Certification of Confluence- and Commutation-Proofs via Parallel Critical Pairs*

Nao Hirokawa
JAIST
Japan
hirokawa@jaist.ac.jp

Dohan Kim
University of Innsbruck
Austria
dohan.kim@uibk.ac.at

Kiraku Shintani
JAIST
Japan
s1820017@jaist.ac.jp

René Thiemann
University of Innsbruck
Austria
rene.thiemann@uibk.ac.at

Abstract

Parallel critical pairs (PCPs) have been used to design sufficient criteria for confluence of term rewrite systems. In this work we formalize PCPs and the criteria of Gramlich, Toyama, and Shintani and Hirokawa in the proof assistant Isabelle. In order to reduce the amount of bureaucracy we deviate from the paper-definition of PCPs, i.e., we switch from a position-based definition to a context-based definition. This switch not only simplifies the formalization task, but also gives rise to a simple recursive algorithm to compute PCPs. We further generalize all mentioned criteria from confluence to commutation and integrate them in the certifier CeTA, so that it can now validate confluence- and commutation-proofs based on PCPs. Because of our results, CeTA is now able to certify proofs by the automatic confluence tool Hakusan, which makes heavy use of PCPs. These proofs include term rewrite systems for which no previous certified confluence proof was known.

CCS Concepts: • **Theory of computation → Logic and verification**; **Equational logic and rewriting**.

Keywords: Certification, Commutation, Confluence, Isabelle

*The authors are listed in alphabetical order regardless of individual contributions or seniority.

CPP '24, January 15–16, 2024, London, UK

ACM ISBN 979-8-4007-0488-8/24/01
https://doi.org/10.1145/3636501.3636949

ACM Reference Format:
Nao Hirokawa, Dohan Kim, Kiraku Shintani, and René Thiemann. 2024. Certification of Confluence- and Commutation-Proofs via Parallel Critical Pairs. In *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '24), January 15–16, 2024, London, UK*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3636501.3636949

1 Introduction

Confluence is an important property of term rewriting systems (TRSs), which has been extensively researched using a wide variety of approaches [6, 10, 12, 28, 29, 31, 33]. One of the common approaches to investigate confluence is to use critical pairs. In particular, it is well-known that a terminating TRS is confluent iff every critical pair of it is joinable. This is a simple and straightforward method for proving confluence, but the assumption of a terminating TRS is a rather strong assumption for proving confluence of a TRS.

There are also many approaches for proving confluence of a TRS without assuming termination. Here, *parallel critical pairs* (PCPs) play an important role (see, for example, [6, 23, 28, 33]). Specifically, confluence criteria using PCPs for left-linear TRSs subsume many earlier confluence criteria for left-linear TRSs [23, 28, 33].

The first aim of this paper is to formally prove several PCP-based confluence criteria in a proof assistant. In detail, we formalize the PCP-based confluence criteria by Gramlich [6] (**G96**), Toyama [28] (**T81**), and Shintani and Hirokawa [23, Theorems 31 and 38] using Isabelle/HOL [22]. In this way, our formalization, in particular the combination of the two criteria by Shintani and Hirokawa (**SH22**), subsumes Knuth and Bendix' criterion [12] in left-linear cases (**KB70**), Toyama's almost parallel closedness [29] (**T88**), PCP-based rule labeling [33] (**ZFM15**), the confluence criteria based on critical pair system [7] (**HM11**), and the generalization of Knuth–Bendix criterion and PCP-based rule labeling by Dowek et al. [4] in first-order cases (**DFJL22**); see Figure 1 for the relationships among these criteria in confluence proving power.

In order to improve the reliability of automated confluence tools such as ACP [1] and CSI [32], their generated

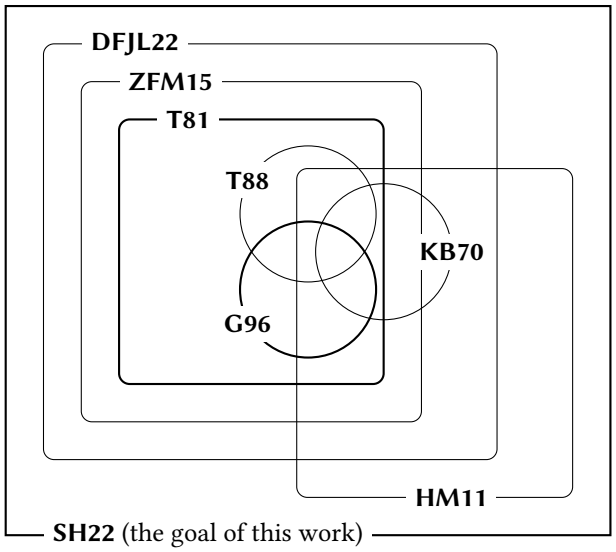

Figure 1. Comparison of the confluence criteria.

proofs can be certified. This is done by CeTA [25], a verified certifier that checks whether confluence techniques have been applied correctly in such proofs. However, the power of CeTA is limited, as prior to this work it did not support any confluence technique that uses PCPs. Consequently, Hakusan [23] was unsupported by CeTA, since Hakusan is solely based on PCPs. The second aim of this paper is therefore, to integrate verified checkers into CeTA that are capable of certifying PCP-based confluence proofs. The third aim is to evaluate the new support of PCPs by CeTA. To this end, we add certificate generation to Hakusan. With the newly added support of PCPs by CeTA and the certificate generation of Hakusan, now all proofs of Hakusan can be certified by CeTA, in particular there are 20 TRSs for which a certified proof has been generated for the first time, i.e., where ACP and CSI could not find a certifiable proof.

Our formalization is based on the library IsaFoR [27], the Isabelle Formalization of Rewriting. In this way we can reuse many known notions, results and algorithms that are required for this work, e.g., the (parallel) rewrite relation, multihole contexts, and a unification algorithm for first-order terms. However, prior to this work IsaFoR did not even contain the notion of a parallel critical pair, and hence also none of those mentioned confluence criteria that are based on PCPs.

We like to stress that our formal proofs deviate from the existing proofs in three different ways. First, we provide an alternative definition of PCPs. The original definition is based on parallelism of positions, unification, and renaming (α-conversion), but they are factors to complicate the use of (structural) induction. The new definition, using multihole contexts and a dedicated unification, facilitates both the formalization task and the (verified) computation of PCPs. Second, we generalize all these aforementioned confluence criteria into commutation criteria. This was often conveniently possible with the help of the proof assistant: Isabelle quickly identified those places where the proofs needed adjustment when generalizing from confluence to commutation. Third, we do not impose the typical variable conditions for TRSs.

The remainder of this paper is organized as follows. In Section 2 we briefly give preliminaries on Isabelle, term rewriting, and parallel critical pairs. In Section 3 we achieve the first aim: we provide a new context-based definition of PCPs and discuss our formalization of PCP-based commutation techniques that generalize PCP-based confluence techniques. In Section 4 we tackle the second aim and describe the development of the verified checkers that support the various PCP-based confluence and commutation criteria. The third aim is discussed in Section 5. Effectiveness of the formalized criteria and performance of certification are evaluated on problems taken from the Confluence Problem Database (COPS) [8]. We summarize in Section 6.

The formalizations that are described in Section 3 and in Section 4 are now part of IsaFoR, and IsaFoR also contains the full soundness proof of CeTA. Both IsaFoR and CeTA are available at the website:

http://cl-informatik.uibk.ac.at/isafor/

Throughout this paper we illustrate the newly developed parts either in Isabelle syntax or in standard mathematical text. In the former case, there might be slight deviations to the actual Isabelle source, e.g., intermediate let-bindings might have been introduced in this paper so that the Isabelle text fits into the 2-column layout. In both cases, the precise Isabelle sources can be observed at a dedicated webpage of IsaFoR/CeTA.[1] To this end, there are numerous hyperlinks in this paper. These are always indicated by a small version of the Isabelle logo and a first such hyperlink provides an overview of the new results.

The confluence tool Hakusan used for evaluating our formalization work is available at:

https://www.jaist.ac.jp/project/saigawa/

The website includes experimental data discussed in Section 5.

Related work. We are not aware of any other formalization of parallel critical pairs. However, commutation techniques have been formalized prior to this work.

Nagele and Middeldorp mechanized a criterion to ensure commutation for parallel closed TRSs [20], a result that is fully subsumed by our new commutation criteria for PCPs: Shintani and Hirokawa proved the subsumption for confluence instead of commutation [23], and their subsumption proof generalizes to commutation.

[1] http://cl-informatik.uibk.ac.at/isafor/experiments/pcp/

Furthermore, Kohl and Middeldorp formalized a commutation criterion for development closed TRSs [13, 14]. Their result is incomparable to our results, i.e., neither does their criterion subsume our PCP-based criteria, nor vice-versa. However, there is a common theme in these formalizations, namely the avoidance of explicit positions. They use proof terms to concisely describe multistep rewriting without positions, and we use multihole contexts to concisely describe parallel rewriting and PCPs without positions. Both parallel and development closedness results have been formalized as part of IsaFoR and proofs based on them are certifiable by CeTA.

Finally, Mitterwallner et al. verified decision procedures for the first-order theory of rewriting [17], a theory that is strong enough to express (ground) commutation. However, their approach is limited to TRSs that satisfy strong syntactic criteria, namely, every variable may occur at most once in a rule. In particular, variables that occur in ℓ must not be used in r for all rules $\ell \to r$. Their work is incomparable to ours in two ways. The formalization is different as their main algorithms are tree-automata based, and in this work tree-automata are not used at all. Moreover, their certifier FORTify implements a verified decision procedure for a sub-class of TRSs, whereas here we formalize just sufficient criteria, but for the much broader class of left-linear TRSs.

2 Preliminaries

2.1 Isabelle

Isabelle [22] is a generic proof assistant and we use it in combination with higher-order logic, i.e., Isabelle/HOL. Although we develop our formalization in a specific proof assistant, most of our formalization could have been conducted in any proof assistant.

- We do not require any advanced features of the proof assistant, i.e., we mostly specify our properties and algorithms with algebraic data types, sets, and functional programs.
- It was helpful to specify subtypes that restrict a given type to satisfy certain properties. For instance, in Isabelle `typedef sub_ty = {x :: ty. p x}` defines a new type `sub_ty` that contains all elements `x` of type `ty` that satisfy the property `p`. Afterwards one can easily define functions involving `sub_ty` by lifting functions that use `ty`. The corresponding command is **lift-definition** from the lifting and transfer package [11].
- The only real dependence on the Isabelle proof assistance is the usage of the IsaFoR library.

We just provide a brief explanation of Isabelle notation that is used in this paper, and refer to the Isabelle documentation for further details [26].

We illustrate the syntax of types in Isabelle by the example type `'a :: infinite × nat ⇒ 'a set`. It indicates a polymorphic type since it uses a type variable `'a`, and this variable can be instantiated with any concrete type that is infinite. The example type is the type of functions that take a pair of an element of type `'a` and a natural number (`nat`) and return a set of elements of type `'a`.

Function definitions in Isabelle allow standard programming concepts like pattern-matching, case-analysis, lambda-abstractions, and let-bindings. Moreover, definitions do not need to be executable, e.g., they may also contain quantifiers and set-comprehension. We use all of these concepts without further explanation.

The Isabelle distribution already contains many function definitions known from functional programming, e.g., `fst` and `snd` are the selectors of pairs `(x,y)`; `xs ! n` is the `n`-th element of list `xs`; `concat` merges a list of lists into a single list; `@` is append; `set` converts a list into a set; `map2 :: ('a × 'b ⇒ 'c) ⇒ 'a list ⇒ 'b list ⇒ 'c list` applies a function on two zipped lists; and `None` and `Some x` are the two possible shapes for a value of the option-type in Isabelle.

2.2 Term Rewriting

We recapitulate basic notions from term rewriting [2].

We consider first-order terms over some signature $\mathcal{F}$ (consisting of function symbols $a, b, \ldots, f, g, \ldots$ with fixed arities) and some infinite set of variables $x, y, \ldots \in \mathcal{V}$. A *position* within a term is a list of indices where ε denotes the empty position, also called the *root position*. The set of positions of a term t is defined as $\mathcal{P}os(t)$, and $\mathcal{V}\mathrm{ar}(t)$ is the set of variables that occur in t. Given $p \in \mathcal{P}os(t)$ we write $t|_p$ for the subterm of t at position p, i.e., $t|_\varepsilon = t$ and $f(t_1, \ldots, t_n)|_{ip} = t_i|_p$. The operation of replacing in a term t the subterm at position p by s is written $t[s]_p$. Two positions are *parallel* iff they are not prefixes of each other. Given an indexed set of pairwise parallel positions $P = \{p_1, \ldots, p_n\}$, we define $t[s_1, \ldots, s_n]_P$ as the parallel replacement of all subterms $t|_{p_i}$ by s_i within t. A substitution σ is a function from variables to terms, and we write $t\sigma$ for the term that is obtained from t by simultaneously replacing all variables x by $\sigma(x)$ within t. A set of term-pairs $\{s_i \approx t_i \mid i \in I\}$ is *unifiable* iff there exists some substitution σ such that $s_i\sigma = t_i\sigma$ for all $i \in I$, and in that case a most general unifier (mgu) exists.

A term t is *linear* if every variable in t occurs exactly once.

A *context* C is either a variable, or a hole $\Box$, or a function symbol f of arity n which is applied to n contexts: $f(C_1, \ldots, C_n)$. We define $\#_\Box(C)$ to be the number of holes in C. Given a context C with $\#_\Box(C) = n$ and given n terms $t_1, \ldots, t_n$, we write $C[t_1, \ldots, t_n]$ for the term which is obtained by filling all holes of C by terms t_1 to t_n when traversing the holes from left to right, e.g., if $C = f(\Box, g(x, \Box))$ then $C[y, a] = f(y, g(x, a))$. Applying a substitution on a context only substitutes the variables, and it will leave the holes in the context untouched. We sometimes write *singlehole* context to restrict to contexts with exactly one hole, or we write *multihole* context to emphasize that now the flexibility of several holes is required.

A TRS $\mathcal{R}$ is a set of ordered pairs of terms, called rules, where a rule is usually written $\ell \to r$. We call a subset of $\mathcal{R}$ a subsystem. A TRS $\mathcal{R}$ is *left-linear* if ℓ is linear for all $\ell \to r \in \mathcal{R}$. The induced rewrite relation of a TRS $\mathcal{R}$ is written as $\to_{\mathcal{R}}$ and can be defined either via positions or via contexts: $s \to_{\mathcal{R}} t$ iff there exist a rule $\ell \to r \in \mathcal{R}$ and a substitution σ such that $s|_p = \ell\sigma$ and $t = s[r\sigma]_p$ for some $p \in \mathcal{P}os(s)$ (or equivalently $s = C[\ell\sigma]$ and $t = C[r\sigma]$ for some singlehole context C). Similarly, we also define the parallel rewrite relation $\pi\!\!\to_{\mathcal{R}}$ in two equivalent ways: $s \pi\!\!\to_{\mathcal{R}} t$ iff there are n rules $\ell_1 \to r_1, \ldots, \ell_n \to r_n \in \mathcal{R}$ and substitutions $\sigma_1, \ldots, \sigma_n$ such that $s|_{p_i} = \ell_i\sigma_i$ for all $i \in \{1, \ldots, n\}$ and $t = s[r_1\sigma_1, \ldots, r_n\sigma_n]_P$ for some parallel set of positions $P = \{p_1, \ldots, p_n\} \subseteq \mathcal{P}os(s)$ (or equivalently: $s = C[\ell_1\sigma_1, \ldots, \ell_n\sigma_n]$ and $t = C[r_1\sigma_1, \ldots, r_n\sigma]$ for some multihole context C). We sometimes just write $\pi\!\!\to$ if $\mathcal{R}$ is fixed by the context, and we write $\overset{C}{\pi\!\!\to}$ or $\overset{P}{\pi\!\!\to}$ if we want to explicitly state the context C or the set of positions P that have been used for rewriting.

A relation $\to$ is *terminating* if there exists no infinite sequence $t_0 \to t_1 \to \cdots$. We say that a TRS $\mathcal{R}$ is *relatively terminating modulo* $\mathcal{S}$ if $\to_{\mathcal{R}} \cdot \to_{\mathcal{S}}^*$ is terminating. Here $\to^*$ denotes the reflexive and transitive closure of $\to$, and $\to_1 \cdot \to_2$ is defined as follows: $s \to_1 \cdot \to_2 u$ if $s \to_1 t \to_2 u$ for some t.

A relation of form $t \,{}^*\!\leftarrow s \to^* u$ is called a *peak*, while terms t and u with $t \to^* \cdot \,{}^*\!\leftarrow u$ are said to be *joinable*. A relation $\to$ is *confluent* if $t \to^* \cdot \,{}^*\!\leftarrow u$ holds for all peaks $t \,{}^*\!\leftarrow \cdot \to^* u$. Similarly, two relations $\to_1$ and $\to_2$ *commute* if $t \to_2^* \cdot \,{}_1^*\!\leftarrow u$ holds for all peaks of form $t \,{}_1^*\!\leftarrow \cdot \to_2^* u$. Confluence is a special case of commutation: $\to$ is confluent iff $\to$ and $\to$ commute.

2.3 Parallel Critical Pairs

Critical pairs are useful for investigating local peaks of two potentially diverging rewrite steps $\leftarrow \cdot \overset{\varepsilon}{\to}$, and similarly, parallel critical pairs are helpful to analyse a local peak of two parallel rewrite steps $\pi\!\!\leftarrow \cdot \overset{\varepsilon}{\to}$; in both situations, one of the steps must be a rewrite step at the root position ε. To define the notion of parallel critical pair, we introduce a few terminologies. A substitution σ is called a *renaming* if σ is a bijective function on variables. A pair (ℓ', r') of terms (or a rule $\ell' \to r'$) is a *variant* of a pair (ℓ, r) if $\ell'\sigma = \ell$ and $r'\sigma = r$ for some renaming σ. For instance, $f(a, y, z) \to g(y, y)$ is a variant of $f(a, x, y) \to g(x, x)$, where the corresponding renaming is $\{x/y, y/z, z/x\}$.

Definition 2.1 (Parallel Critical Pairs and Peaks [2, 6])**.** Let $\mathcal{R}$ be some TRS. Let $\ell_0 \to r_0 \in \mathcal{R}$. Consider any non-empty set of positions $P = \{p_1, \ldots, p_n\}$ where all these positions are

- non-variable positions of ℓ_0, i.e., $\ell_0|_{p_i} \notin \mathcal{V}$, and
- pairwise parallel.

Assume $\ell_i \to r_i$ are variants of rules in $\mathcal{R}$ for all $i \in \{1, \ldots, n\}$ and assume that σ is an mgu of the unification problem $\{\ell_i \approx \ell_0|_{p_i} \mid 1 \le i \le n\}$. In the case $P = \{\varepsilon\}$ further assume that $\ell_1 \to r_1$ is not a variant of $\ell_0 \to r_0$.

Define $s = \ell_0\sigma$, $t = \ell_0\sigma[r_1\sigma, \ldots, r_n\sigma]_P$, and $u = r_0\sigma$. Then

- $t \overset{P}{\pi\!\!\leftarrow} s \overset{\varepsilon}{\to} u$ is a *parallel critical peak* of $\mathcal{R}$, and
- (t, u) is a *parallel critical pair* of $\mathcal{R}$, also written as $t \pi\!\!\leftarrow \rtimes \overset{\varepsilon}{\to} u$.

Note that standard (non-parallel) critical pairs are just parallel critical pairs where the set of positions P in the peak is restricted to be a singleton set: $|P| = 1$.

3 Formalization

Our first aim is to formalize several confluence criteria that are based on parallel critical pairs in a proof assistant. We will first illustrate the problems that already arise when trying to formalize Definition 2.1, a problem that motivated us to design a modified formal definition of PCPs; and afterwards we describe our formalization of the various confluence criteria.

3.1 Unification with Variable Renamings

The first problem one encounters in the formal definition of PCPs w.r.t. Definition 2.1 is the renaming of variables to build critical pairs.

Consider the TRS $\mathcal{R}$ consisting of the following two rules.

$$f(x, a, y) \to g(x, y)$$
$$f(b, x, y) \to h(x, y)$$

A local peak consisting of two root steps must be captured by some PCP, e.g., the peak $g(b, g(b, b)) \leftarrow f(b, a, g(b, b)) \to h(a, g(b, b))$.

The PCPs of $\mathcal{R}$ arise from unifying the left-hand side $f(x, a, y)$ with the renamed version of the other left-hand side $f(b, x', y')$, resulting in the PCP $(g(b, y), h(a, y))$ via mgu $\{x/b, x'/a, y'/y\}$. Indeed, this critical pair captures the peak via the substitution $\{y/g(b, b)\}$. Renaming is clearly required as $f(x, a, y)$ and $f(b, x, y)$ do not unify, i.e., without renaming there would not be any PCP. However, taking Definition 2.1 literally is not desirable, as *every* renaming of variables in rules is considered. This would lead to a definition where $(g(b, z), h(a, z))$ is a PCP for every variable z.

In IsaFoR this problem (which also occurs for standard critical pairs) was previously solved by using a globally fixed renaming scheme—which was hardcoded for strings—and by using a fixed unification algorithm. As a consequence, all formal results of IsaFoR that rely upon critical pairs were restricted in a way that variables had to be strings.

Our new formalization is more generic in the sense that PCPs are defined by taking an arbitrary renaming function as parameter.[2] We just require two renaming functions

[2]We also adjusted the definition of critical pairs in IsaFoR, so that these are now parametric, too.

on variables V: *ren1* : $\mathbb{N} \times V \to V$ is used to rename each variable x of a rule $\ell_i \to r_i$ in Definition 2.1 into $ren1(i, x)$, and *ren2* : $V \to V$ renames the variables in rule $\ell_0 \to r_0$ that is applied at the root). Both *ren1* and *ren2* need to be injective and their range must be disjoint.

In Isabelle we model these two functions in a separate type that enforces the assumptions, and `rename_many` and `rename_single` are the selectors [3].

```
typedef ('v :: infinite) renamingN =
  { (ren1 :: nat × 'v ⇒ 'v, ren2 :: 'v ⇒ 'v).
  inj ren1 ∧ inj ren2 ∧ range ren1 ∩ range ren2 = {} }
lift-definition rename_many ::
  'v :: infinite renamingN ⇒ nat × 'v ⇒ 'v is fst
lift-definition rename_single ::
  'v :: infinite renamingN ⇒ 'v ⇒ 'v is snd
```

The advantage of using a dedicated type is that we get the properties of `rename_many` and `rename_single` for free, i.e., without having to carry around assumptions on injectivity in the remainder of the formalization.

There is also one disadvantage, namely we have to prove that suitable renaming functions always exist, because types must be non-empty in Isabelle/HOL. Here, we use the assumption that `'v` is an infinite type, and then the result follows by using the library on cardinalities: if $|V| = \infty$, then there is a bijection between V and $\mathbb{N} \times V \uplus V$.

We further implement `string_renameN`, an executable instance of a renaming for strings. Here, `ren1` maps pairs (n, v) to xn_v and `ren2` maps v to yv. In these definitions, x, y and _ are concrete characters, v represents some input variable name, and n is a natural number in (n, v) which is converted to a string in xn_v.

Based on the generic renaming interface, we next define a wrapper function `mgu_vd_list` which combines an existing unification algorithm with an arbitrary renaming `ren`. The important fact is that `mgu_vd_list` takes care of the renaming internally, so no explicit renaming has to be performed when defining PCPs.

The idea behind the wrapper is as follows. It takes a list of n pairs of terms for unification. But instead of returning a single unifier, it returns $n + 1$ unifiers which tell us how the terms in the pairs should be instantiated. These unifiers are produced by combining the renaming functions with the mgu that is obtained from the unification algorithm. The following two lemmas show the crucial properties of `mgu_vd_list`. Here, `Some` represents a successful result of the unification algorithm (`None` would represent that the input pairs are not unifiable), μ is a list $[\mu_0, \ldots, \mu_n]$ of substitutions with the same length as the input pairs, and in the completeness result σ is a function from natural numbers i to substitutions σ_i. Therefore, μ_i and σ_i are obtained by `μ ! i` and `σ i`, respectively. Application $t\sigma$ of a substitution is denoted by $t \cdot \sigma$ within Isabelle code.

```
lemma mgu_vd_list_sound:
 assumes mgu_vd_list ren pairs = Some (μ, τ)
 and i < length pairs
shows fst (pairs ! i) · (μ ! i) = snd (pairs ! i) · τ

lemma mgu_vd_list_complete:
 assumes ∀ i < length pairs.
    fst (pairs ! i) · σ i = snd (pairs ! i) · θ
shows ∃ μ τ δ. mgu_vd_list ren pairs = Some (μ, τ) ∧
    θ = τ ∘ δ ∧ ∀ i < length pairs. σ i = (μ ! i) ∘ δ
```

So the soundness lemma claims that the pair (μ, τ) represents a unifier, while the completeness lemma claims that it is even an mgu. Note that the completeness lemma in IsaFoR actually contains some more properties of the substitutions, a fact that we will explain later on.

3.2 Formal Definition of Critical Pairs via Contexts

After having solved the problem of renamings, let us now tackle the other problem in the definition of PCPs: Sets P of parallel positions need to be managed in a way that it is always ensured that the positions in P are indeed parallel and moreover are valid positions of certain terms.

Making all these conditions explicit is cumbersome at least, and therefore, we will provide a new definition of parallel critical pairs that is based on contexts (in the same spirit as one can define $\twoheadrightarrow$ via positions or via contexts.) Contexts automatically by construction have their holes at parallel positions and there is only a minimal side-condition when working with contexts: the number of holes must coincide with the number of terms that are used to fill the holes. As a result, all our proofs in this paper can be performed by a simple structural induction on contexts with low administrative overhead.

We first provide a formal definition of PCPs between a TRS $\mathcal{R}$ for performing the parallel step, and a single rule $\ell_0 \to r_0$ that performs the root step. There is no condition that $\ell_0 \to r_0$ belongs to $\mathcal{R}$; it can be a rule from a different TRS. Later on we then define PCPs between two TRSs $\mathcal{R}$ and $\mathcal{S}$ where $\ell_0 \to r_0$ can be taken arbitrarily from $\mathcal{S}$. We start with a definition in plain text for a better comparison to Definition 2.1. In the definition, we assume that *mgu_vd_list* is a unification algorithm with renaming integrated, i.e., it satisfies the following two properties.

- soundness: if *mgu_vd_list* $[(s_1, t_1), \ldots, (s_n, t_n)]$ results in $([\mu_1, \ldots, \mu_n], \tau)$ then $s_i\mu_i = t_i\tau$ for all $i \in \{1, \ldots, n\}$.
- completeness: if $s_1\sigma_1 = t_1\theta$ and … and $s_n\sigma_n = t_n\theta$, then there are substitutions $\mu_1, \ldots, \mu_n, \tau, \delta$ such that *mgu_vd_list* $[(s_1, t_1), \ldots, (s_n, t_n)] = ([\mu_1, \ldots, \mu_n], \tau)$, $\theta = \tau \circ \delta$, and $\sigma_i = \mu_i \circ \delta$ for all $i \in \{1, \ldots, n\}$.

[3] In proof assistants that do not support such a type-definition, one could write a predicate `is_renaming_function` instead, that encodes the desired properties of the renaming functions. In this way, one can still achieve the flexibility of using arbitrary renaming functions, though at the cost of having to add the assumption `is_renaming_function (ren1, ren2)` to several statements.

Definition 3.1 (Parallel Critical Pairs and Peaks using Contexts). Let $\mathcal{R}$ be a TRS and $\ell_0 \to r_0$ a rule. We define the set of parallel critical pairs and peaks of $\mathcal{R}$ and $\ell_0 \to r_0$ as follows.

Consider any decomposition $\ell_0 = C[\ell_{0,1}, \ldots, \ell_{0,n}]$ where the hole-positions of C are non-variable positions of ℓ_0 and $n \neq 0$. Further let $\ell_i \to r_i \in \mathcal{R}$ for all $1 \leq i \leq n$ and assume $mgu\_vd\_list\ [(\ell_1, \ell_{0,1}), \ldots, (\ell_n, \ell_{0,n})] = ([\sigma_1, \ldots, \sigma_n], \tau)$.

Then we define $t = C\tau[r_1\sigma_1, \ldots, r_n\sigma_n]$, $s = \ell_0\tau = C\tau[\ell_1\sigma_1, \ldots, \ell_n\sigma_n]$, and $u = r_0\tau$ in order to construct

- $t \ {}_{\{\ell_i \to r_i \mid i \in \{1,\ldots,n\}\}}\!\overset{C\tau}{\longleftarrow\!\!\shortmid\!\!\shortmid} s \overset{\varepsilon}{\to} u$ as a *parallel critical peak* of $\mathcal{R}$ and $\ell_0 \to r_0$, and
- (t, u) as a *parallel critical pair* of $\mathcal{R}$ and $\ell_0 \to r_0$, also written as $t \overset{\varepsilon}{\longleftarrow\!\!\shortmid\!\!\shortmid\!\!\rtimes\!\!\to} u$.

Let us briefly compare Definition 3.1 with Definition 2.1 to see their close relationship: In Definition 3.1 a context C with n holes is constructed and the hole-positions of C are exactly the n positions that are used in Definition 2.1. In both definitions renamings take place, however in Definition 3.1 these are obtained via *mgu_vd_list* (they are included in τ and the σ_i's), whereas in Definition 2.1 the renamings are performed explicitly.

There are however also some differences. Of course there is the difference of how parallel positions are encoded, either via contexts or via sets of positions; but there also is a difference between the considered rules: in Definition 2.1, $\ell_0 \to r_0$ is a rule of $\mathcal{R}$ and for $P = \{\varepsilon\}$ there is a special case where $\ell_0 \to r_0$ is the same rule as $\ell_1 \to r_1$; in contrast in Definition 3.1, there is no such special case and membership condition $\ell_0 \to r_0 \in \mathcal{R}$ is not required. Whereas the encoding of positions via contexts will simplify the formal treatment of the definition in Isabelle, the other difference admits a better applicability. In particular dropping condition $\ell_0 \to r_0 \in \mathcal{R}$ will be essential to obtain commutation results and dropping the special case is required for supporting TRSs with extra variables, i.e., where variables in right-hand sides of rules need not occur in left-hand sides.

To illustrate how naturally Definition 3.1 can be encoded in a proof assistant, we here also provide the full Isabelle definition, where `ren` is some renaming function that is fixed in a local context, and where *mgu_vd_list* of Definition 3.1 is instantiated by `mgu_vd_list ren`. After this definition we no longer distinguish between *mgu_vd_list* and `mgu_vd_list ren`.

```
definition parallel_critical_peaks_of_rule R (l0,r0) =
 { (C · τ, (C · τ)[map2 (λ (li,ri) σi. ri · σi) rls σ],
   l0 τ, r0 τ, rls) | C lps rls σ τ.
   l0 = C [ lps ] ∧ hole_poss C ⊆ fun_poss l ∧
   mgu_vd_list ren
      (map2 (λ (li,ri) l0i. (li, l0i)) rls lps)
      = Some (σ, τ) ∧
   #□C ≠ 0 ∧ set rls ⊆ R ∧ length rls = #□C}
```

In the definition we use the notation `C[ts]` for filling a context `C` with a list of terms `ts`. The five-tuples that are produced in the Isabelle definition contain the entries $(C\tau, t, s, u, [\ell_1 \to r_1, \ldots, \ell_n \to r_n])$ of a parallel critical peak. Further, `lps` is the list $[\ell_{0,1}, \ldots, \ell_{0,n}]$ of Definition 3.1 and `rls` is the list $[\ell_1 \to r_1, \ldots, \ell_n \to r_n]$. Interestingly, there is only one side-condition which we have to make explicit and which one would take for granted when reasoning more informally: it is the last condition `length rls = #□C`. The remainder is a one-to-one translation of Definition 3.1, demonstrating that indeed our new notion of PCPs is suitable for mechanized proofs. In particular, multihole contexts are easily defined as an algebraic datatype, and there is no demand to formally define the notion of sets of pairwise parallel positions.

Note that once PCPs[4] between $\mathcal{R}$ and a rule $\ell \to r$ have been defined, it is easy to extend these notions to two TRSs. The set of PCPs of $\mathcal{R}$ and $\mathcal{S}$ is the union of all PCPs between $\mathcal{R}$ and any rule of $\mathcal{S}$, and we write $\mathsf{PCP}(\mathcal{R}, \mathcal{S})$ to indicate this set. Note that $\mathsf{PCP}(\mathcal{R}, \mathcal{S}) = \mathsf{PCP}(\mathcal{S}, \mathcal{R})$ is often violated. For instance, taking $\mathcal{R} = \{f(a, b) \to c\}$ and $\mathcal{S} = \{a \to a', b \to b'\}$ yields $\mathsf{PCP}(\mathcal{R}, \mathcal{S}) = \emptyset$, whereas $\mathsf{PCP}(\mathcal{S}, \mathcal{R})$ contains three elements: $(f(a', b), c)$, $(f(a, b'), c)$, and $(f(a', b'), c)$. We define the PCPs of a single TRS $\mathcal{R}$ as $\mathsf{PCP}(\mathcal{R}) := \mathsf{PCP}(\mathcal{R}, \mathcal{R})$. We further write $\mathsf{CP}(\mathcal{R}, \mathcal{S})$ for the standard critical pairs of $\mathcal{R}$ and $\mathcal{S}$, i.e., those elements of $\mathsf{PCP}(\mathcal{R}, \mathcal{S})$ where the number of holes in the context is exactly one.

3.3 The Key Lemma

After having defined PCPs we are now ready to prove the key lemma of PCPs. It states that a peak arising from a parallel step of $\mathcal{R}$ and a root step with rule $\ell \to r$ can be captured by a PCP of $\mathcal{R}$ and $\ell \to r$.

We will first state the lemma, then discuss its relationship to similar lemmas in the literature, and afterwards briefly discuss its formal proof.

Lemma 3.2 (Capturing a Root-Overlap). *Let $\ell \to r$ be a left-linear rule and let $\mathcal{R}'$ and $\mathcal{R}$ be two TRSs with $\mathcal{R}' \subseteq \mathcal{R}$. Consider a peak of the form $t \ {}_{\mathcal{R}'}\!\overset{C}{\longleftarrow\!\!\shortmid\!\!\shortmid} s \overset{\varepsilon}{\to}_{\{\ell \to r\}} u$. Then one of the following two statements is satisfied.*

1. *There exists a term v such that $t \overset{\varepsilon}{\to}_{\{\ell \to r\}} v \ {}_{\mathcal{R}'}\!\longleftarrow\!\!\shortmid\!\!\shortmid u$.*
2. *There exists a parallel critical peak $t' \ {}_{\mathcal{R}''}\!\overset{C'}{\longleftarrow\!\!\shortmid\!\!\shortmid} s' \overset{\varepsilon}{\to}_{\{\ell \to r\}} u'$ of $\mathcal{R}$ and $\ell \to r$ with $\mathcal{R}'' \subseteq \mathcal{R}'$ and there are substitutions δ and γ such that $t = t'\gamma$, $s = s'\delta$, $u = u'\delta$, $\delta(x) \twoheadrightarrow_{\mathcal{R}'} \gamma(x)$ for all x, and $\delta(x) = \gamma(x)$ for all $x \notin \mathcal{V}\mathrm{ar}(C')$.*

One usually finds more specific versions of this lemma in the literature, e.g., often $\mathcal{R}' = \mathcal{R}$, sometimes the condition $\delta(x) = \gamma(x)$ for all $x \notin \mathcal{V}\mathrm{ar}(C')$ is omitted, etc., cf. [33,

[4] In the sequel we use the abbreviation PCP for both parallel critical peaks and pairs. The peaks contain the full information, and the pairs are just peaks where some data has been discarded.

Lemma 55] or [23, Lemma 16]. However, it will be essential to have all these information and flexibility to support a variety of confluence and commutation techniques. For instance, for rule labeling one often has to consider a case with $\mathcal{R}' \subset \mathcal{R}$ and one further needs to know $\mathcal{R}'' \subseteq \mathcal{R}'$. And for supporting Toyama's condition, the $\delta(x) = \gamma(x)$ property is essential.

The structure of the proof of the lemma is well-known from the literature [5, 24]. One performs a case-analysis on whether the parallel step is completely inside the substitution of the root-step (then the first statement is satisfied), or otherwise there is some overlap and a PCP can be obtained. In the latter case the most difficult part is to achieve the equality $\delta(x) = \gamma(x)$ for $x \notin \mathcal{V}\mathrm{ar}(C')$.

We refer to the formalization for all the technical details, and here briefly mention one problem. In papers it is often assumed that the renaming introduces fresh variables without giving the precise information of what should be fresh. Also in our formalization, *mgu_vd_list* takes as input just the unification problem without any reference of variable-names that must be avoided. Hence, it can easily happen that *mgu_vd_list* renames a variable into a "fresh" one, which actually occurs somewhere else in the proof as part of a term or context that is not contributing to the unification problem.

Example 3.3. Consider the rule $f(g(x_1, a), y) \to h(x_1, y)$ as $\ell \to r$ and let $\mathcal{R} = \{g(x_1, x_2) \to k(x_1, x_2)\}$. For building a PCP between them we choose the context $C = f(\Box, y)$ and invoke *mgu_vd_list* $[(g(x_1, x_2), g(x_1, a))]$. The renaming might internally switch from $g(x_1, a)$ to $g(z, a)$ and we obtain the desired PCP $f(k(z, a), y) \leftarrow\!\!\!\!\shortmid\!\!\!\!\shortmid f(g(z, a), y) \xrightarrow{\varepsilon} h(z, y)$.

However, *mgu_vd_list* $[(g(x_1, x_2), g(x_1, a))]$ might also decide to internally switch from $g(x_1, a)$ to $g(y, a)$, since y does not occur in the unification problem. Consequently, then just taking the unifier $\{x_1/y, x_2/a\}$ would result in $f(k(y, a), y) \leftarrow\!\!\!\!\shortmid\!\!\!\!\shortmid f(g(y, a), y) \xrightarrow{\varepsilon} h(y, y)$. This is not the desired PCP as the previously different variables y and z now became identical. The cause is a clash of the variable y from the unifier with the variable y that appeared in the context $C = f(\Box, y)$. This problem is resolved by the fact that in the definition of PCPs, τ is also applied on C, and τ has to rename y to some other variable, e.g., y', so the PCP would be $f(k(y, a), y') \leftarrow\!\!\!\!\shortmid\!\!\!\!\shortmid f(g(y, a), y') \xrightarrow{\varepsilon} h(y, y')$.

That the renaming in *mgu_vd_list* always results in the intended PCPs is something where we need to know the behavior of the resulting substitutions of *mgu_vd_list* on variables that do not occur in the unification problem (in the example $\tau(y) \neq y$ had to be ensured), and such a property is exactly what is not yet listed in lemma `mgu_vd_list_complete`. So, here is what we additionally proved for *mgu_vd_list*.

Lemma 3.4 (Extended completeness result of *mgu_vd_list*). *Let $[(s_1, t_1), \ldots, (s_n, t_n)]$ be an input for mgu_vd_list where unification succeeds with resulting substitution list μ and substitution τ. Let $V = \bigcup_{1 \le i \le n} \mathcal{V}\mathrm{ar}(t_i)$ and let W be the complement of V. Then all of the following facts are satisfied.*

- *τ is injective on W,*
- *if $x \in W$ then $\tau(x)$ is a variable, and*
- *if $x \in W$ then $\tau(x)$ does neither occur in $\mathcal{V}\mathrm{ar}(t_i\tau)$ nor in $\mathcal{V}\mathrm{ar}((\mu\,!\,i)(y))$ for any i and y.*

The full completeness lemma can actually be derived from lemma `mgu_vd_list_complete`, but the proof is not immediate: we need 120 lines of Isabelle, and apply `mgu_vd_list_complete` on four different combinations of substitutions.

Using these extended results on *mgu_vd_list*, we are finally able to formally prove Lemma 3.2, but still this is a major effort. For instance, six auxiliary substitutions are defined or obtained in the proof, before being able to define the final desired substitutions δ and γ.

- σ is the substitution of the root step
- τ is the substitution that is obtained from σ by performing those parallel rewrites that happen within σ; τ exists since ℓ is linear
- σ_i are the substitutions for the parallel step that overlaps with the root step
- σ_i', τ' and δ' are the substitutions obtained from the completeness result of *mgu_vd_list*
- δ is a merge of substitution σ and δ' that also inverts the renaming that might have happened on variables that do not appear in the unification problem
- γ merges δ and δ' and deals with extra variables that might occur in r

The overall formal proof of Lemma 3.2 is about 400 lines, not including auxiliary lemmas. It is thereby the most complex part of our formalization. In comparison, the formalization of all upcoming confluence and commutation techniques in this section has a size of just 1500 lines.

3.4 Gramlich's and Toyama's Confluence Criteria

The most immediate consequences of Lemma 3.2 are the PCP-based criteria of Gramlich and Toyama to ensure confluence. Their formalization is mostly straight-forward and it turned out that both criteria easily extend to a commutation version that fully covers the confluence case.

Theorem 3.5 (Gramlich's confluence criterion [6], generalized to commutation). *Two left-linear TRSs $\mathcal{R}$ and $\mathcal{S}$ commute if*

- *$t \to_{\mathcal{S}}^* u$ for all $t \; {}_{\mathcal{R}}\!\overset{C}{\leftarrow\!\!\!\!\shortmid\!\!\!\!\shortmid} \rtimes \xrightarrow{\varepsilon}_{\mathcal{S}} u \in \mathrm{PCP}(\mathcal{R}, \mathcal{S})$ with $C \neq \Box$, and*
- *$t \twoheadrightarrow_{\mathcal{R}} \cdot \; {}_{\mathcal{S}}^*\!\leftarrow u$ for all $t \; {}_{\mathcal{S}}\!\leftarrow \rtimes \xrightarrow{\varepsilon}_{\mathcal{R}} u \in \mathrm{CP}(\mathcal{S}, \mathcal{R})$.*

The formalization contains this theorem, but not its original proof. Instead we formalize Toyama's criterion that fully covers Gramlich's one.

Theorem 3.6 (Toyama's confluence criterion [28], generalized to commutation). *Two left-linear TRSs $\mathcal{R}$ and $\mathcal{S}$ commute if*

- $t \to_{\mathcal{S}}^* \cdot {}_{\mathcal{R}}\xleftarrow{\mathcal{V}\mathrm{ar}(C)} u$ *for all* $t\ {}_{\mathcal{R}}\xleftarrow{C} \cdot \xrightarrow{\varepsilon}_{\mathcal{S}} u \in \mathrm{PCP}(\mathcal{R},\mathcal{S})$ *with* $C \neq \Box$, *and*
- $t \twoheadrightarrow_{\mathcal{R}} \cdot {}_{\mathcal{S}}^*\!\leftarrow u$ *for all* $t\ {}_{\mathcal{S}}\!\leftarrow \rtimes \xrightarrow{\varepsilon}_{\mathcal{R}} u \in \mathrm{CP}(\mathcal{S},\mathcal{R})$.

The major advantage of Toyama's criterion is the additional parallel step that is allowed to join PCPs. However, here a variable restricted version of parallel rewriting is considered where $\mathcal{V}\mathrm{ar}(C)$ is a set of forbidden variables. We formalize it—for arbitrary sets of variables V—as $s \xrightarrow{V} t$ if there is a parallel step $s = C[\ell_1\sigma_1,\dots,\ell_n\sigma_n] \twoheadrightarrow C[r_1\sigma_1,\dots,r_n\sigma_n] = t$ such that $\mathcal{V}\mathrm{ar}(r_i\sigma_i) \cap V = \emptyset$ for all $1 \le i \le n$.

Example 3.7. Consider the TRSs $\mathcal{R} = \{a \to b, g(x,y) \to c\}$ and $\mathcal{S} = \{b \to c, h(x,c) \to c, h(x,a) \to g(x,y)\}$. There are no CPs of $\mathcal{S}$ and $\mathcal{R}$ and there is one PCP of $\mathcal{R}$ and $\mathcal{S}$:

$$h(x,b)\ {}_{\mathcal{R}}\xleftarrow{h(x,\Box)} h(x,a) \xrightarrow{\varepsilon}_{\mathcal{S}} g(x,y).$$

The PCP can be joined in the desired way

$$h(x,b) \to_{\mathcal{S}} h(x,c) \to_{\mathcal{S}} c\ {}_{\mathcal{R}}\xleftarrow{\mathcal{V}\mathrm{ar}(h(x,\Box))} g(x,y)$$

since $\mathcal{V}\mathrm{ar}(c) \cap \mathcal{V}\mathrm{ar}(h(x,\Box)) = \emptyset$. Therefore by Theorem 3.6 we conclude that $\mathcal{R}$ and $\mathcal{S}$ commute. Note that it is allowed that $\mathcal{S}$ contains an extra-variable y in the right-hand side of the last rule.

Our formalized proof of Theorem 3.6 closely follows Gramlich's proof of Theorem 3.5, where we add additional reasoning steps to support the stronger joining condition of Toyama. Gramlich considers an arbitrary peak between a rewrite step and a parallel rewrite step. Then he proves by induction on the contexts of these steps that every such peak is joinable in a specific way, in particular only one parallel step is allowed in the join.

Since our definition of PCPs is context based, it is actually easy to exactly follow the original proof structure in Isabelle. We just compose and decompose contexts that occur in parallel rewrite steps, in PCPs, etc., with little formalization overhead.

The only remaining interesting aspect in this part of the formalization is the integration of Toyama's condition. At this point, we prove a crucial property about the variable-restricted parallel rewrite relation. It states that steps in a substitution and a parallel step can be merged into a single parallel step under certain assumptions.

Lemma 3.8 (). *Let $\delta(x) \twoheadrightarrow \gamma(x)$ for all x, let $\delta(x) = \gamma(x)$ for all $x \notin V$. Then $s \xrightarrow{V} t$ implies $s\delta \twoheadrightarrow t\gamma$.*

Lemma 3.8 provides a way to merge the parallel step from δ to γ in Lemma 3.2 with the parallel step from joining a PCP in Theorem 3.6 into a single parallel step. This merge is required in the soundness proof of Theorem 3.6, and this merge is also the part of the proof that was the most challenging one: It forced us to add the condition $\delta(x) = \gamma(x)$ for certain x in Lemma 3.2, i.e., the condition that was difficult to prove.

Note that once Theorem 3.6 has been formalized, one can also derive the (*almost*) *parallel closedness theorem* as a corollary, cf. [23]. However, we did not formalize this subsumption relation, since the parallel closedness theorem was already available in IsaFoR [20].

3.5 Formalizing PCP-based Rule Labeling Techniques

The *decreasing diagrams* [31] technique is a powerful method for proving confluence and commutation of TRSs. Many well-known confluence criteria including Knuth and Bendix' criterion [12], Huet's *parallel closedness* and *strong closedness* [10], and Toyama's confluence criterion based on PCPs [28] can be proven by using decreasing diagrams. *Rule labeling* [31, 33] is a direct application of using decreasing diagrams for confluence and commutation proofs for TRSs.

Both decreasing diagrams and rule labeling are already part of IsaFoR [19]. However, IsaFoR did not cover those results that combine rule labeling with PCPs [23, 33].

Definition 3.9 (cf. [23]). Let $\mathcal{R}$ and $\mathcal{S}$ be left-linear TRSs. A *labeling function* ϕ for $\mathcal{R}$ (resp. ψ for $\mathcal{S}$) is a function from $\mathcal{R}$ (resp. $\mathcal{S}$) to $\mathbb{N}$. Given a labeling function ϕ for $\mathcal{R}$ (resp. ψ for $\mathcal{S}$) and a number $k \in \mathbb{N}$ (resp. $m \in \mathbb{N}$), we define the TRS $\mathcal{R}_{\phi,k}$ and $\mathcal{S}_{\psi,m}$ as follows:

$$\mathcal{R}_{\phi,k} = \{\ell \to r \in \mathcal{R} \mid \phi(\ell \to r) \le k\}$$
$$\mathcal{S}_{\psi,m} = \{\ell \to r \in \mathcal{S} \mid \psi(\ell \to r) \le m\}$$

We say that a local peak $t \xleftarrow{C} \rtimes \xrightarrow{\varepsilon} u$ is $(\psi,\phi,\mathcal{S},\mathcal{R},k,m)$-*decreasing* if

$$t \xleftarrow[\curlyvee k]{}{}^* \cdot \xrightarrow[\mathcal{S}_{\psi,m}]{} \cdot \xleftarrow[\curlyvee km]{}{}^* \cdot \xleftarrow[\mathcal{R}_{\phi,k}]{\mathcal{V}\mathrm{ar}(C)} \cdot \xleftarrow[\curlyvee m]{}{}^* u.$$

Here, $\xleftrightarrow[\curlyvee k]{}$ stands for $\bigcup_{k'<k}(\mathcal{R}_{\phi,k'}\leftarrow \cup \to_{\mathcal{S}_{\psi,k'}})$ and similarly, $\xleftrightarrow[\curlyvee m]{}$ stands for $\bigcup_{m'<m}(\mathcal{R}_{\phi,m'}\leftarrow \cup \to_{\mathcal{S}_{\psi,m'}})$. Also, $\xleftrightarrow[\curlyvee km]{}$ stands for $\xleftrightarrow[\curlyvee k]{} \cup \xleftrightarrow[\curlyvee m]{}$.

The following theorem generalizes Theorem 31 in [23] to its commutation version, where Theorem 31 in [23] subsumes the earlier confluence criterion of Theorem 56 in [33] based on rule labeling and PCPs.

Theorem 3.10 (Shintani and Hirokawa's rule labeling criterion with PCPs, generalized to commutation). *Let $\mathcal{R}$ and $\mathcal{S}$ be left-linear TRSs, and ϕ and ψ the labeling functions for $\mathcal{R}$ and $\mathcal{S}$, respectively. Suppose that $\mathcal{R}_{\phi,0}$ and $\mathcal{S}_{\psi,0}$ commute. The TRSs $\mathcal{R}$ and $\mathcal{S}$ commute if the following conditions hold for all $(k,m) \in \mathbb{N}^2 \setminus \{(0,0)\}$.*

1. *Every parallel critical peak of form $t \xleftarrow[\mathcal{R}_{\phi,k}]{C} s \xrightarrow[\mathcal{S}_{\psi,m}]{\varepsilon} u$ is $(\psi,\phi,\mathcal{S},\mathcal{R},k,m)$-decreasing.*

2. *Every parallel critical peak of form* $t \overset{D}{\underset{\mathcal{S}_{\psi,m}}{\longleftarrow\!\!\shortparallel}} s \xrightarrow[\mathcal{R}_{\phi,k}]{\varepsilon} u$ *is* $(\phi, \psi, \mathcal{R}, \mathcal{S}, m, k)$*-decreasing.*

Again, we formalize Theorem 3.10 by closely following the original proofs. This formalization is feasible since many underlying results are available, in particular Lemma 3.2 and existing results on decreasing diagrams.

Since the original proofs are quite detailed already, we highlight one deviation from the original proof. It reveals a small gap that luckily can easily be fixed.

In order to apply Theorem 3.10 one obviously cannot consider all natural numbers k and m. To solve this problem, one usually just computes k and m implicitly by computing the labels from a PCP (in Isabelle we defined these labeled peaks as `critical_parallel_rule_peaks`) and just demands that these peaks are joinable in Theorem 3.10. So, given a PCP $t \underset{\{\ell_1 \to r_1, \ldots, \ell_n \to r_n\}}{\longleftarrow\!\!\shortparallel} s \xrightarrow[\ell_0 \to r_0]{\varepsilon} u$, one computes $k' = \max\{\phi(\ell_i \to r_i) \mid 1 \le i \le n\}$ and $m' = \psi(\ell_0 \to r_0)$.

It can be argued that it suffices to consider the computed values k' and m': Consider any peak with numbers k and m. Then $m \ge m'$ and $k \ge k'$, because otherwise the applied rules would not be contained in $\mathcal{R}_{\phi,k}$ and $\mathcal{S}_{\psi,m}$. For checking the decreasing conditions it therefore is sufficient to just check that the PCP is decreasing w.r.t. k' and m', since any join w.r.t. k' and m' implies a join w.r.t. k and m, as the increase of k' to k and m' to m can just add more rules to the TRSs that are used in the join, e.g., $\mathcal{R}_{\phi,k'} \subseteq \mathcal{R}_{\phi,k}$.

Until now everything seems fine, however there is one gap that was revealed while performing the formalization. Theorem 3.10 in particular excludes the case $k = m = 0$. So, the whole point of this exclusion is that one does not have to care about PCPs where $k' = m' = 0$. Based on the idea, the original proof [23, Proof of Theorem 31] assumed that a peak comprising such a PCP is always labeled 0; in other words, it assumed $k = m = 0$. However, according to the original theorem one still has to consider e.g., $k = 1$ and $m = 0$ (or $k = 0$ and $m = 1$) and show that the PCP with $k' = m' = 0$ is $(\psi, \phi, \mathcal{S}, \mathcal{R}, k, m)$-decreasing. Fortunately, it can be shown that such a peak can always be closed in a decreasing way, i.e., even if one disregards PCPs with $k' = m' = 0$. This case was not covered in the original proof.

For completeness, we here provide the formalized statement of Theorem 3.10 with the correction of computed labeling numbers. In the theorem, the computed numbers k' and m' above appear as k and m, respectively.

Theorem 3.11 (Shintani and Hirokawa's rule labeling criterion with PCPs, generalized to commutation, with exclusion of PCPs with label 0). *Theorem 3.10 is still correct if one replaces the decreasing conditions by the following ones.*

- *Whenever* $t \underset{\{\ell_1 \to r_1, \ldots, \ell_n \to r_n\}}{\longleftarrow\!\!\shortparallel} s \xrightarrow[\ell_0 \to r_0]{\varepsilon} u \in \mathsf{PCP}(\mathcal{R}, \mathcal{S})$, $k = \max\{\phi(\ell_i \to r_i) \mid 1 \le i \le n\}$ *and* $m = \psi(\ell_0 \to r_0)$ *and* $(k, m) \neq (0, 0)$*, then the peak* $t \longleftarrow\!\!\shortparallel s \xrightarrow{\varepsilon} u$ *is* $(\psi, \phi, \mathcal{S}, \mathcal{R}, k, m)$*-decreasing.*
- *(Symmetric version where the roles of* $\mathcal{R}$ *and* $\mathcal{S}$ *are swapped, as well as* ϕ *and* ψ*.)*

Note that in this part of the formalization it was crucial to add support for subsystems in Lemma 3.2 ($\mathcal{R}' := \mathcal{R}_{\phi,k} \subseteq \mathcal{R}$) and it was further important that PCPs include the information which rules contribute to forming a PCP.

3.6 Formalizing Commutation Techniques Based on Parallel Critical Pair Systems

The last criterion that we cover in our formalization is based on critical pair systems [7]. We consider the stronger compositional version of critical pair systems [23].

Definition 3.12 (PCPS for confluence [23, Definition 34]). Let $\mathcal{R}$ and $\mathcal{C}$ be TRSs. The parallel critical pair system $\mathsf{PCPS}(\mathcal{R}, \mathcal{C})$ of $\mathcal{R}$ *modulo* $\mathcal{C}$ is defined as the TRS:

$$\{s \to t, s \to u \mid t \longleftarrow\!\!\shortparallel s \xrightarrow{\varepsilon} u \in \mathsf{PCP}(\mathcal{R}) \wedge \neg(t \leftrightarrow_{\mathcal{C}}^* u)\}.$$

Theorem 3.13 (Compositional confluence criterion based on PCPS [23, Theorem 38]). *Let* $\mathcal{R}$ *be a left-linear TRS and* $\mathcal{C}$ *a confluent TRS with* $\mathcal{C} \subseteq \mathcal{R}$*. The TRS* $\mathcal{R}$ *is confluent if* $t \to_{\mathcal{R}}^* \cdot {}_{\mathcal{R}}^*\!\!\leftarrow u$ *for all* $t \; {}_{\mathcal{R}}\!\!\longleftarrow\!\!\shortparallel \rtimes \xrightarrow{\varepsilon}_{\mathcal{R}} u \in \mathsf{PCP}(\mathcal{R})$ *and* $\mathsf{PCPS}(\mathcal{R}, \mathcal{C})$ *is relatively terminating modulo* $\mathcal{R}$*.*

Formalizing Theorem 3.13 is not a big challenge at this point: the original proofs are quite detailed; and they are using decreasing diagrams in a similar way as the criteria based on rule labeling, so that many parts of the formalization in Section 3.5 can just be reused.

The actual challenging part was on the question how to generalize this criterion to commutation, a task that did not pose major problems for the criteria of the previous sections. For instance, does one have to demand $\mathcal{C} \subseteq \mathcal{R} \cap \mathcal{S}$, or can one take two TRSs $\mathcal{C} \subseteq \mathcal{R}$ and $\mathcal{D} \subseteq \mathcal{S}$. In the latter case, what is the proper way to generalize $\leftrightarrow_{\mathcal{C}}^*$, e.g., ${}_{\mathcal{C}}^*\!\!\leftarrow \cdot \to_{\mathcal{D}}^*$? After several variations, we came up with following definition and formalize the upcoming theorem in Isabelle.

Definition 3.14 (PCPS for commutation). Let $\mathcal{R}$, $\mathcal{S}$, $\mathcal{C}$, and $\mathcal{D}$ be TRSs. The commutation version of the parallel critical pair system $\mathsf{PCPS}(\mathcal{R}, \mathcal{S}, \mathcal{C}, \mathcal{D})$ is defined as the TRS:

$$\{s \to t, s \to u \mid t \longleftarrow\!\!\shortparallel s \xrightarrow{\varepsilon} u \in \mathsf{PCP}(\mathcal{R}, \mathcal{S}) \wedge \neg(t \to_{\mathcal{D} \cup \mathcal{C}^{-1}}^* u)\}.$$

Note that $\mathsf{PCPS}(\mathcal{R}, \mathcal{C})$ in Definition 3.12 is an instance of $\mathsf{PCPS}(\mathcal{R}, \mathcal{S}, \mathcal{C}, \mathcal{D})$ in Definition 3.14 when $\mathcal{S} = \mathcal{R}$ and $\mathcal{D} = \mathcal{C}$.

The following lemma is the formalized commutation version of Theorem 38 in [23]. It is easy to see that it subsumes the original theorem by choosing $\mathcal{S} = \mathcal{R}$ and $\mathcal{D} = \mathcal{C}$.

Theorem 3.15 (Compositional commutation criterion based on PCPS). *Let* $\mathcal{R}$ *and* $\mathcal{S}$ *be left-linear TRSs, and* $\mathcal{C}$ *and* $\mathcal{D}$ *be commuting TRSs with* $\mathcal{C} \subseteq \mathcal{R}$ *and* $\mathcal{D} \subseteq \mathcal{S}$*. The TRSs* $\mathcal{R}$ *and* $\mathcal{S}$ *commute if*

- $t \to_{\mathcal{S}}^* \cdot {}_{\mathcal{R}}^*\!\leftarrow u$ *for all* $t\ {}_{\mathcal{R}}\!\leftarrow\!\!\shortmid\!\!\rightarrow \overset{\varepsilon}{\to}_{\mathcal{S}} u \in \mathsf{PCP}(\mathcal{R}, \mathcal{S})$,
- $t \to_{\mathcal{R}}^* \cdot {}_{\mathcal{S}}^*\!\leftarrow u$ *for all* $t\ {}_{\mathcal{S}}\!\leftarrow\!\!\shortmid\!\!\rightarrow \overset{\varepsilon}{\to}_{\mathcal{R}} u \in \mathsf{PCP}(\mathcal{S}, \mathcal{R})$,

and $\mathsf{PCPS}(\mathcal{R}, \mathcal{S}, \mathcal{C}, \mathcal{D}) \cup \mathsf{PCPS}(\mathcal{S}, \mathcal{R}, \mathcal{D}, \mathcal{C})$ *is relatively terminating modulo* $\mathcal{R} \cup \mathcal{S}$.

We illustrate the power of our new compositional commutation criterion in Theorem 3.15 in the upcoming Example 3.16. Using this theorem, commutation can easily be proven, but the example cannot be solved by any participant of the Confluence Competition 2023.

Example 3.16. Consider the following rewrite rules that include two different implementations of the Fibonacci function, a recursive one and an iterative one.

$$\mathit{start} \to \mathit{fibRec}(s(s(s(0)))) \tag{1}$$
$$\mathit{start} \to \mathit{fibIter}(s(s(s(0)))) \tag{2}$$
$$\mathit{fibRec}(0) \to 0 \tag{3}$$
$$\mathit{fibRec}(s(0)) \to s(0) \tag{4}$$
$$\mathit{fibRec}(s(s(x))) \to \mathit{add}(\mathit{fibRec}(s(x)), \mathit{fibRec}(x)) \tag{5}$$
$$\mathit{fibIter}(x) \to \mathit{fibIterMain}(0, s(0), x) \tag{6}$$
$$\mathit{fibIterMain}(x, y, 0) \to x \tag{7}$$
$$\mathit{fibIterMain}(x, y, s(z)) \to \mathit{fibIterMain}(y, \mathit{add}(x, y), z) \tag{8}$$
$$\mathit{add}(0, y) \to y \tag{9}$$
$$\mathit{add}(s(x), y) \to s(\mathit{add}(x, y)) \tag{10}$$
$$\mathit{add}(x, y) \to \mathit{add}(y, x) \tag{11}$$

We define $\mathcal{R} = \{(1), (3) - (5), (9) - (11)\}$ and $\mathcal{S} = \{(2), (6) - (8), (9) - (11)\}$, i.e., both TRSs share the implementation of addition and contain exactly one of the two implementations of the Fibonacci function.

By applying Theorem 3.15 we can first remove rules (1) and (2) by choosing $\mathcal{C} = \{(3) - (5), (9) - (11)\}$ and $\mathcal{D} = \{(6) - (8), (9) - (11)\}$. Note that both parallel critical pair systems are empty, since all critical pairs can be joined by $\mathcal{C}$ and $\mathcal{D}$. Therefore the relative termination obligation is trivial and it suffices to prove commutation of $\mathcal{C}$ and $\mathcal{D}$.

We again apply Theorem 3.15 where we choose $\mathcal{C}' = \mathcal{D}' = \{(9) - (11)\}$. Again, all pairs can be joined by $\mathcal{C}'$ and $\mathcal{D}'$, and $\mathsf{PCPS}(\dots) = \emptyset$. Hence, it suffices to prove commutation of $\mathcal{C}'$ and $\mathcal{D}'$, i.e., confluence of $\mathcal{C}'$. At this point the confluence tool CSI is able to find a certifiable proof, and this completes the commutation proof of $\mathcal{R}$ and $\mathcal{S}$.

The full proof of Example 3.16 is available in the supplementary material and it is certified by CeTA. How this is done will be described in the upcoming section.

4 Certification of PCP-Based Criteria

In the previous section we formalized several confluence and commutation criteria in Isabelle. In this section we proceed by developing verified algorithms that can certify whether these criteria have been applied correctly in untrusted confluence and commutation proofs, i.e., proofs that are generated by automated tools that are potentially buggy, or semi-automatically generated proofs such as Example 3.16.

At this point we face the problem that nearly all of the definitions and theorems of the previous section are not executable. We will tackle these problems one by one.

4.1 Computing Parallel Critical Pairs

The first problem is that Definition 3.1 is not executable, in particular the decomposition $\ell_0 = C[\ell_{0,1}, \dots, \ell_{0,n}]$ with existentially quantified $C, \ell_{0,1}, \dots, \ell_{0,n}$ is not formulated constructively at all.[5] To solve this problem we design an algorithm that computes all possible decompositions in a bottom-up way, i.e., by recursion on the term structure of ℓ_0, and it further eliminates contexts that cannot contribute to a PCP.

It is formulated in Isabelle as follows, where the renaming function `ren` has been locally fixed and where we write `f ts` for a term $f(ts)$ and `x` just represents terms that are variables.

```
fun potential_overlaps R x = [(x, [])]
| potential_overlaps R (f ts) = let
  root_os = crit_rules R (f ts);
  merge oss = (f (map fst oss), concat (map snd oss));
  rec_os = concat_lists (map (potential_overlaps R) ts)
in map merge rec_os @
   map (λ lr. (□, [(lr, f ts)])) root_os
```

The function `potential_overlaps R l` computes an overapproximation of all possible decompositions of term `l` and returns these in a list. Each list element is of the form `(C, [(rule1, lp1), ...])`, i.e., it contains the context, the subterms `lp1`, ... of `l` to be plugged into the context, and each rule of `R` for which an overlap is detected. Here, the actual detection for a root-overlap is computed via `crit_rules R (f ts)`: this function computes all root-overlaps of rules of `R` with term `f ts` via `mgu_vd_list`. The expression `concat_lists [xs1, ... xsN]` returns `[[x1, ..., xN] | x1 <- xs1, ..., xN <- xsN]`. Hence, `rec_os` stores a list of potential overlaps lists, where each inner list `oss` contains exactly one overlap information for each argument of `f ts`. These lists are then merged by composing the new context `f (map fst oss)` and by the concatenation of all overlaps of the arguments.

Example 4.1. We illustrate the computation of the potential overlaps for Example 3.7 where we aim at computing the PCPs of $\mathcal{R} = \{a \to b, g(x, y) \to c\}$ and the rule $h(x, a) \to g(x, y) \in \mathcal{S}$. Starting bottom up, we first compute the potential overlaps of $\mathcal{R}$ and x, resulting in $[(x, [])]$, and the overlaps of $\mathcal{R}$ and a, resulting in $[(a, []), (\Box, [(a \to b, a)])]$. Hence, the potential overlaps of $\mathcal{R}$ and $h(x, a)$ are recursively computed as $[(h(x, a), []), (h(x, \Box), [(a \to b, a)])]$.

[5] A similar problem appears in Definition 2.1 where one would have to enumerate all parallel positions of ℓ.

Note that the overapproximation is quite tight: nearly every list entry in `potential_overlaps R l` will give rise to a PCP. There are only two exceptions: first, a decomposition with no holes will be computed and has to be dropped, and second, in case of non-linear input terms `l`, a final unification has to be performed. In our implementation of PCPs we always perform the unification test (`mgu_vd_list ren unifp`) and only keep decompositions with non-empty contexts (`pot_oss` becomes `ne_pot_oss`). We verify in Isabelle that our implementation correctly implements PCPs w.r.t. Definition 3.1. The only difference is that the implementation is list-based, whereas Definition 3.1 is set-based, so we use Isabelle's `set` function in the lemma to convert a list into a set.

```
fun parallel_critical_peaks_of_rule_impl R (l,r) = let
  pot_os = potential_overlaps R l;
  ne_pot_oss = filter (λ (_,rls). rls ≠ []) pot_os;
  generate_pcp (C, rule_lp) = let
    unifp = map (λ ((li,ri),lp). (li,lp)) rule_lp
   in case mgu_vd_list ren unifp of
     None ⇒ []
   | Some (σ, τ) ⇒ let
      rls = map fst rule_lp;
      t = (C · τ)[map (λ (((li,ri),_),σi). ri · σi)
                  (zip rule_lp σ)]
     in [(C · τ, t, l · τ, r · τ, rls)]
in concat (map generate_pcp ne_pot_oss)

lemma set (parallel_critical_peaks_of_rule_impl R lr)
  = parallel_critical_peaks_of_rule (set R) lr
```

Example 4.2 (Continuing Example 4.1)**.** After having computed the potential overlaps of $\mathcal{R}$ and $h(x,a)$ we see that $(h(x,\Box), [(a \to b, a)])$ is the only non-empty potential overlap. The resulting unification problem $[(a,a)]$ is solvable and the algorithm produces the PCP of Example 3.7, represented as quintuple $(h(x,\Box), h(x,b), h(x,a), g(x,y), [a \to b])$.

4.2 Checking Existence of Parallel Critical Pairs

After having a verified algorithm to compute PCPs, the next step is to match the PCPs that are given in a certificate—this might have been computed by some untrusted tool—with those of the verified computation. Here, there might be two differences. The first and most obvious difference is the usage of a different renaming mechanism; and the second problem is the treatment of trivial PCPs $t \overset{C}{\leftarrow\!\!\shortparallel\!\!\rightarrow}{}^{\varepsilon} u$ where $t = u$.

The second problem might arise since our definition also considers self-overlaps, in particular if $\ell \to r \in \mathcal{R}$, then $(r,r) \in \mathrm{PCP}(\mathcal{R})$. However, tools might produce certificates that do not include (r,r). We still accept these proofs since we just check that all non-trivial PCPs are present in the certificate. This is sound, since all trivial PCPs automatically satisfy the various joining-conditions in the commutation criteria of Section 3.

To deal with the first problem, we formalize an algorithm `matching_cp` that checks whether for each computed verified PCP $t \overset{C}{\leftarrow\!\!\shortparallel\!\!\rightarrow}{}^{\varepsilon} s$, a matching PCP $t' \overset{P'}{\leftarrow\!\!\shortparallel\!\!\rightarrow}{}^{\varepsilon} s'$ is present in the certificate. Note that in the format for certificates the more traditional Definition 2.1 is used for PCPs, i.e., a set of positions P' is provided instead of a context. Given these parameters we then check whether (t,s,u) is an instance of (t',s',u') (say via matching substitution σ) and whether the hole-positions of C are exactly P'. Hence, any joining sequence between t' and u' (from the certificate) also yields a joining sequence between $t = t'\sigma$ and $u = u'\sigma$, since (parallel) rewriting is closed under substitutions.

One final problem though is the usage of the variable restricted version of parallel rewriting $\overset{V}{\shortparallel\!\!\rightarrow}$ in Theorems 3.6 and 3.11 (via Definition 3.9): this relation is not closed under substitutions. However, it is closed under variable renamings σ, provided the forbidden variables V are also renamed w.r.t. the inverse of σ. Therefore, our verified algorithm `get_renaming_substs` extracts the renaming σ from the matching algorithm and computes its inverse. Although this sounds trivial, the definition of `get_renaming_substs` and its auxiliary function `extend_finite_map` are definitely not immediate. For instance, the latter algorithm extends an injective map on a finite set of variables, e.g., $\{x \mapsto y, y \mapsto z\}$, to a bijection on the set of all variables, e.g., $\{x \mapsto y, y \mapsto z, z \mapsto x\}$ and all other variables are mapped to theirselves.

Finally, we put everything together, e.g., we define an algorithm `check_toyama_pcp_sequence_comm`. It first uses a combination of tests based on `get_renaming_substs` and `matching_cp` to ensure that variants of all verified PCPs are present in the certificate, and then an algorithm is invoked to compute that indeed all PCPs in the certificate are joinable in a way that is required by the various commutation criteria.

Ultimately, our soundness results are of the following form: whenever a checker accepts a certificate, then the preconditions of one of the theorems in Section 3 is satisfied, e.g. `check_toyama_pcp_sequence_comm` ensures the first joining condition in Theorem 3.6.

4.3 Checking Joining Sequences

In Section 4.2 we just assumed that one can check whether a PCP is joinable. Actually joinability is undecidable for all of the criteria that are stated in Section 3, since these joinability conditions always contain at least one occurrence of $\to^*$, i.e., reachability.

Our checkers for joinability offer two modes of operation, both aiming at easy-to-produce certificates.

The first mode is the automatic mode where just an upper limit n on the number of steps is given. For example, when checking an application of Theorem 3.15, one needs to ensure $t \to_{\mathcal{S}}^* \cdot {}_{\mathcal{R}}^*\!\leftarrow u$ for all $(t,u) \in \mathrm{PCP}(\mathcal{R},\mathcal{S})$. Here the checker will try to find some v such that $t \to_{\mathcal{S}}^{n_1} v {}_{\mathcal{R}}^{n_2}\!\leftarrow u$ with $n_1+n_2 \leq n$ by breadth-first search. The advantage of this automatic

mode is that the certificates are trivial to generate, at the cost of high computation costs during certificate checking. Moreover, the automatic mode cannot cope well with extra variables in right-hand sides of rules or applying rewrite steps in the "wrong" direction, as is allowed in the decreasing-condition in Definition 3.9.

Therefore, in the second mode of operation the joining sequences have to be provided. To be more precise, to join a PCP (t, u) the certificate has to provide a list of intermediate terms, i.e.,

$$t = s_0 \sim s_1 \sim s_2 \sim \ldots \sim s_n = u \tag{A}$$

where the symbol $\sim$ represents some rewrite step. Of course, the certificate might also contain information on which rewrite step was applied, but this would make the certificates more bulky, and therefore we decided to leave this information implicit, i.e., it has to be inferred by the certifier.

Recall that in Definition 3.9, the joins have a complex shape

$$t \to_{\mathcal{R}_1}^* \cdot \overset{}{\underset{\mathcal{R}_2}{\Mapsto}} \cdot \to_{\mathcal{R}_3}^* \cdot \overset{V}{\underset{\mathcal{R}_4}{\Mapsto}} \cdot \to_{\mathcal{R}_5}^* u \tag{B}$$

with five different TRSs $\mathcal{R}_1, \ldots, \mathcal{R}_5$. Here, $\mathcal{R}_1$ is defined as $\bigcup_{k'<k}(\mathcal{R}_{\phi,k'}^{-1} \cup \mathcal{S}_{\psi,k'})$ such that $\to_{\mathcal{R}_1}$ is exactly $\underset{\vee k}{\longleftrightarrow}$ in Definition 3.9, and similarly one can define $\mathcal{R}_2, \ldots, \mathcal{R}_5$.

The question is now how to map the steps in (A) onto the desired shape in (B).

To this end, our implemented checker uses a greedy approach. It starts from the left ($i := 0$) such that $t = s_i$ and then increments i as long as $s_i \to_{\mathcal{R}_1} s_{i+1}$. It then increments i if $s_i \underset{\mathcal{R}_2}{\Mapsto} s_{i+1}$, then increments i repeatedly as long as $s_i \to_{\mathcal{R}_3} s_{i+1}$, and in the same way it deals with $\mathcal{R}_4$ and $\mathcal{R}_5$. Eventually it checks $s_i = s_n = u$ in order to report a valid join.

Observe that this greedy approach is not rejecting any valid joining sequences, since all the participating relations are reflexive: any $\to^*$ relation is reflexive and also a single parallel step is a reflexive relation. So it might happen that the actual intended joining sequence—the one that is detected by the tool—is

$$t = s_0 \to_{\mathcal{R}_1} s_1 \underset{\mathcal{R}_2}{\Mapsto} s_2 \overset{V}{\underset{\mathcal{R}_4}{\Mapsto}} s_3 \to_{\mathcal{R}_5} s_4 \to_{\mathcal{R}_5} s_5 = u,$$

but it is accepted by the checker as

$$t = s_0 \to_{\mathcal{R}_1} s_1 \to_{\mathcal{R}_1} s_2 \to_{\mathcal{R}_1} s_3 \overset{V}{\underset{\mathcal{R}_4}{\Mapsto}} s_4 \to_{\mathcal{R}_5} s_5 = u.$$

For further convenience of tool authors, we implemented the checker in a way that it automatically converts $\to^*$ into $\Mapsto^*$, i.e., the steps from s_i to s_{i+1} can always be parallel steps. In this way, a joining sequence $f(a,a,a) \to f(a,a,b) \to f(a,b,b) \to f(b,b,b)$ via rule $a \to b$ can be compressed into a single parallel step $f(a,a,a) \Mapsto f(b,b,b)$ in the certificate.

4.4 Checking Full Confluence and Commutation Proofs

Already prior to our work, CeTA was capable to check certificates of confluence and commutation proofs. These may consist of a combination of various techniques. Whereas the support of confluence was already quite good (CeTA supported 11 different criteria to ensure confluence), there was only little support for commutation: being parallel closed [21] or being development closed [13, 14] have been the only commutation criteria in CeTA. We therefore added all of the techniques of Section 3 to CeTA by integrating the corresponding checker functions of this section as new alternatives.

Our extensions do not change the soundness result of CeTA: whenever CeTA accepts a certificate of a confluence proof or of a commutation proof, then the input TRS is confluent, or the input TRSs commute, respectively. However, because of our extensions now more proofs can be accepted by CeTA, namely those proofs that require PCPs.

5 Evaluation

In order to evaluate the presented contributions we extended the confluence tool Hakusan to support certificate outputs for the compositional confluence criteria of rule labeling based on PCPs (Theorem 3.11) and of the parallel critical pair systems (Theorem 3.15).[6] Since Hakusan only uses the combination of Theorems 3.11 and 3.15 (assuming $\mathcal{R} = \mathcal{S}$ and $C = \mathcal{D}$), by this extension all confluence proofs of the tool are certifiable by CeTA. Note that certificates generated by the tool indicate joining sequences explicitly.

We briefly recall how the criteria are automated in the tool [23, Section 8]: Joinability of PCPs is tested by finding suitable joining sequences of the form $\to^{n_1} \cdot {}^{n_2}\!\leftarrow$ with $n_1, n_2 \leq 5$. Suitable labeling functions for rule labeling are found by the SMT solver Z3 [3] (version 4.12.2) via SMT encodings [7, Section 4], while relative termination is shown by employing the termination tool TTT2 [15] (version 1.20.1). Suitable confluent subsystems for the compositional confluence criteria are found by enumeration.

The problem set used in experiments consists of 462 left-linear TRSs taken from the confluence problems database COPS [8]. Out of the 462 TRSs, at least 233 are known to be confluent while at least 189 are non-confluent. The tests were run on a PC with Intel Core i7-8500Y CPU (1.5 GHz) and 16 GiB memory of RAM using timeouts of 60 seconds.

Table 1 summarizes the experimental results for Theorem 3.11, Theorem 3.15, and their combinations.

- **R**: Theorem 3.11 with $\mathcal{R}_{\phi,0} = \mathcal{R}_{\psi,0} = \emptyset$.
- **C**: Successive application of Theorem 3.15.
- **RC**: Theorem 3.11 but confluence of an employed subsystem is shown by (single) application of Theorem 3.15 (with $C = \emptyset$).

[6] The tool and the experimental data are available from: https://www.jaist.ac.jp/project/saigawa/

Table 1. Comparison of criteria on 462 left-linear TRSs.

	R	**C**	**RC**	**CR**	**k**	**s**	**a**	**o**	**rt**	**cc**
certified	133	107	141	138	50	62	52	90	96	77
timeouts	*30*	*67*	*116*	*42*	*102*	*0*	*3*	*96*	*102*	*102*

Table 2. Comparison of tools on 462 left-linear TRSs.

	Hakusan	ACP	CSI	FORT-h	*total*
proved	144	73	159	34	179
certified	144	73	157	33	177

- **CR**: Theorem 3.15 where confluence of a subsystem is shown by Theorem 3.11 (with $\mathcal{R}_{\phi,0} = \mathcal{R}_{\psi,0} = \emptyset$).

Since searching suitable subsystems is time consuming, **RC** applies Theorem 3.15 only once. Note that successive applications of Theorem 3.11 can be simulated by a single application of it. For a comparison sake we also include in the table the results of the confluence criteria that CeTA supports:

- **k**: Knuth and Bendix's confluence criterion [12].
- **s**: Strong closedness [10].
- **a**: Almost development closedness [14, 30].
- **o**: The original rule labeling for linear TRSs [31].
- **rt**: Rule labeling enhanced by relative termination [33].
- **cc**: Successive application of the confluence criterion based on critical-pair-closing systems [9].

For the comparisons we employed the other confluence tools ACP version 0.72 and CSI version 1.2.7. The former tool was used for **k**, **s**, and **a**, while the latter was for **o**, **rt**, and **cc**.

Table 1 indicates the numbers of problems proved by the corresponding criterion and of problems that exceeded the timeout. For instance, the numbers in column **R** are read as follows: For 133 TRSs Hakusan produced certificates of confluence proofs by using **R**, and moreover, their certificates were successfully verified by CeTA. For 30 TRSs the tool could not finish confluence analysis within the time. For the remaining 299 TRSs nothing is concluded by **R**.

The table clearly shows that the confluence proving power of **R** and **C** are comparable to those of the existing methods listed above (**k**, **s**, **a**, **o**, **rt**, and **cc**). If a certificate provides joining sequences, the execution time of CeTA is negligible. In fact, any certificate produced by Hakusan was verified within a second.

Table 2 summarizes the experimental results for Hakusan and the confluence tools ACP, CSI, and FORT-h version 2.0 [17] supporting certificate outputs. The latter tools are taken from the TRS category of the annual confluence competition (CoCo) [16] held in 2023.

- Hakusan applies **RC** and then **CR** after applying the redundant rule elimination technique called the reduction method [24, Corollary 8.4].
- ACP uses **s** and **a**.
- CSI applies **k**, **s**, **a**, **o**, **rt**, and **cc** after applying redundant rule addition/elimination techniques [18].
- FORT-h is a decision procedure for left-linear and right-ground systems, and the generated answers are certified by FORTify version 2.0. Note that our problem set contains 122 left-linear and right-ground TRSs.
- *total* indicates the unions of their results.

The reduction method used in Hakusan is a special case of Theorem 3.11. So all techniques in Hakusan are covered by the presented formalization (Theorems 3.11 and 3.15).

Concerning the results,[7] all problems handled by ACP can also be handled by Hakusan. FORT-h and FORTify can solve two problems that Hakusan cannot handle. CSI can produce more certifiable confluence proofs than Hakusan, but nevertheless there are also 20 TRSs in our experiments for which only Hakusan was able to provide a certifiable proof; in detail, 16 TRSs are handled by **R**, 2 by **RC**, 1 by **CR**, and 1 by **RC** with the reduction method. The combination of Hakusan and CSI amounts to 177. It is worth noting that Toyama's confluence criterion (Theorem 3.6) and PCP-based parallel rule labeling have already been implemented in ACP and CSI, respectively. Therefore, if the tools support certificate generation for these criteria, at least 82 proofs can be certified by ACP and 173 by CSI. Overall it is demonstrated that our work significantly increases the power of reliable confluence analysis.

6 Summary and Outlook

Our initial aim was to formalize existing PCP-based confluence techniques in Isabelle. This led to a new context-based definition of PCPs, and to a generalization of all these techniques to commutation.

In total, more than 4300 lines of Isabelle have been added to IsaFoR. Here, 400 lines have been spent on combining renamings with unification, and 2400 lines cover all the over results in Section 3. The remaining 1500 lines define and verify the checkers of Section 4.

The new version of CeTA is now able to certify all proofs of Hakusan, leading to a significant increase in certifiable confluence proofs. It remains as future work to figure out, whether our generalization to use PCPs for commutation is also useful for automated commutation analysis. For instance, the proof in Example 3.16 looks quite natural and might be generated by automated tools in the future.

Acknowledgements

This research was supported by the Austrian Science Fund (FWF) project I 5943, the Japan Society for the Promotion of

[7] For 2 TRSs, CSI claimed to have a confluence proof, but then crashed during the certificate generation. FORTify could not verify one confluence certificate in 60 seconds.

Science (JSPS) KAKENHI Grant Number JP22K11900, and JSPS-FWF Grant Number JPJSBP120222001.

References

[1] Takahito Aoto, Junichi Yoshida, and Yoshihito Toyama. 2009. Proving Confluence of Term Rewriting Systems Automatically. In *Rewriting Techniques and Applications, 20th International Conference, RTA 2009, Brasília, Brazil, June 29 - July 1, 2009, Proceedings (Lecture Notes in Computer Science, Vol. 5595)*, Ralf Treinen (Ed.). Springer, 93–102. https://doi.org/10.1007/978-3-642-02348-4_7

[2] Franz Baader and Tobias Nipkow. 1998. *Term rewriting and all that.* Cambridge University Press.

[3] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 4963)*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

[4] Gilles Dowek, Gaspard Férey, Jean-Pierre Jouannaud, and Jiaxiang Liu. 2022. Confluence of left-linear higher-order rewrite theories by checking their nested critical pairs. *Math. Struct. Comput. Sci.* 32, 7 (2022), 898–933. https://doi.org/10.1017/S0960129522000044

[5] Bernhard Gramlich. 1995. *Confluence without Termination via Parallel Critical Pairs.* Technical Report SEKI-Report SR-95-13. Fachbereich Informatik, Universität Kaiserslautern.

[6] Bernhard Gramlich. 1996. Confluence without Termination via Parallel Critical Pairs. In *Trees in Algebra and Programming - CAAP'96, 21st International Colloquium, Linköping, Sweden, April, 22-24, 1996, Proceedings (Lecture Notes in Computer Science, Vol. 1059)*, Hélène Kirchner (Ed.). Springer, 211–225. https://doi.org/10.1007/3-540-61064-2_39

[7] Nao Hirokawa and Aart Middeldorp. 2011. Decreasing Diagrams and Relative Termination. *J. Autom. Reason.* 47, 4 (2011), 481–501. https://doi.org/10.1007/s10817-011-9238-x

[8] Nao Hirokawa, Julian Nagele, and Aart Middeldorp. 2018. Cops and CoCoWeb: Infrastructure for Confluence Tools. In *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10900)*, Didier Galmiche, Stephan Schulz, and Roberto Sebastiani (Eds.). Springer, 346–353. https://doi.org/10.1007/978-3-319-94205-6_23

[9] Nao Hirokawa, Julian Nagele, Vincent van Oostrom, and Michio Oyamaguchi. 2019. Confluence by Critical Pair Analysis Revisited. In *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11716)*, Pascal Fontaine (Ed.). Springer, 319–336. https://doi.org/10.1007/978-3-030-29436-6_19

[10] Gérard P. Huet. 1980. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems: Abstract Properties and Applications to Term Rewriting Systems. *J. ACM* 27, 4 (1980), 797–821. https://doi.org/10.1145/322217.322230

[11] Brian Huffman and Ondrej Kuncar. 2013. Lifting and Transfer: A Modular Design for Quotients in Isabelle/HOL. In *Certified Programs and Proofs - Third International Conference, CPP 2013, Melbourne, VIC, Australia, December 11-13, 2013, Proceedings (Lecture Notes in Computer Science, Vol. 8307)*, Georges Gonthier and Michael Norrish (Eds.). Springer, 131–146. https://doi.org/10.1007/978-3-319-03545-1_9

[12] Donald E. Knuth and Peter B. Bendix. 1970. Simple Word Problems in Universal Algebras. In *Computational Problems in Abstract Algebra*, J. Leech (Ed.). Pergamon Press, 263–297. https://doi.org/10.1016/B978-0-08-012975-4.50028-X

[13] Christina Kohl and Aart Middeldorp. 2023. A Formalization of the Development Closedness Criterion for Left-Linear Term Rewrite Systems. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2023, Boston, MA, USA, January 16-17, 2023*, Robbert Krebbers, Dmitriy Traytel, Brigitte Pientka, and Steve Zdancewic (Eds.). ACM, 197–210. https://doi.org/10.1145/3573105.3575667

[14] Christina Kohl and Aart Middeldorp. 2023. Formalizing Almost Development Closed Critical Pairs (Short Paper). In *14th International Conference on Interactive Theorem Proving, ITP 2023, July 31 to August 4, 2023, Białystok, Poland (LIPIcs, Vol. 268)*, Adam Naumowicz and René Thiemann (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 38:1–38:8. https://doi.org/10.4230/LIPIcs.ITP.2023.38

[15] Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. 2009. Tyrolean Termination Tool 2. In *Rewriting Techniques and Applications, 20th International Conference, RTA 2009, Brasília, Brazil, June 29 - July 1, 2009, Proceedings (Lecture Notes in Computer Science, Vol. 5595)*, Ralf Treinen (Ed.). Springer, 295–304. https://doi.org/10.1007/978-3-642-02348-4_21

[16] Aart Middeldorp, Julian Nagele, and Kiraku Shintani. 2021. CoCo 2019: report on the eighth confluence competition. *Int. J. Softw. Tools Technol. Transf.* 23, 6 (2021), 905–916. https://doi.org/10.1007/s10009-021-00620-4

[17] Fabian Mitterwallner, Alexander Lochmann, Aart Middeldorp, and Bertram Felgenhauer. 2021. Certifying Proofs in the First-Order Theory of Rewriting. In *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 12652)*, Jan Friso Groote and Kim Guldstrand Larsen (Eds.). Springer, 127–144. https://doi.org/10.1007/978-3-030-72013-1_7

[18] Julian Nagele, Bertram Felgenhauer, and Aart Middeldorp. 2015. Improving Automatic Confluence Analysis of Rewrite Systems by Redundant Rules. In *Proceedings of the 26th International Conference on Rewriting Techniques and Applications (LIPIcs, Vol. 36)*. 257–268. https://doi.org/10.4230/LIPIcs.RTA.2015.257

[19] Julian Nagele, Bertram Felgenhauer, and Harald Zankl. 2017. Certifying Confluence Proofs via Relative Termination and Rule Labeling. *Log. Methods Comput. Sci.* 13, 2 (2017). https://doi.org/10.23638/LMCS-13(2:4)2017

[20] Julian Nagele and Aart Middeldorp. 2016. Certification of Classical Confluence Results for Left-Linear Term Rewrite Systems. In *Interactive Theorem Proving - 7th International Conference, ITP 2016, Nancy, France, August 22-25, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9807)*, Jasmin Christian Blanchette and Stephan Merz (Eds.). Springer, 290–306. https://doi.org/10.1007/978-3-319-43144-4_18

[21] Julian Nagele and Aart Middeldorp. 2016. Certification of Classical Confluence Results for Left-Linear Term Rewrite Systems. In *Interactive Theorem Proving - 7th International Conference, ITP 2016, Nancy, France, August 22-25, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9807)*, Jasmin Christian Blanchette and Stephan Merz (Eds.). Springer, 290–306. https://doi.org/10.1007/978-3-319-43144-4_18

[22] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. 2002. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic.* Lecture Notes in Computer Science, Vol. 2283. Springer. https://doi.org/10.1007/3-540-45949-9

[23] Kiraku Shintani and Nao Hirokawa. 2022. Compositional Confluence Criteria. In *7th International Conference on Formal Structures for Computation and Deduction, FSCD 2022, August 2-5, 2022, Haifa, Israel (LIPIcs, Vol. 228)*, Amy P. Felty (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 28:1–28:19. https://doi.org/10.4230/LIPIcs.FSCD.2022.28

[24] Kiraku Shintani and Nao Hirokawa. 2023. *Compositional Confluence Criteria.* Technical Report. arXiv. https://doi.org/10.48550/arXiv.2303.

03906

[25] Christian Sternagel and René Thiemann. 2013. Formalizing Knuth-Bendix Orders and Knuth-Bendix Completion. In *24th International Conference on Rewriting Techniques and Applications, RTA 2013, June 24-26, 2013, Eindhoven, The Netherlands (LIPIcs, Vol. 21)*, Femke van Raamsdonk (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 287–302. https://doi.org/10.4230/LIPIcs.RTA.2013.287

[26] the Isabelle community. 2023. Tutorials and manuals for Isabelle2023. https://isabelle.in.tum.de/documentation.html. Accessed: 2023-11-22.

[27] René Thiemann and Christian Sternagel. 2009. Certification of Termination Proofs Using CeTA. In *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5674)*, Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel (Eds.). Springer, 452–468. https://doi.org/10.1007/978-3-642-03359-9_31

[28] Yoshihito Toyama. 1981. On the Church-Rosser property of term rewriting systems. *NTT ECL Technical Report (Japanese)* 17672 (1981).

[29] Yoshihito Toyama. 1988. Commutativity of Term Rewriting Systems. In *Programming of Future Generation Computers II*. North-Holland, 393–407.

[30] Vincent van Oostrom. 1997. Developing Developments. *Theor. Comput. Sci.* 175, 1 (1997), 159–181. https://doi.org/10.1016/S0304-3975(96)00173-9

[31] Vincent van Oostrom. 2008. Confluence by Decreasing Diagrams. In *Rewriting Techniques and Applications, 19th International Conference, RTA 2008, Hagenberg, Austria, July 15-17, 2008, Proceedings (Lecture Notes in Computer Science, Vol. 5117)*, Andrei Voronkov (Ed.). Springer, 306–320. https://doi.org/10.1007/978-3-540-70590-1_21

[32] Harald Zankl, Bertram Felgenhauer, and Aart Middeldorp. 2011. CSI - A Confluence Tool. In *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6803)*, Nikolaj S. Bjørner and Viorica Sofronie-Stokkermans (Eds.). Springer, 499–505. https://doi.org/10.1007/978-3-642-22438-6_38

[33] Harald Zankl, Bertram Felgenhauer, and Aart Middeldorp. 2015. Labelings for Decreasing Diagrams. *J. Autom. Reason.* 54, 2 (2015), 101–133. https://doi.org/10.1007/s10817-014-9316-y

Received 2023-09-19; accepted 2023-11-25