinable* (*innermost joinable*) if $s \xrightarrow{*} \circ \xleftarrow{*} t$ ($s \xrightarrow{*}{}^i \circ \xleftarrow{*}{}^i t$, respectively.) A TRS $\mathcal{R}$ is *confluent* (*locally confluent*) or *Church–Rosser* if $s \xleftarrow{*} \circ \xrightarrow{*} t$ (resp. $s \leftarrow \circ \to t$) implies s and t are joinable. A TRS $\mathcal{R}$ is *terminating* (*innermost-terminating*) if there exists no infinite rewrite sequence $s_1 \to s_2 \to s_3 \to \cdots$ ($s_1 \to^i s_2 \to^i s_3 \to^i \cdots$, respectively). A *development rewrite step* $\multimap$ [9] is inductively defined as follows: (1) $\xrightarrow{=} \subseteq \multimap$, (2) $s_i \multimap t_i$ $(1 \le i \le n)$ implies $f(s_1, \ldots, s_n) \multimap f(t_1, \ldots, t_n)$, or (3) $s \multimap t$ if $s/p = l\sigma$, $s[r\sigma']_p = t$ and $\sigma(x) \multimap \sigma'(x)$ for any $x \in V$.

Let s, t be terms whose variables are disjoint. The term s *overlaps* on t (at position p) when there exists a non-variable subterm $u = t/p$ of t such that u and s are unifiable. Let $l_1 \to r_1$ and $l_2 \to r_2$ be rewrite rules w.l.o.g. whose variables are disjoint. Suppose that l_1 overlaps on l_2 at position p. Let σ be the most general unifier of l_1 and l_2/p. Then the term $l_2[l_1]\sigma$ yields a peak $l_2[r_1]\sigma \leftarrow l_2[l_1]\sigma \to r_2\sigma$. The pair $\langle l_2[r_1]\sigma, r_2\sigma \rangle$ is called the *critical pair* obtained by the overlap of $l_1 \to r_1$ on $l_2 \to r_2$ at position p. In the case of self-overlap (i.e. when $l_1 \to r_1$ and $l_2 \to r_2$ are identical modulo renaming), we do not consider the case $p = \epsilon$. It is an *outer critical pair* if $p = \epsilon$; else an *inner critical pair*. The set $\mathrm{CP}_{in}(l_1 \to r_1, l_2 \to r_2)$ ($\mathrm{CP}_{out}(l_1 \to r_1, l_2 \to r_2)$) is the set of inner critical pairs (outer critical pairs, respectively) obtained by the overlap of $l_1 \to r_1$ on $l_2 \to r_2$. For TRSs $\mathcal{R}_1$ and $\mathcal{R}_2$ the set $\mathrm{CP}_{in}(\mathcal{R}_1, \mathcal{R}_2)$ of inner critical pairs are defined by $\bigcup_{l_1 \to r_1 \in \mathcal{R}_1} \bigcup_{l_2 \to r_2 \in \mathcal{R}_2} \mathrm{CP}_{in}(l_1 \to r_1, l_2 \to r_2)$. $\mathrm{CP}_{out}(\mathcal{R}_1, \mathcal{R}_2)$ is defined similarly. We set $\mathrm{CP}(\mathcal{R}_1, \mathcal{R}_2) = \mathrm{CP}_{in}(\mathcal{R}_1, \mathcal{R}_2) \cup \mathrm{CP}_{out}(\mathcal{R}_1, \mathcal{R}_2)$, $\mathrm{CP}_{\alpha}(\mathcal{R}) = \mathrm{CP}_{\alpha}(\mathcal{R}, \mathcal{R})$ ($\alpha \in \{in, out\}$) and $\mathrm{CP}(\mathcal{R}) = \mathrm{CP}_{in}(\mathcal{R}) \cup \mathrm{CP}_{out}(\mathcal{R})$. A TRS $\mathcal{R}$ is *overlay* if $\mathrm{CP}_{in}(\mathcal{R}) = \emptyset$. Note that $\langle s, t \rangle \in \mathrm{CP}_{out}(\mathcal{R})$ iff $\langle t, s \rangle \in \mathrm{CP}_{out}(\mathcal{R})$.

3 Direct Methods

In this section, we explain direct methods employed in our confluence prover.

Proposition 1 (Knuth–Bendix's criterion[8]). A terminating TRS $\mathcal{R}$ is confluent if and only if $s\downarrow = t\downarrow$ for all $\langle s, t\rangle \in \mathrm{CP}(\mathcal{R})$.

Thus it is decidable whether a terminating TRS $\mathcal{R}$ is confluent or not. Hence termination proving take an important part of the confluence proving procedure. Termination, however, is undecidable property of TRSs and we should take into account the case where the prover fails to prove termination of terminating TRSs.

Since termination implies innermost-termination, it is natural to expect that innermost-termination proving is more powerful than termination proving. This is especially true for recent termination tools based on dependency pairs (see e.g. [4]). This motivates the following criterion mentioned by Ohlebusch [10] pp.125–126, which can be easily proved by Theorem 3.23 of Gramlich [6].

Theorem 1 (Gramlich–Ohlebusch's criterion). For innermost-terminating overlay TRS $\mathcal{R}$, $\mathcal{R}$ is confluent if and only if $s\downarrow^i = t\downarrow^i$ for all $\langle s, t\rangle \in \mathrm{CP}(\mathcal{R})$.

Thus for overlay TRSs, one can safely switch the termination proof to the innermost-termination proof and try Gramlich–Ohlebusch's criterion instead of Knuth–Bendix's criterion.

When our confluence prover fails to detect (innermost) termination, our prover next checks several sufficient confluence conditions.

Proposition 2 (Huet–Toyama–van Oostrom's criterion[20]). A left-linear TRS $\mathcal{R}$ is confluent if (1) $s \multimap t$ for all $\langle s, t\rangle \in \mathrm{CP}_{in}(\mathcal{R})$ and (2) $s \multimap \circ \overset{*}{\leftarrow} t$ for all $\langle s, t\rangle \in \mathrm{CP}_{out}(\mathcal{R})$.

Because $u \overset{*}{\rightarrow} v$ is undecidable in general, we use $u \multimap v$ instead, i.e. our prover checks (2′) $s \multimap \circ \leftarrow\!\circ\!- t$ for all $\langle s, t\rangle \in \mathrm{CP}_{out}(\mathcal{R})$ in the place of (2). The check of the following criterion is approximated in a similar way.

Proposition 3 (Huet's strong-closedness criterion[7]). A linear TRS $\mathcal{R}$ is confluent if $s \overset{=}{\rightarrow} \circ \overset{*}{\leftarrow} t$ and $s \overset{*}{\rightarrow} \circ \overset{=}{\leftarrow} t$ for all $\langle s, t\rangle \in \mathrm{CP}(\mathcal{R})$.

So far there is no mechanism of detecting non-confluence of non-terminating TRSs. Therefore, we add a simple non-confluence check based on the following easy observation.

Theorem 2 (A simple non-confluence criterion). If there exist terms s', t' such that $s \overset{*}{\rightarrow} s'$ and $t \overset{*}{\rightarrow} t'$ for some $\langle s, t\rangle \in \mathrm{CP}(\mathcal{R})$ which satisfy either (1) $s', t' \in \mathrm{NF}(\mathcal{R})$ and $s' \neq t'$; or (2) $s', t' \in \mathrm{HNF}(\mathcal{R})$ and $s'(\epsilon) \neq t'(\epsilon)$. Then $\mathcal{R}$ is not confluent.

Since it is undecidable whether a term s is in head normal form, our prover checks a simple sufficient criterion $s(\epsilon) \notin \{l(\epsilon) \mid l \to r \in \mathcal{R}\}$ instead.

4 Divide and Conquer Methods

When all of direct methods fail to prove the (non-)confluence of the TRS, our prover next tries to infer it using divide–and–conquer approach. Several criteria to infer the (non-)confluence of a TRS from that of its subsystems are known [1, 2, 9, 15–17].

4.1 Persistent Decomposition

The first decomposition method our prover tries is the one based on the persistency [23], which is an extension of the direct-sum decomposition [15].

Definition 1 (persistent property[23]). A property $\mathcal{P}$ is said to be *persistent* if $\mathcal{P}$ holds for a many-sorted TRS $\mathcal{R}$ if and only if $\mathcal{P}$ holds for its underlying unsorted TRS $\Theta(\mathcal{R})$. Here Θ is the operation that forget the sort information.

Proposition 4 (persistency of confluence[2]). Confluence is a persistent property of TRSs.

Let us describe how persistency of confluence is used to show the confluence of a TRS from its subsystems.

Let $\mathcal{S}$ be a set of sorts. A sort attachment τ is a mapping $\mathcal{F} \to \mathcal{S}^*$ such that $\mathrm{arity}(f) = n$ implies $\tau(f) \in \mathcal{S}^{n+1}$, which is written as $f : \alpha_1 \times \cdots \times \alpha_n \to \alpha_0$. A sort attachment τ is consistent with a TRS $\mathcal{R}$ if for any $l \to r \in \mathcal{R}$, l and r are well-sorted under τ with the same sort. For any term t well-sorted under τ, let t^τ be the many-sorted term sorted by τ. Any sort attachment τ consistent with $\mathcal{R}$ induces a many-sorted TRS $\mathcal{R}^\tau = \{l^\tau \to r^\tau \mid l \to r \in \mathcal{R}\}$. Let us denote by T^τ the set of many-sorted terms, by T^τ_α the set of many-sorted terms of sort α, and let $\mathrm{T}^\tau_{\trianglelefteq\alpha} = \{t \in \mathrm{T}^\tau \mid t \trianglelefteq u \text{ for some } u \in \mathrm{T}^\tau_\alpha\}$. For any set A of terms closed under rewrite steps, let us write $\mathrm{Conf}(A)$ iff $s \to^* \circ \leftarrow^* t$ for any $s, t \in A$ such that $s \leftarrow^* \circ \to^* t$. By persistency, $\mathrm{Conf}(\mathrm{T}(\mathcal{F}, V))$ iff $\mathrm{Conf}(\mathrm{T}^\tau)$ iff $\mathrm{Conf}(\mathrm{T}^\tau_\alpha)$ for any sort α. Because any rewrite rule that can be applied to a term of sort α is in $\mathcal{R}^\tau \cap (\mathrm{T}^\tau_{\trianglelefteq\alpha})^2$, we have $\mathrm{Conf}(\mathrm{T}^\tau_\alpha)$ iff $\mathcal{R}^\tau \cap (\mathrm{T}^\tau_{\trianglelefteq\alpha})^2$ is confluent. Since the confluence of any set A of unsorted terms implies confluence of $\{t^\tau \mid t \in A\}$, $\mathcal{R}$ is confluent iff $\Theta(\mathcal{R}^\tau \cap (\mathrm{T}^\tau_{\trianglelefteq\alpha})^2)$ is confluent for any $\alpha \in \mathcal{S}$. Thus the following persistent criterion is obtained.

Definition 2 (persistent decomposition). Let $\mathcal{R}$ be a TRS and τ a sort attachment consistent with $\mathcal{R}$. Then $\max(\{\Theta(\mathcal{R}^\tau \cap (\mathrm{T}_{\trianglelefteq\alpha})^2) \mid \alpha \in \mathcal{S}\})$ is said to be a *persistent decomposition* of $\mathcal{R}$. Here max is the operation of taking maximal (w.r.t. subset relation) sets. We write $\mathcal{R} = \mathcal{R}_1 \oplus^\tau \cdots \oplus^\tau \mathcal{R}_n$ if $\{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ is a persistent decomposition of $\mathcal{R}$. A persistent decomposition is said to be *minimal* if each components has no persistent decomposition.

Theorem 3 (persistent criterion). A TRS $\mathcal{R} = \mathcal{R}_1 \oplus^\tau \cdots \oplus^\tau \mathcal{R}_n$ is confluent if and only if so is each $\mathcal{R}_i$.

Example 1 (confluence proof by persistency decomposition). Let

$$\mathcal{R}_a = \left\{ \begin{array}{lll} \mathsf{f}(\mathsf{a}(x), x) \to \mathsf{f}(x, \mathsf{a}(x)) & & \mathsf{g}(\mathsf{b}(x), y) \to \mathsf{g}(\mathsf{a}(\mathsf{a}(x)), y) \\ \mathsf{f}(\mathsf{b}(x), x) \to \mathsf{f}(x, \mathsf{b}(x)) & & \mathsf{g}(\mathsf{c}(x), y) \to y \\ \mathsf{a}(x) \to \mathsf{b}(x) & & \end{array} \right\}.$$

The direct methods do not apply to $\mathcal{R}_a$, because $\mathcal{R}_a$ is neither terminating nor left-linear. Consider the attachment τ on $\mathcal{S} = \{0, 1, 2\}$ such that $\mathsf{f} : 0 \times 0 \to 1$,

$\mathsf{a} : 0 \to 0$, $\mathsf{b} : 0 \to 0$, $\mathsf{c} : 0 \to 0$, and $\mathsf{g} : 0 \times 2 \to 2$. Then we have the following persistent decomposition of $\mathcal{R}_a$:

$$\mathcal{R}_{a1} = \left\{ \begin{array}{ll} \mathsf{f}(\mathsf{a}(x), x) & \to \mathsf{f}(x, \mathsf{a}(x)) \\ \mathsf{f}(\mathsf{b}(x), x) & \to \mathsf{f}(x, \mathsf{b}(x)) \\ \mathsf{a}(x) & \to \mathsf{b}(x) \end{array} \right\}, \quad \mathcal{R}_{a2} = \left\{ \begin{array}{lll} \mathsf{a}(x) & \to & \mathsf{b}(x) \\ \mathsf{g}(\mathsf{b}(x), y) & \to & \mathsf{g}(\mathsf{a}(\mathsf{a}(x)), y) \\ \mathsf{g}(\mathsf{c}(x), y) & \to & y \end{array} \right\}.$$

$\mathcal{R}_{a1}$ is confluent by Knuth–Bendix's criterion and $\mathcal{R}_{a2}$ is confluent by Huet–Toyama–van Oostrom's criterion. Thus the confluence of $\mathcal{R}_a$ follows from the persistency of confluence. □

Since a most general sort attachment consistent with a TRS is unique (modulo renaming of sorts), we know that a minimal persistent decomposition of $\mathcal{R}$ is unique. Since the smaller a component of persistent decomposition becomes, the easier it becomes to prove confluence of that component. Thus it suffices to compute the minimal persistent decomposition and try to prove (non-)confluence of each components. Furthermore, since the persistency is a necessary and sufficient criterion, if non-confluence is detected in any component then non-confluence of the whole TRS is concluded.

4.2 Layer-Preserving Decomposition

The second decomposition method our prover tries is the one based on the (composable) layer-preserving combination [9]. In what follows, $\mathcal{D}(l \to r) = \{l(\epsilon), r(\epsilon)\}$, $\setminus$ and $\uplus$ stand for the set difference and disjoint union.

Definition 3 (layer-preserving[9]). A pair $\langle \mathcal{R}_1, \mathcal{R}_2 \rangle$ of TRSs is said to be a *layer-preserving* combination if there exists a partition $\mathcal{D}_1 \uplus \mathcal{D}_2 \uplus \mathcal{C}$ of function symbols such that (1) for any $\alpha \in \mathcal{R}_1 \setminus \mathcal{R}_2$, $\mathcal{D}(\alpha) \subseteq \mathcal{D}_1$ and $\mathcal{F}(\alpha) \subseteq \mathcal{D}_1 \cup \mathcal{C}$ (2) for any $\alpha \in \mathcal{R}_2 \setminus \mathcal{R}_1$, $\mathcal{D}(\alpha) \subseteq \mathcal{D}_2$ and $\mathcal{F}(\alpha) \subseteq \mathcal{D}_2 \cup \mathcal{C}$ (3) for any $\alpha \in \mathcal{R}_1 \cap \mathcal{R}_2$, $\mathcal{F}(\alpha) \subseteq \mathcal{C}$. If $\langle \mathcal{R}_1, \mathcal{R}_2 \rangle$ is a layer-preserving combination the union $\mathcal{R}_1 \cup \mathcal{R}_2$ is denoted by $\mathcal{R}_1 \Cup \mathcal{R}_2$.

Based on the definition of layer-preserving combination given above we define the layer-preserving decomposition as follows.

Definition 4 (layer-preserving decomposition). A set $\{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ ($n \geq 1$, $\mathcal{R}_i \neq \emptyset$) of TRSs is said to be a *layer-preserving decomposition* of $\mathcal{R}$ (denoted by $\mathcal{R} = \mathcal{R}_1 \Cup \cdots \Cup \mathcal{R}_n$) if $\mathcal{R} = \mathcal{R}_1 \cup \cdots \cup \mathcal{R}_n$ and $\langle \mathcal{R}_i, \mathcal{R}_j \rangle$ is a layer-preserving combination for $i \neq j$. A layer-preserving decomposition is said to be *minimal* if each component has no layer-preserving decomposition.

Proposition 5 (layer-preserving criterion[9]). A TRS $\mathcal{R} = \mathcal{R}_1 \Cup \cdots \Cup \mathcal{R}_n$ is confluent if and only if so is each $\mathcal{R}_i$.

Example 2 (confluence proof by layer-preserving decomposition). Let

$$\mathcal{R}_b = \left\{ \begin{array}{lll} \mathsf{f}(x, \mathsf{a}(\mathsf{g}(x))) \to \mathsf{g}(\mathsf{f}(x, x)) & & \mathsf{a}(x) \to x \\ \mathsf{f}(x, \mathsf{g}(x)) \to \mathsf{g}(\mathsf{f}(x, x)) & & \mathsf{h}(x) \to \mathsf{h}(\mathsf{a}(\mathsf{h}(x))) \end{array} \right\}$$

It is clear that the direct methods do not apply to $\mathcal{R}_b$. Taking the partition of function symbols as $\mathcal{D}_1 = \{\mathsf{f}, \mathsf{g}\}, \mathcal{D}_2 = \{\mathsf{h}\}, \mathcal{C} = \{\mathsf{a}\}$, we have the following layer-preserving decomposition of $\mathcal{R}_b$:

$$\mathcal{R}_{b1} = \left\{ \begin{array}{ll} \mathsf{f}(x, \mathsf{a}(\mathsf{g}(x))) & \to \mathsf{g}(\mathsf{f}(x, x)) \\ \mathsf{f}(x, \mathsf{g}(x)) & \to \mathsf{g}(\mathsf{f}(x, x)) \\ \mathsf{a}(x) & \to x \end{array} \right\}, \ \mathcal{R}_{b2} = \left\{ \begin{array}{l} \mathsf{a}(x) \to x \\ \mathsf{h}(x) \to \mathsf{h}(\mathsf{a}(\mathsf{h}(x))) \end{array} \right\}.$$

$\mathcal{R}_{b1}$ is confluent by Knuth–Bendix's criterion and $\mathcal{R}_{b2}$ is confluent by Huet–Toyama–van Oostrom's criterion. Thus the confluence of $\mathcal{R}_b$ follows from the layer-preserving criterion. □

Note that the layer-preserving decomposition does not apply to $\mathcal{R}_a$ in Example 1 because of a collapsing rule. Similarly, the persistent decomposition does not apply to $\mathcal{R}_b$ in Example 2 because the only possible sort attachment is on a single sort. Thus, the two decompositions are incomparable.

One can show that a minimal layer-preserving decomposition is unique. Same as the case of persistent decomposition, it suffices to compute the minimal layer-preserving decomposition and if non-confluence is detected in any component then non-confluence of the whole TRS is concluded.

4.3 Commutative Decomposition

The third and last decomposition method our prover tries is the one based on commutation [13].

Definition 5 (commutation[13]). TRSs $\mathcal{R}_1$ and $\mathcal{R}_2$ *commute* if $\overset{*}{\leftarrow}_{\mathcal{R}_1} \circ \overset{*}{\rightarrow}_{\mathcal{R}_2} \subseteq \overset{*}{\rightarrow}_{\mathcal{R}_2} \circ \overset{*}{\leftarrow}_{\mathcal{R}_1}$. If $\mathcal{R}_1$ and $\mathcal{R}_2$ commute then their union $\mathcal{R}_1 \cup \mathcal{R}_2$ is denoted by $\mathcal{R}_1 \sqcup \mathcal{R}_2$.

Definition 6 (commutative decomposition). A set $\{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ ($n \geq 1$, $\mathcal{R}_i \neq \emptyset$) of TRSs is said to be a *commutative decomposition* of $\mathcal{R}$ (denoted by $\mathcal{R} = \mathcal{R}_1 \sqcup \cdots \sqcup \mathcal{R}_n$) if $\mathcal{R} = \mathcal{R}_1 \cup \cdots \cup \mathcal{R}_n$ and $\mathcal{R}_i, \mathcal{R}_j$ commute for $i \neq j$. A commutative decomposition is said to be *minimal* if each component has no commutative decomposition.

Proposition 6 (commutativity criterion[13]). A TRS $\mathcal{R} = \mathcal{R}_1 \sqcup \cdots \sqcup \mathcal{R}_n$ is confluent if so is each $\mathcal{R}_i$.

Since the commutativity criterion is merely a sufficient criterion, unlike previous two decompositions, it can not be used to infer the non-confluence from that of its subsystems. Moreover, since non-left-linear rules destroy commutativity, the commutative decomposition is restricted to left-linear TRSs.

Commutativity of TRSs is undecidable in general but a sufficient condition is known [16]. Our prover employs a slightly more general condition which is obtained by extending the proof of [20] along the line of [16][1].

[1] The detailed proof can be found in [22].

Proposition 7 (sufficient condition for commutativity[22]). Left-linear TRSs $\mathcal{R}_1$ and $\mathcal{R}_2$ commute if (1) $s \multimap_{\mathcal{R}_2} t$ for any $\langle s,t\rangle \in \mathrm{CP}_{in}(\mathcal{R}_1,\mathcal{R}_2)$, and (2) $s \multimap_{\mathcal{R}_1} \circ \overset{*}{\leftarrow}_{\mathcal{R}_2} t$ for any $\langle t,s\rangle \in \mathrm{CP}(\mathcal{R}_2,\mathcal{R}_1)$.

Condition (2) is undecidable and our prover uses a condition $s \multimap_{\mathcal{R}_1} \circ \leftarrow\!\!\circ_{\mathcal{R}_2} t$ instead of the condition $s \multimap_{\mathcal{R}_1} \circ \overset{*}{\leftarrow}_{\mathcal{R}_2} t$.

Example 3 (confluence proof by commutative decomposition). Let

$$\mathcal{R}_c = \left\{ \begin{array}{lll} \mathsf{f}(x) & \to \mathsf{g}(x) & \mathsf{f}(x) \to \mathsf{h}(\mathsf{f}(x)) \\ \mathsf{h}(\mathsf{f}(x)) & \to \mathsf{h}(\mathsf{g}(x)) & \mathsf{g}(x) \to \mathsf{h}(\mathsf{g}(x)) \end{array} \right\}.$$

Neither the direct methods nor the previous decomposition methods apply to $\mathcal{R}_c$. One can divide $\mathcal{R}_c$ into the following $\mathcal{R}_{c1}$ and $\mathcal{R}_{c2}$ which commute from Proposition 7.

$$\mathcal{R}_{c1} = \left\{ \begin{array}{ll} \mathsf{f}(x) & \to \mathsf{g}(x) \\ \mathsf{h}(\mathsf{f}(x)) & \to \mathsf{h}(\mathsf{g}(x)) \end{array} \right\},\ \mathcal{R}_{c2} = \left\{ \begin{array}{ll} \mathsf{f}(x) & \to \mathsf{h}(\mathsf{f}(x)) \\ \mathsf{g}(x) & \to \mathsf{h}(\mathsf{g}(x)) \end{array} \right\}.$$

$\mathcal{R}_{c1}$ is confluent by Knuth–Bendix's criterion and $\mathcal{R}_{c2}$ is confluent by Huet–Toyama–van Oostrom's criterion. Thus, the confluence of $\mathcal{R}_c$ follows from the commutativity criterion. □

Contrast to the persistent or layer-preserving decompositions, a minimal commutative decomposition is not unique. Furthermore, unlike the previous two decompositions, it does not always hold that a smaller decomposition is more useful to prove confluence [22], i.e. there is an example that can be proved by a non-minimal commutative decomposition but minimal ones fail.

5 Implementation and Experiments

We have implemented a confluence prover ACP (Automated Confluence Prover) in SML/NJ; the length of codes is about 14,000 lines[2]. It has a command line interface which takes an argument to specify a filename containing a TRS specification in TPDB[3] format. Several options are supported so that partial combinations of decompositions can be tested. An external termination prover can be specified in the place of an internal termination prover.

The overview of the prover is illustrated in Figure 1. Procedure *Direct* consists of the direct methods explained in Section 1. If *Direct* fails (i.e. neither confluence nor non-confluence is detected), then the prover finds the minimal persistent decomposition $\mathcal{R} = \mathcal{R}_1 \oplus^{\tau} \cdots \oplus^{\tau} \mathcal{R}_n$. If the decomposition is not proper (i.e. $n = 1$) then the prover tries a minimal layer-preserving decomposition to $\mathcal{R} = \mathcal{R}_1$. If

[2] A heap image of ACP that can be loaded into an SML/NJ runtime system, examples used for the experiments, and details of experiments can be obtained from `http://www.nue.riec.tohoku.ac.jp/tools/acp/`.

[3] `http://www.lri.fr/~marche/tpdb/`

the decomposition is proper (i.e. $n > 1$) then the prover applies the procedure *Direct* to each components $\mathcal{R}_1, \ldots, \mathcal{R}_n$. For each component $\mathcal{R}_i$ on which *Direct* fails, then the prover tries a minimal layer-preserving decomposition and so on. When the direct methods and successive three kinds of proper decompositions fail, the prover aborts the proof of (non-)confluence of (that component of) the system. Furthermore, if there is another possibility of decompositions then the prover backtracks and tries another decomposition.

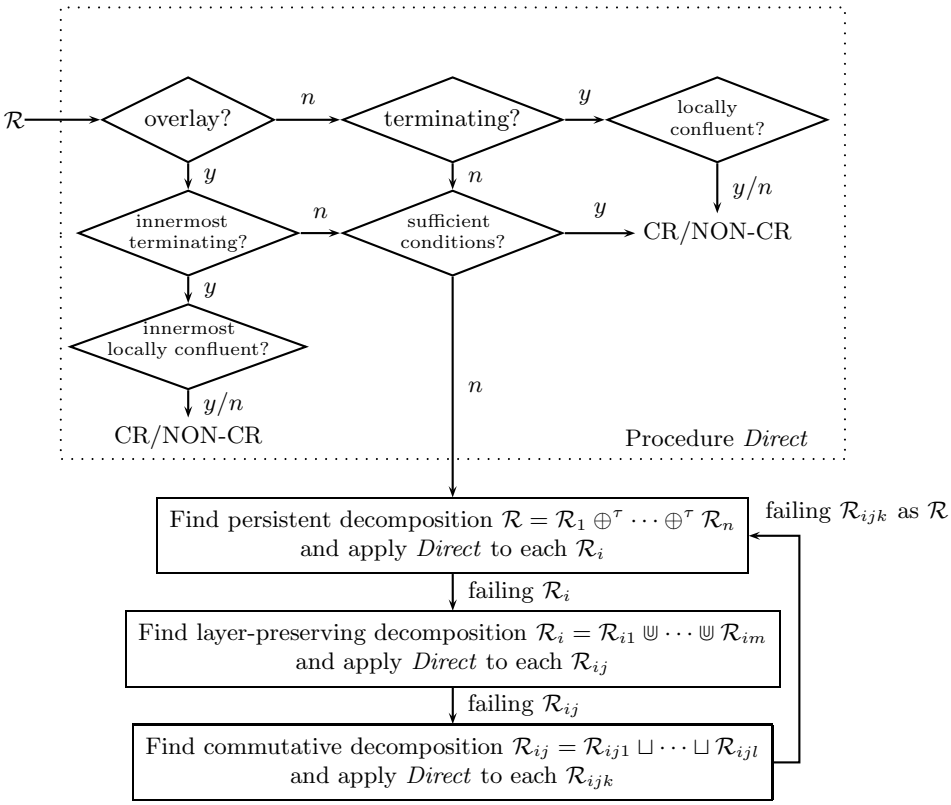

Fig. 1. Overview of ACP

For the experiment, we prepare a collection of 103 examples extracted from the literatures on confluence. We have tested confluence proving using various combinations of decomposition techniques. All tests have been performed in a PC equipped with Intel Xeon processors of 2.66GHz and a memory of 7GB.

The table below summarizes our experimental results. d, p, l, c, c' stand for direct methods, (minimal) persistent decomposition, (minimal) layer-preserving

decomposition, minimal commutative decomposition, (possibly non-minimal) commutative decomposition, respectively.

	d	dp	dl	dc	dc'	dpl	dpc	dpc'	dlc	dlc'	dplc	dplc'
success (CR)	30	35	34	41	43	37	47	49	46	47	49	50
success (NON-CR)	13	13	13	13	13	13	13	13	13	13	13	13
failure	60	55	56	49	47	53	43	41	44	43	41	40
timeout (60 sec.)	0	0	0	0	0	0	0	0	0	0	0	0
total time (sec.)	4.1	4.4	4.8	7.0	33.6	4.9	8.1	30.9	7.6	34.2	8.1	34.4

It is seen that decomposition techniques are effective to prove confluence, although it is ineffective to prove non-confluence. Each decomposition techniques succeeds to prove confluence of some different examples. Commutative decomposition is costly but most powerful of three decompositions.

6 Conclusion

We have presented an automated confluence prover ACP for TRSs, in which divide–and–conquer approach based on the persistent, layer-preserving, commutative decompositions is employed. To the best of our knowledge, an automated confluence prover based on such an approach has been unknown. We believe that our approach is useful to integrate different (non-)confluence criteria to prove the (non-)confluence of large systems.

The previous version of our confluence prover has been described in [22]. The new version is different in the following points in particular: new direct criteria (Gramlich–Ohlebusch's criterion, Huet's strong-closedness criterion, a simple non-confluence criterion) are added; the direct-sum decomposition is replaced with the persistent decomposition; the layer-preserving decomposition is added; the algorithm for computing commutative decompositions is changed to improve the efficiency.

There are still many confluence criteria which are not included in our prover—for example, stronger sufficient criteria for left-linear TRSs (e.g. [11, 12, 18]), the decreasing diagram technique (e.g. [19, 21]) and decision procedures for some subclass of TRSs (e.g. [5, 14]). It is our future work to include these criteria and make the system more powerful.

Acknowledgments

Thanks are due to anonymous referees for detailed and helpful comments. This work was partially supported by grants from JSPS, Nos. 19500003 and 20500002.

References

1. T. Aoto and Y. Toyama. On composable properties of term rewriting systems. In *Proc. of ALP'97 – HOA'97*, volume 1298 of *LNCS*, pages 114–128. Springer-Verlag, 1997.

2. T. Aoto and Y. Toyama. Persistency of confluence. *Journal of Universal Computer Science*, 3(11):1134–1147, 1997.
3. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
4. J. Giesl, R. Thiemann, and P. Schneider-Kamp. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.
5. G. Godoy and A. Tiwari. Confluence of shallow right-linear rewrite systems. In *Proc. of CSL'05*, volume 3634 of *LNCS*, pages 541–556. Springer-Verlag, 2005.
6. B. Gramlich. Abstract relations between restricted termination and confluence property of rewrite systems. *Fundamenta Informaticae*, 24:3–23, 1995.
7. G. Huet. Confluent reductions: abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.
8. D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.
9. E. Ohlebusch. *Modular Properties of Composable Term Rewriting Systems*. PhD thesis, Universität Bielefeld, 1994.
10. E. Ohlebusch. *Advanced Topics in Term Rewriting Systems*. Springer-Verlag, 2002.
11. S. Okui. Simultaneous critical pairs and Church-Rosser property. In *Proc. of RTA-98*, volume 1379 of *LNCS*, pages 2–16. Springer-Verlag, 1998.
12. M. Oyamaguchi and Y. Ohta. A new parallel closed condition for Church-Rosser of left-linear TRS's. In *Proc. of RTA-97*, volume 1232 of *LNCS*, pages 187–201. Springer-Verlag, 1997.
13. B. K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *Journal of the ACM*, 20:160–187, 1973.
14. A. Tiwari. Deciding confluence of certain term rewriting systems in polynomial time. In *Proc. of LICS 2002*, pages 447–458. IEEE Computer Society Press, 2002.
15. Y. Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1):128–143, 1987.
16. Y. Toyama. Commutativity of term rewriting systems. In K. Fuchi and L. Kott, editors, *Programming of Future Generation Computers II*, pages 393–407. North-Holland, Amsterdam, 1988.
17. Y. Toyama. Confluent term rewriting systems (invited talk). In *Proc. of RTA 2005*, volume 3467 of *LNCS*, page 1. Springer-Verlag, 2005. Slides are available from `http://www.nue.riec.tohoku.ac.jp/user/toyama/slides/toyama-RTA05.pdf`.
18. Y. Toyama and M. Oyamaguchi. Conditional linearization of non-duplicating term rewriting systems. *IEICE Trans. Information and Systems*, E84-D(5):439–447, 2001.
19. V. van Oostrom. Confluence by decreasing diagrams. *Theoretical Computer Science*, 126(2):259–280, 1994.
20. V. van Oostrom. Developing developments. *Theoretical Computer Science*, 175(1):159–181, 1997.
21. V. van Oostrom. Confluence by decreasing diagrams: converted. In *Proc. of RTA 2008*, volume 5117 of *LNCS*, pages 306–320. Springer-Verlag, 2008.
22. J. Yoshida, T. Aoto, and Y. Toyama. Automating confluence check of term rewriting systems. *Computer Software*, to appear. In Japanese.
23. H. Zantema. Termination of term rewriting: interpretation and type elimination. *Journal of Symbolic Computation*, 17:23–50, 1994.