Mechanizing Confluence

dissertation

by

Julian Nagele

submitted to the Faculty of Mathematics, Computer Science and Physics of the University of Innsbruck

in partial fulfillment of the requirements for the degree of Doctor of Philosophy

advisor: Univ.-Prof. Dr. Aart Middeldorp

Innsbruck, October 2017

universität
innsbruck

Mechanizing Confluence

Automated and Certified Analysis of First- and Higher-Order Rewrite Systems

universität
innsbruck

dissertation

Mechanizing Confluence

Automated and Certified Analysis of First- and Higher-Order Rewrite Systems

Julian Nagele
mail@jnagele.net

October 2017

advisor: Univ.-Prof. Dr. Aart Middeldorp

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht.
Die vorliegende Arbeit wurde bisher in gleicher oder ähnlicher Form noch nicht als Magister-/Master-/Diplomarbeit/Dissertation eingereicht.

Datum	Unterschrift

Abstract

This thesis is devoted to the mechanized confluence analysis of rewrite systems. Rewrite systems consist of directed equations and computation is performed by successively replacing instances of left-hand sides of equations by the corresponding instance of the right-hand side. Confluence is a fundamental property of rewrite systems, which ensures that different computation paths produce the same result. Since rewriting is Turing-complete, confluence is undecidable in general. Nevertheless, techniques have been developed that can be used to determine confluence for many rewrite systems and several automatic confluence provers are under active development. Our goal is to improve three aspects of automatic confluence analysis, namely (a) reliability, (b) power, and (c) versatility. The importance of these aspects is witnessed by the annual international confluence competition, where the leading automated tools analyze confluence of rewrite systems. To improve the *reliability* of automatic confluence analysis, we formalize confluence criteria for rewriting in the proof assistant Isabelle/HOL, resulting in a verified, executable checker for confluence proofs. To enhance the *power* of confluence tools, we present a remarkably simple technique, based on the addition and removal of redundant equations, that strengthens existing techniques. To make automatic confluence analysis more *versatile*, we develop a higher-order confluence tool, making automatic confluence analysis applicable to systems with functional and bound variables.

Acknowledgments

At the beginning of this thesis I want to express my gratitude to the many people who have influenced my work and made it an exciting and, on the whole, enjoyable journey.

First of all, I would like to thank my advisor Aart Middeldorp. His advice, support, and encouragement were vital. I sincerely appreciate his guidance in both research and other aspects of life, such as traveling Japan. I am grateful to all members of the Computational Logic group, both past and present, for taking me in and creating such a pleasant environment. Thank you for everything you have taught me. I explicitly mention Bertram Felgenhauer, for countless insights and occasional Albernheiten, Cezary Kaliszyk for motivating enthusiasm, Vincent van Oostrom, for collaboration and chitchat on etymology, Christian Sternagel, for energizing discussions and Isabelle and TEX support, Thomas Sternagel, for being my PhD and coffee companion, René Thiemann, for introducing me to interactive theorem proving in the first place, and Sarah Winkler for internship encouragement and advice. Special thanks go to Harald Zankl, for taking me under his wing and supporting me through some of the most important steps of the way.

I had the pleasure of collaborating with many amazing people. I am indebted to them all, in particular my fellow confluence competitors: the problem submitters, the tool authors, and especially my CoCo co-organizers, Takahito Aoto, Nao Hirokawa, and Naoki Nishida. I would also like to say thank you to Nuno Lopes, who was my mentor during my internship at Microsoft Research, for offering me a glimpse into the fascinating world of compilers and challenging me to apply my knowledge in an unfamiliar domain.

Finally, I wish to thank my friends and family, especially my parents Barbara and Joachim, for their constant support and unwavering belief that has enabled me to pursue this work, and of course Maria, for all the experiences and things we share.

This work has been supported by the Austrian Science Fund through the FWF projects P22467 and P27528.

Contents

Chapter 1

Introduction

Science is knowledge which we understand so well
that we can teach it to a computer.
Donald E. Knuth

In safety-critical systems, like medical devices and spaceflight, ensuring correctness of computer programs is imperative. Notorious failures, like the malfunctioning Therac-25 medical accelerator, the Pentium floating point division bug, or the crash of the Ariane 5 rocket, show that modern applications have long reached a level of complexity where human judgment and testing are not sufficient to guarantee faultless operation.

Instead formal methods can be used to avoid subtle errors and provide a high degree of reliability. Formal methods apply mathematics and logic to software, formal verification employs these methods for proving correctness of programs. We argue that in computer science formality is not optional. Formal methods merely make that explicit, after all, programming languages are formal languages and every program is a formula.

Recent years have seen tremendous success in formally verified software, such as the seL4 microkernel and the CompCert C compiler. Alas, formal verification is still very labor-intensive. It is our believe that in order for formal verification to be widely adopted, a high degree of automation, where large parts of the verification process can be performed at the push of a button, is indispensable.

Consequently, we need tools that check properties of critical parts of programs automatically. However, these tools are programs themselves, which makes them vulnerable to the very same problems. That is, they will contain errors. To ensure correctness of the verification process itself, a common approach is to use so-called computational proof assistants, which are computer programs designed for checking mathematical inferences in a highly trustworthy fashion.

1.1 Rewriting and Confluence

Considering the myriad of existing programming languages and their complexity, in order to facilitate mathematical reasoning and to ensure that results are applicable to all concrete implementations, we would like to perform correctness proofs on a more abstract level, using a mathematical model of computation.

Rewriting is such a model of computation. It is the process of transforming objects in a step-wise fashion, replacing equals by equals. Typically the objects are expressions in some formal language, and the transformations are given by a set of equations. Applied in a directed way, they describe rewrite steps, performed by replacing instances of left-hand sides of equations by the corresponding instance of the right-hand side. If no further replacement is possible, we have reached the result of our computation. Consider the equations

$$\begin{aligned} 0 + y &= y & 0 \times y &= 0 \\ \mathsf{s}(x) + y &= \mathsf{s}(x + y) & \mathsf{s}(x) \times y &= y + (x \times y) \end{aligned}$$

Applying them from left to right, they are rewrite rules for computing addition and multiplication on natural numbers represented as Peano numbers, i.e., using a zero constant and a successor function. For instance we can compute 1×2 as follows:

$$\begin{aligned} \underline{\mathsf{s}(0) \times \mathsf{s}(\mathsf{s}(0))} &= \mathsf{s}(\mathsf{s}(0)) + (\underline{0 \times \mathsf{s}(\mathsf{s}(0))}) \\ &= \underline{\mathsf{s}(\mathsf{s}(0)) + 0} \\ &= \mathsf{s}(\underline{\mathsf{s}(0) + 0}) \\ &= \mathsf{s}(\mathsf{s}(\underline{0 + 0})) \\ &= \mathsf{s}(\mathsf{s}(0)) \end{aligned}$$

In each step we instantiated an equation by replacing the variables x and y, and after finding the left-hand side of that equation in a subexpression (indicated by underlining), we replaced it by the corresponding right-hand side.

Specifying programs by such sets of directed equations, called rewrite systems, is the essence of the functional programming paradigm. Indeed rewriting as sketched above is Turing-complete and consequently, all basic properties like termination, i.e., the question whether all computations finish in a finite amount of time, are undecidable.

Returning to our example equations for arithmetic we find that, given some expression, there are often multiple possibilities to apply an equation, yielding different results. For instance, in the second step of the derivation above, we replaced the subexpression $0 \times \mathsf{s}(\mathsf{s}(0))$ by 0 using the second equation, but we could also have rewritten the whole expression using the third equation, which would have resulted in $\mathsf{s}(\mathsf{s}(0) + 0 \times \mathsf{s}(\mathsf{s}(0)))$. So how can we be sure that our equations, given the same input, will always yield the same result? Often steps even overlap, that is, they act on some common part of the expression that is rewritten. A set of rules where where such diverging computation paths can always be joined back together is called confluent.[1]

Confluence is a fundamental notion that appears in several areas of computer science. Besides guaranteeing partial correctness, i.e., uniqueness of results, when implementing functions by means of rewrite rules, confluence was used by Church and Rosser to establish consistency of the λ-calculus and later by Knuth and Bendix as the basis for the method of completion for deciding equational theories [54]. Generalizing confluence to multiple sets of equations yields the related notion of commutation, which can be used to study correctness of program transformations [47, 110].

Using rewriting as a model for functional programming, or in applications like theorem proving and program transformations, needs the possibility to not only specify functions, but to make them the objects of the rewriting process itself. Consider for example higher-order functions in functional programming languages, as in the following Haskell program that applies a function to each element of a list.

```
map f [] = []
map f (h:t) = f h : map f t
```

Inspecting the second equation, one finds that `f` occurs as a variable on the left-hand side, but is also applied to `h` as a function on the right-hand side. Similarly, bound variables are a common feature in many programming languages and logics. For example the following equation might be used to express a quantifier equivalence in predicate logic:

$$\neg\forall x.P(x) = \exists x.\neg P(x)$$

Also in mathematics many equations contain bound variables, for instance when integrals or derivatives are present. This explains the need for a theory

[1] From Latin confluere: "to flow together".

of higher-order rewriting, where the equations may contain bound variables and functional variables.

1.2 Formalization and Certification

In recent years there has been tremendous progress in establishing confluence or non-confluence of rewrite systems automatically, with a number of tools under active development. For first-order rewriting the main tools are ACP [8], CoLL-Saigawa [51, 96], and our own tool, CSI [66, 114]. Also for confluence of higher-order rewriting tool support has emerged, witnessed by ACPH and CSIˆho, which are extensions of the first-order provers ACP and CSI, and the new tool SOL [37]. Because confluence is undecidable, there are three different answers these fully automatic tools can give: `YES` if confluence could be established, `NO` if non-confluence was proved, and `MAYBE` (or a timeout) if no definitive answer could be obtained.

The achievements in confluence research have enabled the confluence competition (CoCo) [4] where automated tools try to establish/refute confluence.[2] The problems the tools try to solve are from the confluence problems database (Cops, version 764 at the time of writing).[3]

As the power of the confluence provers grew, so did their complexity and the proofs they produce are often complicated and large. So if we are to use them in a formal verification context, should we trust their output? Of course not, and indeed there have been occasions where confluence tools delivered wrong proofs and answers. To remedy this dilemma two approaches suggest themselves. An obvious solution would be to formally verify that the confluence tool is correct. But this approach clearly does not scale well. Since tools use different algorithms and are written in different languages, each would need its own correctness proof. Even worse, after every change in a tool we would have to adapt the corresponding proof, which severely limits the potential for optimization. The other solution is to check the output of the tools using a trustable, independent certifier. A certifier is a different kind of automated tool that reads proof certificates and either accepts them as correct or rejects them as erroneous, thereby increasing credibility of the proofs found by the confluence tools. To ensure correctness of the certifier itself, the predominant solution is

[2] `http://coco.nue.riec.tohoku.ac.jp/`
[3] `http://cops.uibk.ac.at`

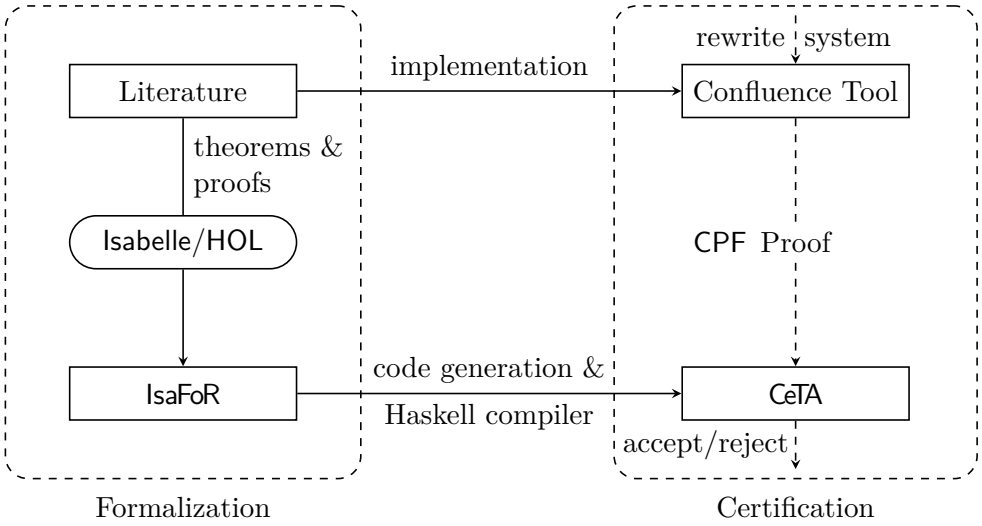


Figure 1: Certification of confluence proofs.

to use proof assistants like Coq [14] and Isabelle/HOL [74] to first formalize the underlying theory in the proof assistant, and then use this formalization to obtain verified, executable check functions for inspecting the certificates.

The tool CeTA [106] is such a certifier, for termination, confluence and complexity proofs for rewrite systems. Other certifiers exist for termination proofs, notably Rainbow [16] and CiME3 [17]. Proof certificates are specified in the certification problem format (CPF) [99]. The main structure of a certificate in CPF consists of an input, in our case the rewrite system whose confluence should be certified, and a proof. Given a certificate in CPF, CeTA will either answer `CERTIFIED` or return a detailed error message why the proof was `REJECTED`. Its correctness is formally proved as part of IsaFoR, the Isabelle Formalization of Rewriting. IsaFoR contains executable "check"-functions for each formalized proof technique together with formal proofs that whenever such a check succeeds, the technique is indeed applied correctly. Isabelle's code-generation facility [35], which ensures partial correctness [36], is used to obtain a trusted Haskell program from these check functions: the certifier CeTA. The big picture of mechanized confluence analysis is shown in Figure 1: Confluence tools implement results from the literature and use them to automatically check confluence of rewrite systems, producing proofs in CPF. The theorems and

proofs about the implemented confluence results are formalized in Isabelle/HOL, resulting in the formal library IsaFoR, from which the certifier CeTA is generated. CeTA can then inspect the proofs in CPF.

1.3 Overview

We contribute to three aspects of mechanized confluence analysis: reliability, power, and versatility. To improve the reliability of confluence tools, we formalize confluence criteria using Isabelle/HOL and integrate them into IsaFoR and CeTA. To enhance their power, we present a technique, based on the addition and removal of redundant rewrite rules, that strengthens existing criteria. Finally, to make confluence analysis more versatile we develop the higher-order confluence tool CSI^ho.

The remainder of the thesis is organized as follows. In Chapter 2 we introduce the basics about first- and higher-order rewriting and confluence that we will use later on. The next two chapters present the main confluence criteria that we formalize on top of IsaFoR. Chapter 3 is devoted to confluence criteria that are based on resolving overlapping rewrite steps in a restricted fashion, while Chapter 4 describes our formalization of the rule labeling, which shows confluence provided rewrite steps can be labeled in a certain way, using labels that are equipped with a well-founded order. In Chapter 5 we describe a simple, but surprisingly useful technique, based on adding and removing rewrite rules that can be simulated by other rules, which often makes other confluence criteria more powerful. Next we look at higher-order rewriting, where we will first discuss confluence of the λ-calculus in Chapter 6, before describing the theory behind our higher-order confluence prover CSI^ho in Chapter 7. Both CSI and CSI^ho are described in detail in Chapter 8, where we also provide an extensive experimental evaluation.

A large part of this work is devoted to formalizing confluence criteria in Isabelle/HOL. For presentation in this thesis we use different levels of abstraction. Code listings display parts of the formalization directly generated from the formal Isabelle text (using syntax translations for pretty printing), while in the main text we prefer to use standard mathematical notation and descriptions. Whenever this leads to noteworthy discrepancies they will be pointed out along the way. We hope that this mix of standard rewriting notation and formalization snippets will appeal to readers from both backgrounds.

We provide the Isabelle/HOL theory files for the formalizations from the subsequent chapters as part of IsaFoR. To be able to browse these files one needs a working Isabelle installation as well as the archive of formal proofs, which is available at

`http://www.isa-afp.org/`

The IsaFoR/CeTA version that corresponds to formalizations described this thesis is 2.31. It is available from

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

For further compilation instructions, we refer to the `README` file in the IsaFoR-sources.

Chapter 2

Rewriting

Well, when all is said and done, the only thing computers can do for us is to manipulate symbols and produce results of such manipulations.

Edsger W. Dijkstra (EWD 1036)

In this chapter we introduce several rewriting formalisms. We will consider first-order term rewrite systems, the lambda calculus, and higher-order rewrite systems, which can be understood as combination of the former two. To deal with the common notions of all three in a uniform manner, we first introduce abstract rewrite systems. They are abstract in the sense that all the potential structure of the objects that are subject to rewriting is abstracted away. While we do strive to keep this thesis self-contained, the material is quite dense, and so familiarity with the basics of rewriting [11, 105] and (typed) lambda calculi [12, 13] will be helpful.

2.1 Preliminary Notions

This section gives an overview of general terminology and fixes common notation that will be used later on. We denote the set of *natural numbers* by $\mathbb{N}$ and the set of positive natural numbers by $\mathbb{N}_+$. Given sets $A_1, \ldots, A_n$ (for some $n \in \mathbb{N}$) by $A_1 \times \cdots \times A_n$ we denote the *cartesian product*:

$$A_1 \times \cdots \times A_n = \{(a_1, \ldots, a_n) \mid a_i \in A_i \text{ for all } 1 \leqslant i \leqslant n\}$$

An *n-ary relation* R over $A_1, \ldots, A_n$ is a subset of $A_1 \times \cdots \times A_n$. If $n = 2$ then R is called a *binary relation* and if $R \subseteq A \times A$ then we say that R is a binary relation over A. For a binary relation R its *inverse* is defined by

$R^{-1} = \{(b, a) \mid (a, b) \in R\}$.[1] The *identity relation* id_A on A is $\{(a, a) \mid a \in A\}$. Given two binary relations $R \subseteq A \times B$ and $S \subseteq B \times C$ we write $R \cdot S$ for their *composition*:

$$R \cdot S = \{(a, c) \mid (a, b) \in R \text{ and } (b, c) \in S \text{ for some } b \in B\}$$

Let R be a binary relation over A. For $n \in \mathbb{N}$ by R^n we denote the *n-fold composition* of R, i.e., $R^0 = \mathsf{id}_A$ and $R^{n+1} = R \cdot R^n$. The *reflexive closure* of R is $R^= = R \cup \mathsf{id}_A$, the *transitive closure* is $R^+ = \bigcup_{n \geqslant 1} R^n$, the *reflexive transitive closure* is $R^* = R^+ \cup \mathsf{id}_A$, and the *symmetric closure* of R is $R \cup R^{-1}$.

The *multiset extension* of a binary relation $>$, denoted $>_{\mathsf{mul}}$, is defined as $X \uplus Y >_{\mathsf{mul}} X \uplus Z$ if $Y \neq \varnothing$ and for all $z \in Z$ there is a $y \in Y$ such that $y > z$, where $\uplus$ denotes the sum of multisets.

2.2 Abstract Rewrite Systems

To model computations in an abstract way we consider a set of objects together with a binary relation that describes transformations on these objects.

Definition 2.1. An *abstract rewrite system* (ARS) is a pair $\mathcal{A} = (A, R)$ consisting of a set A and a binary relation R on A.

In IsaFoR, an ARS is just is a binary relation where the domain is left implicit in the type. When no confusion will arise we will sometimes also do this in textual descriptions and identify an ARS with its binary relation. To indicate that rewriting is a directed transformation process the relation R is usually written as $\to$ and instead of $(a, b) \in \to$ we write $a \to b$ and call $a \to b$ a *rewrite step*. Taking the transitive reflexive closure of $\to$ yields rewrite sequences. A *finite rewrite sequence* is a non-empty sequence $(a_1, \ldots, a_n)$ of elements in A such that $a_i \to a_{i+1}$ for all $1 \leqslant i < n$. Given objects $a, b \in A$ we say that a *rewrites* to b if $a \to^* b$ and call b a *reduct* of a or *reachable* from a. If there is a rewrite step from a to some b we say that a is *reducible*. Objects that are not reducible are called *normal forms* and we write $\mathsf{NF}(\mathcal{A})$ or $\mathsf{NF}(\to)$ for the set of all normal forms of $\mathcal{A}$. If a has the normal form b, i.e., if $a \to^* b$ and $b \in \mathsf{NF}(\to)$ we write $a \to^! b$. Objects a and b are *joinable*, written $a \downarrow b$,

[1] When convenient we mirror the notation of a relation to denote its inverse, for instance we write $\leqslant$ instead of $\geqslant^{-1}$.

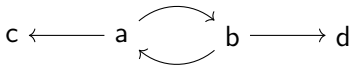


Figure 2: An abstract rewrite system.

if they have a common reduct, i.e., if there is an object c with $a \to^* c \,{}^*\!\leftarrow b$. They are *meetable* if they have a common ancestor, i.e., meetability is defined as $\uparrow = {}^*\!\leftarrow \cdot \to^*$. We write $\leftrightarrow$ for the symmetric closure of $\to$. A *conversion* between objects a and b is a sequence $(c_1, \ldots, c_n)$ such that $c_1 = a$, $c_n = b$, and $c_i \leftrightarrow c_{i+1}$ for all $1 \leqslant i < n$. We then write $a \leftrightarrow^* b$ and say that a and b are *convertible*. An *infinite rewrite sequence* is an infinite sequence of objects $(a_i)_{i \in \mathbb{N}}$ such that $a_i \to a_{i+1}$ for all $i \in \mathbb{N}$.

Example 2.2. Consider the ARS $\mathcal{A} = (A, \to)$ with $A = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$ and $\to = \{(\mathsf{a}, \mathsf{b}), (\mathsf{b}, \mathsf{a}), (\mathsf{a}, \mathsf{c}), (\mathsf{b}, \mathsf{d})\}$, depicted as directed graph in Figure 2. In this ARS we have for example $\mathsf{a} \to^! \mathsf{d}$, $\mathsf{a} \to^* \mathsf{a}$, and $\mathsf{b} \downarrow \mathsf{c}$. The objects c and d are convertible normal forms.

These basic definitions are enough to discuss many of the commonly studied global properties of ARSs. When an ARS has a property we also often say that the underlying relation has that property, leaving the set of objects implicit.

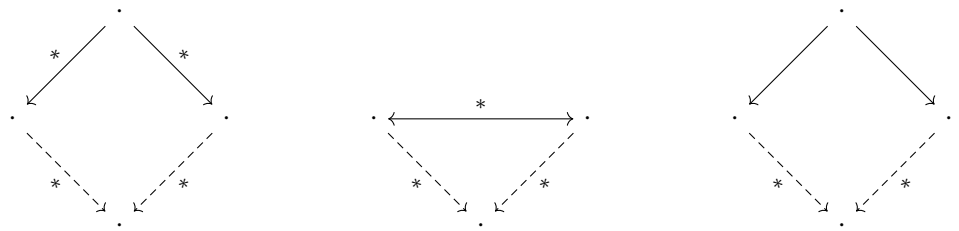
Definition 2.3. An ARS is *terminating*, if it admits no infinite rewrite sequences.

The ARS from Example 2.2 is not terminating, because the sequence $\mathsf{a} \to \mathsf{b} \to \mathsf{a} \to \mathsf{b} \to \cdots$ constitutes an infinite rewrite sequence. We now turn to confluence and various related properties.

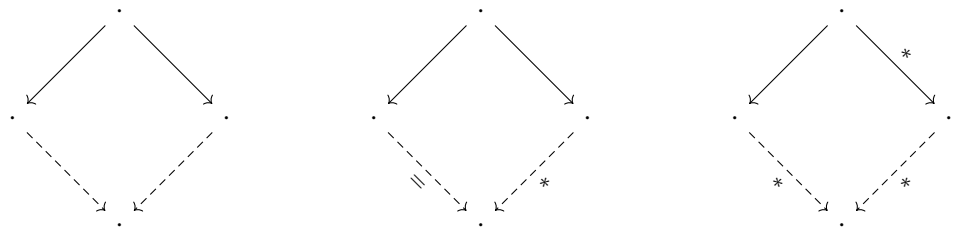
Definition 2.4. An ARS $\mathcal{A} = (A, \to)$ is *confluent*, if whenever $t \,{}^*\!\leftarrow s \to^* u$ then there is an object v such that $t \to^* v \,{}^*\!\leftarrow u$, or more succinctly, $\to$ is confluent if

$$ {}^*\!\leftarrow \cdot \to^* \subseteq \to^* \cdot {}^*\!\leftarrow $$

Confluence is commonly drawn as the diagram shown in Figure 3(a), where solid lines indicate universal quantification and dashed lines indicate existential quantification. In light of this depiction a situation $s \,{}^*\!\leftarrow \cdot \to^* t$ is called a *peak*. Such a peak is called *joinable* if its endpoints s and t are joinable, i.e., if there is a so-called *valley* $s \to^* \cdot \,{}^*\!\leftarrow t$.



(a) Confluence. (b) Church-Rosser. (c) Local confluence.



(d) Diamond. (e) Strong confluence. (f) Semi-confluence.

Figure 3: Confluence and various related properties.

When it was originally studied confluence was stated in the following different but equivalent formulation, named after Alonzo Church and J. Barkley Rosser, who first showed the property for the λ-calculus.

Definition 2.5. An ARS $\mathcal{A} = (A, \to)$ has the *Church-Rosser* property (CR) if

$$\leftrightarrow^* \subseteq \to^* \cdot {}^*\!\leftarrow$$

The Church-Rosser property is depicted in Figure 3(b). That it is equivalent to confluence is not hard to show.

Lemma 2.6. *An ARS is confluent if and only if it has the Church-Rosser property.*

Proof. That CR implies confluence is trivial, since every peak obviously is a conversion. For the other direction assume $s \leftrightarrow^n t$ and proceed by induction on n. If $n = 0$ then $s = t$ and obviously $s \downarrow t$. Otherwise write $s \leftrightarrow t \leftrightarrow^n u$. From the induction hypothesis we obtain v with $t \to^* v \, {}^*\!\leftarrow u$. If $s \to t$ this

immediately yields $s \to^* v \,{}^*\!\leftarrow u$. If $s \leftarrow t$ then we have the peak $s \leftarrow t \to^* v$ and by confluence $s \downarrow v$, which with $u \to^* v$ yields the desired $s \downarrow u$. □

Because we usually do not deal with finite ARSs, there will in general be infinitely many peaks to check when we try to establish confluence. Since we are interested in automatic confluence checking we will need to break them down in some finite criteria. A first idea, on the abstract level, might be to localize the test, i.e., to only consider peaks consisting of single steps, so-called *local peaks*.

Definition 2.7. An ARS $\mathcal{A} = (A, \to)$ is *locally confluent* if

$$\leftarrow \cdot \to \subseteq \to^* \cdot {}^*\!\leftarrow$$

The diagram for local confluence, also known as the *weak Church-Rosser* property, is drawn in Figure 3(c). Unfortunately joinability of all local peaks is not sufficient for confluence in general.

Example 2.8. The ARS from Example 2.2 is locally confluent: its two non-trivial local peaks $\mathsf{c} \leftarrow \mathsf{a} \to \mathsf{b}$ and $\mathsf{a} \leftarrow \mathsf{b} \to \mathsf{d}$ are both joinable. It is however not confluent, since it admits the peak $\mathsf{d} \,{}^*\!\leftarrow \mathsf{a} \to^* \mathsf{c}$, but c and d are not joinable.

When trying to impose restrictions on an ARS to still get a relationship between local confluence and confluence one realizes that the problem goes away for terminating systems. This is captured in the following famous lemma, due to Newman.

Lemma 2.9 (Newman [69])**.** *A terminating ARS is confluent if it is locally confluent.*

Proof. Assume $t \,{}^*\!\leftarrow s \to^* u$, we perform well-founded induction on s with respect to $\to$, which is well-founded, since the ARS in question is terminating, to show $t \downarrow u$. If $s = t$ or $s = u$ then trivially $t \downarrow u$, so assume there is at least one step on both sides: $t \,{}^*\!\leftarrow t' \leftarrow s \to u' \to^* u$. From local confluence we obtain $t' \to^* v \,{}^*\!\leftarrow u'$ for some v. This yields the new peak $t \,{}^*\!\leftarrow t' \to^* v$. Since $s \to t'$ we can apply the induction hypothesis and obtain a w with $t \to^* w \,{}^*\!\leftarrow v$. We are left with the peak $w \,{}^*\!\leftarrow v \,{}^*\!\leftarrow u' \to^* u$, which, since $s \to u'$, we again close using the induction hypothesis. □

So for terminating systems we can reduce showing confluence to local confluence.[2] However many systems of interest will not be terminating and so we need a different approach. A common one is to restrict attention to local peaks by also restricting the joining sequences, employing one of the following two properties.

Definition 2.10. An ARS $\mathcal{A} = (A, \to)$ has the *diamond property* if

$$\leftarrow \cdot \rightarrow \subseteq \rightarrow \cdot \leftarrow$$

It is *strongly confluent* if

$$\leftarrow \cdot \rightarrow \subseteq \rightarrow^{=} \cdot {}^{*}\leftarrow$$

The diamond property and strong confluence are depicted in Figures 3(d) and 3(e) respectively. Clearly a relation is confluent if its reflexive transitive closure has the diamond property and any relation that has the diamond property is strongly confluent. Beware of symmetry in the diagram for strong confluence. If there is a peak $t \leftarrow s \rightarrow u$, and consequently a valley $t \rightarrow^{=} \cdot {}^{*}\leftarrow u$, then there is also the symmetric peak $u \leftarrow s \rightarrow t$ and hence there must be a valley $u \rightarrow^{=} \cdot {}^{*}\leftarrow t$, making strong confluence only slightly weaker than requiring $\leftarrow \cdot \rightarrow \subseteq \rightarrow^{=} \cdot {}^{=}\leftarrow$. The next lemma justifies the name strong confluence.

Lemma 2.11. *Any strongly confluent ARS is confluent.* □

We defer the proof of this statement for a bit and will instead carry it out in the more general setting of commutation. Commutation is the natural generalization of confluence to two relations.

Definition 2.12. Two relations $\rightarrow_1$ and $\rightarrow_2$ *commute* if

$$ {}^{*}_{1}\leftarrow \cdot \rightarrow_2^{*} \subseteq \rightarrow_2^{*} \cdot {}^{*}_{1}\leftarrow$$

They *locally commute* if

$$ {}_{1}\leftarrow \cdot \rightarrow_2 \subseteq \rightarrow_2^{*} \cdot {}^{*}_{1}\leftarrow$$

They *strongly commute* if

$$ {}_{1}\leftarrow \cdot \rightarrow_2 \subseteq \rightarrow_2^{=} \cdot {}^{*}_{1}\leftarrow$$

[2] In fact when moving to term rewriting we will see that local confluence and consequently also confluence are even decidable for terminating systems.

Obviously confluence of a relation is equivalent to it commuting with itself. Like for confluence we will want to restrict attention to local peaks by showing strong commutation. Note that strong commutation is not symmetric, i.e., strong commutation of $\to_1$ and $\to_2$ does not imply strong commutation of $\to_2$ and $\to_1$.

Example 2.13. Consider the two ARSs $\to_1$ and $\to_2$ defined by the following steps:

$$\mathsf{a} \to_1 \mathsf{b} \qquad \mathsf{c} \to_1 \mathsf{d} \qquad \mathsf{d} \to_1 \mathsf{b} \qquad \mathsf{a} \to_2 \mathsf{c}$$

Then $\to_1$ and $\to_2$ strongly commute because the peak $\mathsf{b} \,{}_1\!\leftarrow \mathsf{a} \to_2 \mathsf{c}$ is joinable as $\mathsf{b} \,{}_1\!\leftarrow \mathsf{d} \,{}_1\!\leftarrow \mathsf{c}$. However, for the symmetric peak $\mathsf{c} \,{}_2\!\leftarrow \mathsf{a} \to_1 \mathsf{b}$ there is no valley of the shape $\to_1^= \cdot \,{}^*_2\!\leftarrow$, since $\mathsf{b} \in \mathsf{NF}(\to_2)$ and the only step from c with $\to_1$ is $\mathsf{c} \to_1 \mathsf{d}$. Hence $\to_2$ and $\to_1$ do not strongly commute.

To show that strong commutation implies commutation we proceed in two steps—we first partially localize the diagram.

Definition 2.14. Two relations $\to_1$ and $\to_2$ *semi-commute* if

$$\,{}^*_1\!\leftarrow \cdot \to_2 \subseteq \to_2^* \cdot \,{}^*_1\!\leftarrow$$

Instantiating $\to_1$ and $\to_2$ to the same relation yields the related notion of *semi-confluence*, shown in Figure 3(f). Semi-commutation turns out to be equivalent to commutation.

Lemma 2.15. *Two binary relations $\to_1$ and $\to_2$ semi-commute if and only if they commute.*

Proof. That commutation implies semi-commutation is immediate from the definitions. For the other direction assume $t \,{}^*_1\!\leftarrow s \to_2^* u$. Then there is an n such that $s \to_2^n u$. We show $t \to_2^* \cdot \,{}^*_1\!\leftarrow u$ by induction on n. If $n = 0$ then $s = u$ and we are done. Otherwise write $t \,{}^*_1\!\leftarrow s \to_2 u' \to_2^n u$. From semi-commutation we obtain a v with $t \to_2^* v \,{}^*_1\!\leftarrow u'$. This leaves us with the peak $v \,{}^*_1\!\leftarrow u' \to_2^n u$, which, by the induction hypothesis, can be joined as $v \to_2^* \cdot \,{}^*_1\!\leftarrow u$. Together with $t \to_2^* v$ this yields the desired $t \to_2^* \cdot \,{}^*_1\!\leftarrow u$. □

Note that Lemma 2.15 entails that, unlike strong commutation, semi-commutation is symmetric, despite the asymmetry in its definition.

Lemma 2.16. *Let $\to_1$ and $\to_2$ be binary relations. If $\to_1$ and $\to_2$ strongly commute then they commute.*

Proof. We show semi-commutation of $\to_1$ and $\to_2$. Assume $t \,{}^*_1\!\leftarrow s \to_2 u$. Then there is a natural number n with $t \,{}^n_1\!\leftarrow s \to_2 u$. We show $t \to_2^* \cdot \,{}^*_1\!\leftarrow u$ by induction on n. If $n = 0$ then $s = t$ and we are done. Otherwise we have $t \,{}^n_1\!\leftarrow t' \,{}_1\!\leftarrow s \to_2 u$. Strong commutation yields a v with $t' \to_2^= v \,{}^*_1\!\leftarrow u$. We consider two cases. If $t' = v$ then $u \to_1^* t' \to_1^* t$. Otherwise $t' \to_2 v$ and we close the peak $t \,{}^n_1\!\leftarrow t' \to_2 v$ by the induction hypothesis as $t \to_2^* \cdot \,{}^*_1\!\leftarrow v$ and using $u \to_1^* v$. □

To conclude commutation from strong commutation it is often useful to not use the relations in question directly but auxiliary relations that lie between one-step and many-step rewriting. The following lemma shows how to achieve this.

Lemma 2.17. *Let $\to_1$, $\to_2$, $\to_{1'}$, and $\to_{2'}$ be binary relations. If $\to_1 \subseteq \to_{1'} \subseteq \to_1^*$ and $\to_2 \subseteq \to_{2'} \subseteq \to_2^*$ and $\to_{1'}$ and $\to_{2'}$ commute then $\to_1$ and $\to_2$ commute.*

Proof. Straightforward induction proofs show $\to_1^* = \to_{1'}^*$ and $\to_2^* = \to_{2'}^*$. Then commutation of $\to_1$ and $\to_2$ follows from commutation of $\to_{1'}$ and $\to_{2'}$. □

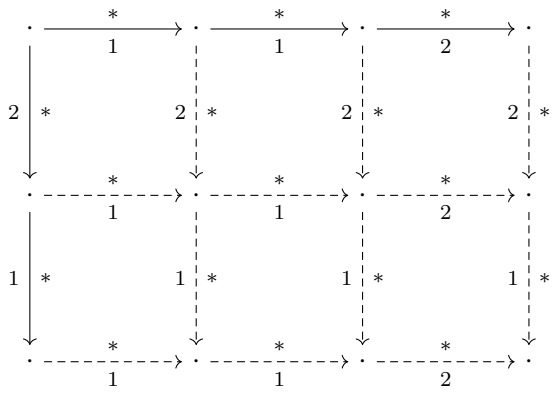
Instantiating commutation to confluence yields a corresponding useful corollary.

Corollary 2.18. *Let $\to_1$ and $\to_2$ be binary relations. If $\to_1 \subseteq \to_2 \subseteq \to_1^*$ and $\to_2$ is confluent then $\to_1$ is confluent.* □

The following important connection between confluence and commutation allows to obtain confluence of the union of two confluent relations provided they commute.

Lemma 2.19 (Hindley [38])**.** *Let $\to_1$ and $\to_2$ be confluent relations. If $\to_1$ and $\to_2$ commute then $\to_1 \cup \to_2$ is confluent.*

Proof. By splitting the rewrite sequences in a peak $u \,{}^*_{1\cup 2}\!\leftarrow s \to_{1\cup 2}^* t$ and repeatedly filling the diagrams using the confluence and commutation assumptions:



□

We now turn our attention to concrete instantiations of abstract rewriting. The first one we consider results from using first-order terms as objects and a rewrite relation generated from a set of rules.

2.3 Term Rewriting

A *signature* $\mathcal{F}$ is a set of function symbols. Each function symbol $f \in \mathcal{F}$ comes with an associated *arity* $\mathsf{ar}(f) \in \mathbb{N}$. Function symbols with arity 0 are called *constants*. With $\mathcal{V}$ we denote an infinite set of *variables* disjoint from $\mathcal{F}$.

Definition 2.20. The set of *terms* $\mathcal{T}(\mathcal{F}, \mathcal{V})$ over a signature $\mathcal{F}$ and a set of variables $\mathcal{V}$, is defined inductively by

- if $x \in \mathcal{V}$ then $x \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, and
- if $f \in \mathcal{F}$ with $\mathsf{ar}(f) = n$ and $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ for $1 \leqslant i \leqslant n$ then $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$.

We write $\mathcal{V}\mathsf{ar}(t)$ for the set of variables occurring in the term t and denote the *number of occurrences* of the variable x in term t by $|t|_x$.

In IsaFoR variables, function, symbols, and terms are represented as types instead of sets. The datatype for terms is given in Listing 1, showing how terms are built over function symbols of type α and variables of type β. The main difference to the textbook definition is that there is no arity restriction on function applications. Consequently the name of a function symbol alone, is not enough to identify it uniquely, since the same name could be used with

datatype (α, β) *term* = Var β | Fun α (α, β) *term list*

Listing 1: First-order terms in IsaFoR.

different arities. Thus, whenever the signature is essential IsaFoR uses pairs (f, n) of function symbols and arities.

We often need to address specific subterms of a term. To this end we use positions, strings that encode the path from the root of the term to the subterm in question.

Definition 2.21. *Positions* are strings of positive natural numbers, i.e., elements of $\mathbb{N}_+^*$. We denote the *root position*, i.e., the empty sequence, by ϵ. A position q is *above* a position p, written $q \leqslant p$, if $qq' = p$ for some position q', in which case $p \backslash q$ is defined to be q'. Furthermore q is *strictly above* p, written as $q < p$, if $q \leqslant p$ and $q \neq p$. If q is above p we also say that p is *below* q. Finally, positions q and p are *parallel*, written as $q \parallel p$, if neither $q \leqslant p$ nor $p < q$.

Definition 2.22. The *set of positions in a term* $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ is defined as

$$\mathsf{Pos}(t) = \begin{cases} \{\epsilon\} & \text{if } t \text{ is a variable} \\ \{\epsilon\} \cup \{iq \mid 1 \leqslant i \leqslant n \text{ and } q \in \mathsf{Pos}(t_i)\} & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

The subterm of t at position $p \in \mathsf{Pos}(t)$ is defined as

$$t|_p = \begin{cases} t & \text{if } p = \epsilon \\ t_i|_q & \text{if } p = iq \text{ and } t = f(t_1, \ldots, t_n) \end{cases}$$

The set of *function symbol positions* of t is $\mathsf{Pos}_{\mathcal{F}}(t) = \{p \in \mathsf{Pos}(t) \mid t|_p \notin \mathcal{V}\}$ and the set of *variable positions* of t is $\mathsf{Pos}_{\mathcal{V}}(t) = \mathsf{Pos}(t) \setminus \mathsf{Pos}_{\mathcal{F}}(t)$. The *size* of t is defined as the size of $\mathsf{Pos}(t)$ and denoted by $|t|$.

A central operation on terms is replacing subterms. To this end we use contexts, terms containing holes that can be filled by other terms.

Definition 2.23. Let $\Box$ be a constant symbol with $\Box \notin \mathcal{F}$, called *hole*. A *context* is a term in $\mathcal{T}(\mathcal{F} \cup \{\Box\}, \mathcal{V})$, with exactly one occurrence of $\Box$. A *multihole context* is a term that may contain an arbitrary number of holes.

datatype (α, β) *ctxt* = Hole
| More α (α, β) *term list* (α, β) *ctxt* (α, β) *term list*

datatype (α, β) *mctxt* = MVar β | MHole | MFun α (α, β) *mctxt list*

Listing 2: Contexts and multihole contexts in IsaFoR.

In IsaFoR contexts and multihole contexts are not defined as special terms but as separate datatypes as shown in Listing 2. This facilitates proofs by induction and directly enforces the restriction that contexts contain exactly one hole. On the other hand special conversion functions between, for example multihole contexts without holes and terms, are now required, but in a formalization environment the cleaner separation is worth the effort.

Definition 2.24. For a term t and position $p \in \mathsf{Pos}(t)$ the context obtained by replacing $t|_p$ with the hole, $t[]_p$, is defined by

$$t[]_p = \begin{cases} \Box & \text{if } p = \epsilon \\ f(t_1, \ldots, t_i[]_q, \ldots, t_n) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } p = iq \end{cases}$$

Definition 2.25. For a context C and term t by $C[t]$ we denote the term obtained by filling the hole in C with t:

$$C[t] = \begin{cases} t & \text{if } C = \Box \\ f(t_1, \ldots, C'[t], \ldots, t_n) & \text{if } C = f(t_1, \ldots, C', \ldots, t_n) \end{cases}$$

Moreover $t[s]_p$ denotes the term obtained by replacing the subterm of t at position p by s, i.e., $t[s]_p = (t[]_p)[s]$.

If $C[s] = t$ for some context C then s is called a *subterm* of t and we write $s \trianglelefteq t$. If additionally $C \neq \Box$ then s is a *proper subterm* of t, which is denoted by $s \triangleleft t$.

Filling the holes in a multihole context C with terms $t_1, \ldots, t_n$ is more involved, because we have to partition the list of terms according to the distribution of holes in C.

Definition 2.26. For a multihole context C containing n holes and a list of

terms $t_1, \ldots, t_n$ we define $C[t_1, \ldots, t_n]$ by the following equations:

$$\begin{aligned} \Box[t] &= t \\ x[] &= x \\ f(C_1, \ldots, C_c)[t_1, \ldots, t_n] &= f(C_1[\bar{t}_1], \ldots, C_c[\bar{t}_c]) \end{aligned}$$

where $\bar{t}_1, \ldots, \bar{t}_c$ is a partitioning of $t_1, \ldots, t_n$ such that the length of $\bar{t}_i$ matches the number of holes in C_i for all $1 \leqslant i \leqslant c$.

Dealing with this definition and multihole contexts in IsaFoR requires some non-trivial overhead. One needs to take care of for instance partitioning, flattening, or shortening lists of terms. While cumbersome and time-consuming the reasoning involved usually does not yield new insights and so we will not elaborate on in the text.

We now turn to instantiating variables in terms.

Definition 2.27. A *substitution* is a mapping σ from $\mathcal{V}$ to $\mathcal{T}(\mathcal{F}, \mathcal{V})$. We write $t\sigma$ for the result of applying σ to the term t:

$$t\sigma = \begin{cases} \sigma(x) & \text{if } t = x \\ f(t_1\sigma, \ldots, t_n\sigma) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

For terms s and t, we call s an *instance* of t and say that t *matches* s if there is a substitution σ such that $s = t\sigma$. A substitution μ is *unifier* of s and t if $s\mu = t\mu$ in which case s and t are called *unifiable*. A *most general unifier* μ is a unifier with $\sigma = \mu\rho$ for some ρ for any other unifier σ.

Finite substitutions, i.e., mappings with $x \neq \sigma(x)$ for finitely many $x \in \mathcal{V}$, are often written in the form $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$. When $\sigma = \{x \mapsto s\}$ we also write $t\,[x := s]$ for $t\sigma$.

Relations that are preserved under contexts and under substitutions are of special interest.

Definition 2.28. A binary relation R on terms is *closed under contexts* if $s\ R\ t$ implies $C[s]\ R\ [t]$ for all terms s and t and all contexts C. It is *closed under substitutions* if for all substitutions σ we have $s\sigma\ R\ t\sigma$ whenever $s\ R\ t$. A *rewrite relation* is a binary relation on terms that is closed under both contexts and substitutions.

$$\frac{(\ell,\, r) \in \mathcal{R}}{(\ell,\, r) \in \textbf{\textit{rstep}}\ \mathcal{R}} \qquad \frac{(s,\, t) \in \textbf{\textit{rstep}}\ \mathcal{R}}{(s \cdot \sigma,\, t \cdot \sigma) \in \textbf{\textit{rstep}}\ \mathcal{R}} \qquad \frac{(s,\, t) \in \textbf{\textit{rstep}}\ \mathcal{R}}{(C\langle s\rangle,\, C\langle t\rangle) \in \textbf{\textit{rstep}}\ \mathcal{R}}$$

Listing 3: The definition of rewriting in IsaFoR.

$$\textbf{\textit{rstep\_r\_c\_s}}\ r\ C\ \sigma = \{(s,\, t).\ s = C\langle \textbf{\textit{fst}}\ r \cdot \sigma\rangle \wedge t = C\langle \textbf{\textit{snd}}\ r \cdot \sigma\rangle\}$$

$$\textbf{\textit{rstep\_r\_p\_s}}\ \mathcal{R}\ r\ p\ \sigma =$$
$$\{(s,\, t).\ p \in \textbf{\textit{poss}}\ s \wedge r \in \mathcal{R} \wedge s\ |\_\ p = \textbf{\textit{fst}}\ r \cdot \sigma \wedge t = (s\ []\_\ p)\langle \textbf{\textit{snd}}\ r \cdot \sigma\rangle\}$$

Listing 4: Alternative characterizations of rewriting in IsaFoR.

We are now ready to define term rewriting.

Definition 2.29. A *rewrite rule* is a pair of terms (ℓ, r), written $\ell \to r$. A *term rewrite system* (TRS) is set of rewrite rules. For a TRS $\mathcal{R}$ we define $\to_{\mathcal{R}}$ to be the smallest rewrite relation that contains $\mathcal{R}$ and associate with it the ARS $(\mathcal{T}(\mathcal{F}, \mathcal{V}), \to_{\mathcal{R}})$.

Put differently, a term s rewrites to a term t using the TRS $\mathcal{R}$ if we can find a rule $\ell \to r$ in $\mathcal{R}$, a context C and a substitution σ such that $s = C[\ell\sigma]$ and $t = C[r\sigma]$. Equivalently, we have $s \to_{\mathcal{R}} t$ if $s|_p = \ell\sigma$ and $t|_p = r\sigma$, for a rule $\ell \to r$ in $\mathcal{R}$, a position p, and a substitution σ. When the rewrite step takes place at the root, i.e., $p = \epsilon$, we write $s \to^{\epsilon}_{\mathcal{R}} t$. Similar to ARSs we will often identify a TRS $\mathcal{R}$ with its rewrite relation $\to_{\mathcal{R}}$. Note that we do not impose the common *variable conditions*, i.e., the restriction that ℓ is not a variable and all variables in r are contained in ℓ.

In IsaFoR $\to_{\mathcal{R}}$ is defined as an inductive set by the rules show in Listing 3, where ***rstep*** $\mathcal{R}$ is IsaFoR's notation for $\to_{\mathcal{R}}$. Alternative characterizations, based on finding a redex in a term are also available, see Listing 4.

Many confluence results depend on variables occurring with restricted multiplicity in rewrite rules.

Definition 2.30. A term t is *linear* if every variable occurs at most once in it. A rewrite rule $\ell \to r$ is *left-linear* if ℓ is linear, *right-linear* if r is linear, and *linear* if it is both left- and right-linear. A TRS is *(left-, right-)linear* if all its rules are (left-, right-)linear. A rewrite rule $\ell \to r$ is *duplicating* if $|\ell|_x < |r|_x$ for some $x \in \mathcal{V}$. By $\mathcal{R}_{\mathsf{d}}$ and $\mathcal{R}_{\mathsf{nd}}$, we denote the *duplicating* and *non-duplicating rules* of a TRS $\mathcal{R}$, respectively.

We will sometimes consider rewriting with a TRS $\mathcal{R}$ relative to some other TRS $\mathcal{S}$, which intuitively means that arbitrarily many $\mathcal{S}$-steps are allowed between $\mathcal{R}$-steps.

Definition 2.31. A *relative TRS* $\mathcal{R}/\mathcal{S}$ is a pair of TRSs $\mathcal{R}$ and $\mathcal{S}$ with the induced rewrite relation $\to_{\mathcal{R}/\mathcal{S}} = \to_{\mathcal{S}}^* \cdot \to_{\mathcal{R}} \cdot \to_{\mathcal{S}}^*$.

Sometimes we identify a TRS $\mathcal{R}$ with the relative TRS $\mathcal{R}/\varnothing$ and vice versa, which is justified by $\to_{\mathcal{R}/\varnothing} = \to_{\mathcal{R}}$.

The extra structure of terms as objects allows to define rewrite relations that lie between one and many step rewriting, which will be useful in connection with Lemma 2.17. We will consider two such relations, the first allows to contract multiple parallel redexes in one go.

Definition 2.32. For a TRS $\mathcal{R}$, the *parallel rewrite relation* $\mathrel{+\!\!\!\twoheadrightarrow_{\mathcal{R}}}$ is defined inductively by

- $x \mathrel{+\!\!\!\twoheadrightarrow_{\mathcal{R}}} x$ if x is a variable,
- $\ell\sigma \mathrel{+\!\!\!\twoheadrightarrow_{\mathcal{R}}} r\sigma$ if $\ell \to r \in \mathcal{R}$, and
- $f(s_1, \ldots, s_n) \mathrel{+\!\!\!\twoheadrightarrow_{\mathcal{R}}} f(t_1, \ldots, t_n)$ if f is a function symbol of arity n and $s_i \mathrel{+\!\!\!\twoheadrightarrow_{\mathcal{R}}} t_i$ for all $1 \leqslant i \leqslant n$.

The following properties of parallel rewriting are well-known and follow by straightforward induction proofs.

Lemma 2.33. *The following properties of* $\mathrel{+\!\!\!\twoheadrightarrow}$ *hold:*

- $\to_{\mathcal{R}} \subseteq \mathrel{+\!\!\!\twoheadrightarrow_{\mathcal{R}}} \subseteq \to_{\mathcal{R}}^*$,
- $s \mathrel{+\!\!\!\twoheadrightarrow_{\mathcal{R}}} s$ *for all terms* s,
- *if* $x\sigma \mathrel{+\!\!\!\twoheadrightarrow_{\mathcal{R}}} x\tau$ *for all* $x \in \mathcal{V}\mathsf{ar}(s)$ *then* $s\sigma \mathrel{+\!\!\!\twoheadrightarrow_{\mathcal{R}}} s\tau$. □

When not only allowing parallel redexes to be contracted simultaneously, but also nested ones we get what are commonly known as *multisteps* or *development steps*. For a binary relation R on terms we write $\sigma \mathrel{R} \sigma'$ if $\sigma(x) \mathrel{R} \sigma'(x)$ for all variables x.

Definition 2.34. For a TRS $\mathcal{R}$ the *multistep rewrite relation* $\mathrel{\circ\!\!\!\twoheadrightarrow_{\mathcal{R}}}$ is defined inductively by

- $x \mathrel{-\!\!\!\twoheadrightarrow_{\mathcal{R}}} x$ if x is a variable,
- $\ell\sigma \mathrel{-\!\!\!\twoheadrightarrow_{\mathcal{R}}} r\sigma'$ if $\ell \to r \in \mathcal{R}$ and σ, σ' are substitutions with $\sigma \mathrel{-\!\!\!\twoheadrightarrow_{\mathcal{R}}} \sigma'$, and
- $f(s_1, \ldots, s_n) \mathrel{-\!\!\!\twoheadrightarrow_{\mathcal{R}}} f(t_1, \ldots, t_n)$ if f is a function symbol of arity n and $s_i \mathrel{-\!\!\!\twoheadrightarrow_{\mathcal{R}}} t_i$ for $1 \leqslant i \leqslant n$.

Most of the confluence and commutation results in this work are based on (left-)linearity and restricted joinability of critical pairs. Critical pairs arise from situations where two redexes overlap with each other. The definition we use is slightly non-standard in two regards. First we consider critical pairs for two rewrite systems to use them in a commutation setting. Second we do not exclude root overlaps of a rule with (a variant of) itself as is commonly done. This allows us to dispense with the variable condition that all variables in the right-hand side of a rule must also occur on the left. Moreover, if a TRS does satisfy the condition then all extra critical pairs that would normally be excluded are trivial.

Definition 2.35. A *critical overlap* $(\ell_1 \to r_1, C, \ell_2 \to r_2)_\mu$ of two TRSs $\mathcal{R}_1$ and $\mathcal{R}_2$ consists of variants $\ell_1 \to r_1$ and $\ell_2 \to r_2$ of rewrite rules in $\mathcal{R}_1$ and $\mathcal{R}_2$ without common variables, a context C, such that $\ell_2 = C[\ell']$ with $\ell' \notin \mathcal{V}$ and a most general unifier μ of ℓ_1 and ℓ'. From a critical overlap $(\ell_1 \to r_1, C, \ell_2 \to r_2)_\mu$ we obtain a *critical peak* $C\mu[r_1\mu] \mathrel{{}_{\mathcal{R}_1}\!\leftarrow} C\mu[\ell_1\mu] \mathrel{\to_{\mathcal{R}_2}} r_2\mu$ and the corresponding *critical pair* $C\mu[r_1\mu] \mathrel{{}_{\mathcal{R}_1}\!\leftarrow\!\rtimes\!\rightarrow_{\mathcal{R}_2}} r_2\mu$. If $C = \Box$, the corresponding critical pair is called an *overlay* and written as $r_1\mu \mathrel{{}_{\mathcal{R}_1}\!\leftarrow\!\bowtie\!\rightarrow_{\mathcal{R}_2}} r_2\mu$, otherwise it is called an *inner critical pair*, and denoted using $\mathrel{{}_{\mathcal{R}_1}\!\leftarrow\!\rtimes\!\rightarrow_{\mathcal{R}_2}}$. When considering the critical pairs of a TRS $\mathcal{R}$ with itself we drop the subscripts and write $\leftarrow\!\rtimes\!\rightarrow$ instead of $\mathrel{{}_{\mathcal{R}}\!\leftarrow\!\rtimes\!\rightarrow_{\mathcal{R}}}$. We call a critical pair *joinable* if the peak from which it originates is joinable.

The next example shows that if the variable condition is not satisfied, critical pairs that arise from overlapping a rule with itself at the root are essential.

Example 2.36. Consider the linear rewrite system $\mathcal{R}$ consisting of the single rule $\mathsf{a} \to y$. Because of the peak $x \leftarrow \mathsf{a} \to y$, $\mathcal{R}$ is not confluent and indeed $x \leftarrow\!\rtimes\!\rightarrow y$ is a non-joinable critical pair according to our definition.

2.4 Lambda Calculus

We now turn to higher-order systems, i.e., systems with bound variables and functions as first-class citizens. The prototypical higher-order system is the lambda calculus, which we consider in its simply typed incarnation. In the next section it will then serve as meta-language for general higher-order rewriting.

For a (formalized) treatment of the untyped lambda calculus see Chapter 6, for a general introduction to the lambda calculus we refer to [12].

Definition 2.37. Given a set of *base types*, $\mathcal{B}$, the set of *simple types* over $\mathcal{B}$, denoted $\mathcal{T}_\mathcal{B}$ is defined inductively as follows:

- every base type $\iota \in \mathcal{B}$ is a simple type,
- if σ and τ are simple types, then $\sigma \to \tau$ is a simple type.

A type[3] with at least one occurrence of $\to$ is called functional. Following standard conventions $\to$ is right-associative and outermost parentheses are omitted. We denote types by $\sigma, \tau, \rho, \ldots$ and base types by ι, κ.

Let $\mathcal{V}$ be a set of typed variables, containing countably many variables of each type. When the type of variable can be inferred from the context or is of no interest we often omit it and write x instead of $x : \sigma$. Simply typed lambda terms are built from variables, λ-abstraction and application.

Definition 2.38. The set of *typed lambda terms*, $\Lambda^\to$, is defined inductively by the following rules:

$$\frac{x : \sigma \in \mathcal{V}}{x : \sigma \in \Lambda^\to} \qquad \frac{u : \tau \to \sigma \qquad t : \tau}{u\ t : \sigma \in \Lambda^\to} \qquad \frac{x : \tau \in \mathcal{V} \qquad t : \rho}{\lambda x.\, t : \tau \to \rho \in \Lambda^\to}$$

When $s : \sigma \in \Lambda^\to$ we say that s has type σ. The abstraction $\lambda x.\, s$ *binds* the variable x in s. A variable that is not bound is *free*. We collect all free variables of s in $\mathsf{fv}(s)$ and all bound variables in $\mathsf{bv}(s)$. A term is *linear* if no free variable occurs in it more that once. We adopt the usual conventions for notation: application is left associative and abstraction is right associative. We write $\lambda x_1\, x_2\, \ldots\, x_n.\, s$ instead of $\lambda x_1.\, \lambda x_2.\, \ldots\, \lambda x_n.\, s$.

[3] Since we do not consider any other kind of types, we just say type instead of simple type from now on.

Term equality is modulo α, that is, renaming of bound variables. Hence we tacitly assume bound variables to be fresh whenever necessary—the variable convention. If we want to state α-conversion explicitly we write $s =_\alpha t$, for instance $\lambda x.\, x =_\alpha \lambda y.\, y$.

Substitutions are like in the first-order setting, except they now take types and bound variables into account.

Definition 2.39. A *substitution* is a type-preserving mapping from variables to terms. The *capture-avoiding* application of a substitution σ to a term t is denoted by $t\sigma$.

Capture-avoiding means that we use α whenever necessary. For example $(\lambda x.\, y)\,[y := x] = (\lambda z.\, x)$, since $\lambda x.\, y =_\alpha \lambda z.\, y$. See Definition 6.1 for a formal definition of capture avoiding substitution. Contexts are also adapted to types.

Definition 2.40. We assume special symbols $\Box_\sigma$ for each type σ, called holes. A *context* is a term C containing exactly one occurrence of a hole $\Box_\sigma$. The replacement of $\Box_\sigma$ in a context C, by a term s of type σ is denoted by $C[s]$.

Note that here s may contain variables bound in C. For instance, for $C = \lambda x.\, \Box_\iota$ and $x : \iota$ we have $C[x] = \lambda x.\, x$. Lambda terms are rewritten using β-reduction and η-expansion.

Definition 2.41. The *β-rule* is defined by the following schema:

$$(\lambda x.\, s)\; t \to s\,[x := t]$$

Here the expression $(\lambda x.\, s)\; t$ is called β-redex. The *β-step* relation, denoted by $s \to_\beta t$, is the smallest relation the contains the β-rule and is closed under contexts.

It is well-known that β-reduction is terminating and confluent in the simply typed λ-calculus [13]. We denote the unique β-normal form of a term s by $s{\downarrow_\beta}$. Additionally we will use η-expansion. Since, in the presence of β-reduction, η-expansion is not terminating in general, we use a restricted version, which guarantees termination.

Definition 2.42. Let C be a context. The *η-expansion* relation is defined by

$$C[s] \to_\eta C[\lambda x.\, s\; x]$$

if s is of functional type $\sigma \to \tau$, $x : \sigma$ is a fresh variable, and no β-redex is created.

Avoiding creation of β-redexes ensures termination of $\to_\beta \cup \to_\eta$. Note that no β-redex is created if s is neither an abstraction nor the left-hand side of an application. That is, the two situations $s\ t \to_\eta (\lambda x.\ s\ x)\ t \to_\beta s\ t$ and $\lambda y.\ s[y] \to_\eta \lambda x.\ (\lambda y.\ s[y])\ x \to_\beta \lambda x.\ s[x] =_\alpha \lambda y.\ s[y]$ are avoided.

Using this η-expansion every term s has a unique η-long form, which we denote by $s{\uparrow^\eta}$. The unique η-long β-normal form of s is denoted by $s{\updownarrow^\eta_\beta}$. One can think of η-expansion as making the arguments of a term with functional type explicit by using extensionality. In particular for a term s of type $\sigma_1 \to \cdots \to \sigma_n \to \tau$ we have $s{\uparrow^\eta} = \lambda x_1, \ldots\ x_n.\ s\ x_1 \cdots x_n$, which when applied to arguments $t_1, \ldots, t_n$, β-reduces to the expected $s\ t_1 \cdots t_n$. The ARS associated with the simply typed lambda calculus is $(\Lambda^\to, \to_\beta \cup \to_\eta)$.

Based on the simply typed λ-calculus we now introduce higher-order rewriting, where terms additionally contain typed function symbols, which will get their meaning by higher-order rewrite rules.

2.5 Higher-Order Rewriting

We are now ready to introduce higher-order rewriting. To extend first-order term rewriting with a binding mechanism and functional variables we use a metalanguage and express all binding mechanisms via this metalanguage. Following Nipkow [64, 70] we use the simply-typed lambda calculus with constants as metalanguage. Following van Oostrom and van Raamsdonk [77, 87] we refer to this metalanguage as the substitution calculus. Another possibility would be to use the untyped lambda-calculus using complete developments, which yields the closely related formalism of Combinatory Reduction Systems [53].

Definition 2.43. A *signature* is a set of typed constants $\mathcal{F}$. The set of *pre-terms* is defined inductively by extending the rules for typed λ-terms from Definition 2.38 with

$$\frac{f : \sigma \in \mathcal{F}}{f : \sigma}$$

A *term* is a pre-term in η-long β-normal form, i.e., every pre-term s corresponds to a unique term $s{\updownarrow^\eta_\beta}$.[4]

[4] An alternative, equivalent definition is to define terms as equivalence classes modulo $\beta\eta$ and choosing the normalized pre-term as representative.

To ease readability terms are often written in algebraic notation. That is, a nested application $s\ t_1 \cdots t_n$ is written as $s(t_1, \ldots, t_n)$. We will make use of this convention throughout the remainder of this thesis.

Definition 2.44. Let s be a term. Then s is of the form $\lambda x_1 \ldots x_n.\, a(s_1, \ldots, s_n)$ with $a \in \mathcal{V} \cup \mathcal{F}$. The *top* of s is defined as $\mathsf{tp}(s) = a$.

Subterms and positions are defined as follows for higher-order terms.

Definition 2.45. Let s be a term. The *set of subterms* of s is defined as

$$\mathsf{Sub}(s) = \begin{cases} \{s\} \cup \mathsf{Sub}(t) & \text{if } s = \lambda x.\, t \\ \{s\} \cup \bigcup_{i=1}^{n} \mathsf{Sub}(s_i) & \text{if } s = a(s_1, \ldots, s_n) \end{cases}$$

We write $s \trianglerighteq t$ for $t \in \mathsf{Sub}(s)$ and $s \triangleright t$ for $s \trianglerighteq t$ and $s \neq t$. The *set of positions* of s is defined by

$$\mathsf{Pos}(s) = \begin{cases} \{\epsilon\} \cup \{1p \mid p \in \mathsf{Pos}(t)\} & \text{if } s = \lambda x.\, t \\ \{\epsilon\} \cup \{ip \mid 1 \leqslant i \leqslant n \text{ and } p \in \mathsf{Pos}(s_i)\} & \text{if } s = a(s_1, \ldots, s_n) \end{cases}$$

We again denote the subterm of s at position $p \in \mathsf{Pos}(s)$ by $s|_p$, i.e.,

$$s|_p = \begin{cases} s & \text{if } p = \epsilon \\ t|_q & \text{if } p = 1q \text{ and } s = \lambda x.\, t \\ s_i|_q & \text{if } p = iq \text{ and } s = a(s_1, \ldots, s_n) \end{cases}$$

Definition 2.46. A *rewrite rule* over $\mathcal{F}$ and $\mathcal{V}$ is a pair of terms $\ell \to r$ in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ such that ℓ and r have the same base type and all free variables of r occur as free variables in ℓ and $\mathsf{tp}(\ell)$ is not a variable. A *higher-order rewrite system* (HRS) consists of a signature $\mathcal{F}$ and a set $\mathcal{R}$ of rules.

Definition 2.47. For a rule $\ell \to r \in \mathcal{R}$, a substitution σ and a context C, the rewrite relation $\to_{\mathcal{R}}$ is defined by

$$C[\ell\sigma\!\downarrow_{\beta}^{\eta}] \to_{\mathcal{R}} C[r\sigma\!\downarrow_{\beta}^{\eta}]$$

When this definition was conceived, it was unknown whether higher-order matching and thus this rewrite relation are decidable. Thus attention is commonly restricted to so-called patterns.

Definition 2.48. A term s is called a *pattern* if for all subterms of s of the form $x(t_1, \ldots, t_n)$ with $x \in \mathsf{fv}(s)$ and $n > 0$, all t_i are the η-long forms of different bound variables. An HRS $\mathcal{R}$ is a *pattern rewrite system* (PRS) if for all $\ell \to r \in \mathcal{R}$, ℓ is a pattern.

Example 2.49. Consider a single base type o, a signature consisting of $\mathsf{c} : \mathsf{o}$ and $\mathsf{f} : \mathsf{o} \to \mathsf{o}$ and variables $x : \mathsf{o}$, $y : \mathsf{o}$, $z : \mathsf{o} \to \mathsf{o}$, $v : \mathsf{o} \to \mathsf{o} \to \mathsf{o}$ and $w : (\mathsf{o} \to \mathsf{o}) \to \mathsf{o}$. Then $\lambda x.\, \mathsf{f}(x)$, x, $\lambda z.\, w(\lambda x.\, z(x))$ and $\lambda x\, y.\, v(x, y)$ are patterns, while $z(\mathsf{c})$, $\lambda x.\, v(x, x)$, $\lambda x\, y.\, v(y, \mathsf{c})$ and $\lambda x.\, w(v(x))$ are not.

The main result about patterns is due to Miller [65]. It states that unification, and hence also matching, is decidable for patterns and that if they are unifiable, a most general unifier can be computed. Qian showed that patterns can be unified in linear time [86]. Although, Stirling recently showed [100] that higher-order matching is indeed decidable, this does not mean that PRSs are no longer of interest. First, higher-order matching, although decidable, has non-elementary complexity [97, 111]. Moreover, since even second-order unification is undecidable [32], as soon as unification is involved one really needs a restriction, for example to compute critical pairs.

There is another problem concerning confluence with non-pattern systems. For PRSs the restriction $\mathsf{fv}(\ell) \supseteq \mathsf{fv}(r)$ guarantees that rewriting does not introduce new variables. This fails for general HRSs. For instance for an HRS $\mathcal{R}$ consisting of the single rule $\mathsf{f}(F(x)) \to \mathsf{f}(x)$ we have $\mathsf{f}(y) \to_{\mathcal{R}} \mathsf{f}(x)$ using the substitution $\{F \mapsto \lambda z.\, y\}$.

The map function, applying a function to all elements of a list, can be modeled as a PRS.

Example 2.50 (Cop #430)**.** The signature $\mathcal{F}$ consists of

$$\begin{aligned} \mathsf{0} &: \mathsf{nat} \\ \mathsf{s} &: \mathsf{nat} \to \mathsf{nat} \\ \mathsf{nil} &: \mathsf{natlist} \\ \mathsf{cons} &: \mathsf{nat} \to \mathsf{natlist} \to \mathsf{natlist} \\ \mathsf{map} &: (\mathsf{nat} \to \mathsf{nat}) \to \mathsf{natlist} \to \mathsf{natlist} \end{aligned}$$

The two rewrite rules in $\mathcal{R}$ are:

$$\begin{aligned} \mathsf{map}(\lambda x.\, F(x), \mathsf{nil}) &\to \mathsf{nil} \\ \mathsf{map}(\lambda x.\, F(x), \mathsf{cons}(h, t)) &\to \mathsf{cons}(F(h), \mathsf{map}(\lambda x.\, F(x), t)) \end{aligned}$$

Here the types of the variables are $F : \mathsf{nat} \to \mathsf{nat}$, $h : \mathsf{nat}$, $t : \mathsf{natlist}$ and $x : \mathsf{nat}$. Also note that, since terms are in η-long β-normal form, F gets η-expanded to $\lambda x.\, F(x)$. As an example computation in this PRS consider mapping the identity function over the singleton list containing $\mathsf{s}(\mathsf{0})$:

$$\begin{aligned}\mathsf{map}(\lambda x.\, x, \mathsf{cons}(\mathsf{s}(\mathsf{0}), \mathsf{nil})) &\to \mathsf{cons}((\lambda x.\, x)\ \mathsf{s}(\mathsf{0}), \mathsf{map}(\lambda x.\, x, \mathsf{nil}))\updownarrow_\beta^\eta \\ &= \mathsf{cons}(\mathsf{s}(\mathsf{0}), \mathsf{map}(\lambda x.\, x, \mathsf{nil})) \\ &\to \mathsf{cons}(\mathsf{s}(\mathsf{0}), \mathsf{nil}\updownarrow_\beta^\eta) \\ &= \mathsf{cons}(\mathsf{s}(\mathsf{0}), \mathsf{nil})\end{aligned}$$

Note that in the first step the term does match the left-hand side of the second rule because $(\lambda y.\, (\lambda x.\, x) \cdot y) \to_\beta (\lambda y.\, y) \neq_\alpha (\lambda x.\, x)$ and so

$$\mathsf{map}(\lambda x.\, F(x), \mathsf{cons}(h, t))\sigma\updownarrow_\beta^\eta = \mathsf{map}(\lambda x.\, x, \mathsf{cons}(\mathsf{s}(\mathsf{0}), \mathsf{nil}))$$

with $\sigma = \{F \mapsto \lambda x.\, x, h \mapsto \mathsf{s}(\mathsf{0}), t \mapsto \mathsf{nil}\}$.

Chapter 3

Closing Critical Pairs

I have had my results for a long time,
but I do not yet know how I am to arrive at them.
Carl F. Gauss

In this chapter we present the formalization of three classic confluence results for first-order term rewrite systems. We have formalized proofs, showing that (a) linear strongly closed systems, (b) left-linear parallel closed systems, and (c) left-linear almost parallel closed systems are confluent. The former two results are due to Huet [45] and presented in full detail in the textbook of Baader and Nipkow [11, Lemma 6.3 and Section 6.4]. The third result is due to Toyama [107].

In the second part of the chapter we are concerned with more a recent idea due to Oyamaguchi and Hirokawa [82] that can be understood as a combination of Huet's results and Newman's Lemma: one can weaken the joining conditions for the critical pairs if additionally termination is required for the subsystem of rules that are used to perform the joining steps.

Several of the criteria in this chapter are extended to commutation. To this end we first revisit the critical pair lemma for commutation in Section 3.1. In Section 3.2 we show that linear strongly closed rewrite systems are confluent. Linearity is an important limitation, but the result does have its uses [31]. Section 3.3 is devoted to the formalization of the result of Huet that a left-linear rewrite system is confluent if its critical pairs are parallel closed. In Section 3.4 we consider Toyama's generalization of the previous result. Apart from a weaker joinability requirement on overlays, the result is again extended to commutation. Section 3.5 is devoted to critical pair closing systems. In Section 3.6 we explain what is needed for the automatic certification of confluence proofs that employ the formalized techniques. In the final section we conclude with an outlook on future work, in particular the challenges that need to be overcome when

extending the results from parallel closed rewrite systems to development closed higher-order rewrite systems [78].

The main IsaFoR theories relevant in this chapter are `Strongly_Closed.thy`, for the result on strongly closed rewrite systems, `Parallel_Closed.thy` for results on parallel closed systems (where we make heavy use of multihole contexts, cf. `Multihole_Context.thy`), `Critical_Pair_Closing_Systems.thy` for critical-pair-closing systems, and `Critical_Pair_Closure_Impl.thy` for the executable check functions.

3.1 Critical Pair Lemma

The famous critical pair lemma states that in order to check local confluence it suffices to check joinability of all critical pairs.

Lemma 3.1 (Knuth and Bendix [54], Huet [45])**.** *A TRS is locally confluent if and only if all its critical pairs are joinable.* □

Together with Newman's Lemma this yields decidability of confluence for finite terminating TRSs: for every critical pair $s \leftarrow\rtimes\rightarrow t$ compute terms s' and t' with $s \to^! s'$ and $t \to^! t'$, which is possible since the TRS is terminating. If $s' = t'$ for all critical pairs, then the TRS is locally confluent by the critical pair lemma and thus also confluent by Newman's Lemma. If one finds a critical pair with $s' \neq t'$ then the TRS is not confluent, since $s' \,{}^*\!\leftarrow \cdot \rightarrow^* t'$ but s' and t' are not joinable, because they are distinct normal forms. For commutation the situation is different, however—without further restrictions the critical pair lemma does not hold.

Example 3.2. Consider the two one rule TRSs $\mathcal{R}$ and $\mathcal{S}$:

$$\mathcal{R} : \mathsf{f}(x, x) \to x \qquad\qquad \mathcal{S} : \mathsf{a} \to \mathsf{b}$$

They do not admit any critical pairs, but do not satisfy local commutation, as witnessed by the non-commuting peak

$$\mathsf{a} \;{}_{\mathcal{R}}\!\leftarrow \mathsf{f}(\mathsf{a}, \mathsf{a}) \to_{\mathcal{S}} \mathsf{f}(\mathsf{b}, \mathsf{a})$$

The problem is easy to see. Applying the second rule to one of the arguments of $\mathsf{f}(\mathsf{a}, \mathsf{a})$ destroyed the possibility to apply the first rule. This effect also happens in the confluence setting, but there one can do additional steps to re-balance

the term, which, when dealing with commutation, is not possible. To recover the result we need to restrict attention to left-linear TRSs.[1]

Lemma 3.3. *Left-linear TRSs $\mathcal{R}$ and $\mathcal{S}$ locally commute if their critical pairs are joinable:*

$$(\mathcal{R}{\leftarrow}{\rtimes}{\rightarrow}\mathcal{S}) \cup (\mathcal{R}{\leftarrow}{\ltimes}{\rightarrow}\mathcal{S}) \subseteq \rightarrow_{\mathcal{S}}^{*} \cdot {}_{\mathcal{R}}^{*}{\leftarrow}$$

Proof. Consider a peak $t \,{}_{\mathcal{R}}{\leftarrow}\, s \rightarrow_{\mathcal{S}} u$. Then there are positions $p_1, p_2 \in \mathsf{Pos}(s)$, substitutions σ_1, σ_2 and rules $\ell_1 \to r_1 \in \mathcal{R}$ and $\ell_2 \to r_2 \in \mathcal{S}$ with $s|_{p_1} = \ell_1\sigma_1$, $s|_{p_2} = \ell_2\sigma_2$ and $t = s[r_1\sigma_1]_{p_1}$, $u = s[r_2\sigma_2]_{p_2}$. We show the existence of a term v with $t \rightarrow_{\mathcal{S}}^{*} v$ and $u \rightarrow_{\mathcal{R}}^{*} v$ by analyzing the positions p_1 and p_2. If they are parallel then $t \rightarrow_{\mathcal{S}} t[r_2\sigma_2]_{p_2} = u[r_1\sigma_1]_{p_1} \,{}_{\mathcal{R}}{\leftarrow}\, u$. If they are not parallel then one is above the other.

Without loss of generality assume $p_1 \leqslant p_2$, then there is a position q with $p_2 = p_1q$. If $q \in \mathsf{Pos}_{\mathcal{F}}(\ell_1)$ then $\ell_1|_q\sigma_1 = \ell_2\sigma_2$ and thus there is a critical pair $r_1\mu \,{}_{\mathcal{R}}{\leftarrow}{\ltimes}{\rightarrow}_{\mathcal{S}}\, \ell_1\mu[r_2\mu]_q$, which is joinable by assumption, and by closure under contexts and substitutions also t and u are joinable.

If $q \notin \mathsf{Pos}_{\mathcal{F}}(\ell_1)$ the step in $\mathcal{S}$ happens in the substitution. That is, there is a variable x and positions q_1, q_2 with $q = q_1q_2$, $q_1 \in \mathsf{Pos}(\ell_1)$, $\ell_1|_{q_1} = x$. Suppose $|r_1|_x = n$, then $t \rightarrow_{\mathcal{S}}^{n} v \,{}_{\mathcal{R}}{\leftarrow}\, u$ for $v = s[r_1\tau]_{p_1}$ with $\tau(y) = \sigma_1(y)$ for all $y \in \mathcal{V} \setminus \{x\}$ and $\tau(x) = \sigma_1(x)[r_2\sigma_2]_{q_2}$. Note that $u = s[\ell_1\tau]_{p_1}$ due to linearity of ℓ_1, see also the proof of Lemma 3.6 below. □

Analyzing a local peak in this way—the two redexes are either parallel, give rise to a critical pair, or one is in the substitution of the other—is a common theme. We will use it again in e.g. the proof of strong closedness and in Section 4.2.1 where more details about the joining sequences are needed in order to be used in the context of decreasing diagrams.

3.2 Strongly Closed Critical Pairs

The next criterion we consider is due to Huet [45] and based on the observation that in a linear rewrite system it suffices to have strong-confluence like joins for all critical pairs in order to guarantee strong confluence of the rewrite system.

[1] The problems with non-left-linearity run even deeper: commutation is not preserved under signature extension for non-left-linear TRSs [39]—but for left-linear ones it is.

Definition 3.4. A TRS $\mathcal{R}$ is *strongly closed* if every critical pair $s \leftarrow\!\rtimes\!\rightarrow t$ of $\mathcal{R}$ satisfies both $s \to^{=} \cdot {}^{*}\!\leftarrow t$ and $s \to^{*} \cdot {}^{=}\!\leftarrow t$.

The following folklore lemma tells us that in a linear term applying a substitution can be done by replacing the one subterm where the variable occurs and applying the remainder of the substitution.

Lemma 3.5. *Let t be a linear term and let $p \in \mathsf{Pos}_{\mathcal{V}}(t)$ be a position with $t|_p = x$. Then for substitutions σ and τ with $\sigma(y) = \tau(y)$ for all $y \in \mathcal{V}\mathsf{ar}(t)$ different from x we have $t\tau = t\sigma[\tau(x)]_p$.* □

The proof that linear strongly closed systems are strongly confluent is very similar to the one of the critical pair lemma, by analyzing the relative positions of the rewrite steps in a peak. The next lemma, which appears implicitly in Huet's proof of Corollary 3.7, takes care of the case where one position is above the other.

Lemma 3.6. *Let $\mathcal{R}$ be a linear, strongly closed TRS and assume $s \to_{\mathcal{R}} t$ with rule $\ell_1 \to r_1$ and substitution σ_1 at position p_1 and let $s \to_{\mathcal{R}} u$ with rule $\ell_2 \to r_2$ and substitution σ_2 at position p_2 with $p_1 \leqslant p_2$. Then there are terms v and w with $t \to^{*}_{\mathcal{R}} v \,\overline{\overline{\mathcal{R}}}\!\leftarrow u$ and $t \to^{\overline{\overline{=}}}_{\mathcal{R}} w \,{}^{*}_{\mathcal{R}}\!\leftarrow u$.*

Proof. By assumption we have $s|_{p_1} = \ell_1\sigma_1$, $s|_{p_2} = \ell_2\sigma_2$, $t = s[r_1\sigma_1]_{p_1}$, and $u = s[r_2\sigma_2]_{p_2}$. Since $p_1 \leqslant p_2$ there is a position q with $p_2 = p_1 q$ and $(\ell_1\sigma_1)|_q = \ell_2\sigma_2$. So we can write u as $u = s[(\ell_1\sigma_1)[r_2\sigma_2]_q]_{p_1}$.

We now distinguish whether the step from s to u overlaps with the one from s to t or takes place in the substitution. i.e., we perform a case analysis on $q \in \mathsf{Pos}_{\mathcal{F}}(\ell_1)$. If that is the case then $\ell_1|_q\sigma_1 = \ell_2\sigma_2$ and thus there is a critical pair $\ell_1\mu[r_2\mu]_q \leftarrow\!\rtimes\!\rightarrow r_1\mu$. But then by assumption we obtain terms v and w with $r_1\mu \to^{*}_{\mathcal{R}} v \,\overline{\overline{\mathcal{R}}}\!\leftarrow \ell_1\mu[r_2\mu]_q$ and $r_1\mu \to^{\overline{\overline{=}}}_{\mathcal{R}} w \,{}^{*}_{\mathcal{R}}\!\leftarrow \ell_1\mu[r_2\mu]_q$. Closure under contexts and substitutions yields the desired result.

If $q \notin \mathsf{Pos}_{\mathcal{F}}(\ell_1)$ we obtain positions q_1, q_2 and a variable x with $q = q_1q_2$, $q_1 \in \mathsf{Pos}(\ell_1)$, $\ell_1|_{q_1} = x$, and $(x\sigma_1)|_{q_2} = \ell_2\sigma_2$. Define the substitution τ as

$$\tau(y) = \begin{cases} (x\sigma_1)[r_2\sigma_2]_{q_2} & \text{if } y = x \\ y\sigma_1 & \text{otherwise} \end{cases}$$

Since ℓ_1 is linear by assumption we have $\ell_1\tau = (\ell_1\sigma_1)[(x\sigma_1)[r_2\sigma_2]_{q_2}]_{q_1}$ using Lemma 3.5 and hence also $\ell_1\tau = (\ell_1\sigma_1)[r_2\sigma_2]_q$. But then $u = s[\ell_1\tau]_{p_1}$ and

thus $u \to_{\mathcal{R}} s[r_1\tau]_{p_1}$. We conclude by showing $t \to_{\mathcal{R}}^{=} s[r_1\tau]_{p_1}$. If $x \notin \mathcal{V}\mathsf{ar}(r_1)$ then $r_1\tau = r_1\sigma_1$ and thus $t = s[r_1\tau]_{p_1}$. Otherwise we obtain a position $q' \in \mathsf{Pos}(r_1)$ with $r_1|_{q'} = x$. Then, again using linearity and Lemma 3.5 we have $r_1\tau = (r_1\sigma_1)[(x\sigma_1)[r_2\sigma_2]_{q_2}]_{q'}$ and hence $r_1\tau = (r_1\sigma_1)[r_2\sigma_2]_{q'q2}$. Since also $r_1\sigma_1 = (r_1\sigma_1)[l_2\sigma_2]_{q'q2}$ we have $r_1\sigma_1 \to_{\mathcal{R}} r_1\tau$ and by closure under context also $t \to_{\mathcal{R}} s[r_1\tau]_{p_1}$. □

Now the main result of this section follows easily.

Corollary 3.7 (Huet [45])**.** *If a TRS $\mathcal{R}$ is linear and strongly closed then $\to_{\mathcal{R}}$ is strongly confluent.*

Proof. Assume $s \to_{\mathcal{R}} t$ and $s \to_{\mathcal{R}} u$. Then there are positions $p_1, p_2 \in \mathsf{Pos}(s)$, substitutions σ_1, σ_2 and rules $\ell_1 \to r_1$, $\ell_2 \to r_2$ in $\mathcal{R}$ with $s|_{p_1} = \ell_1\sigma_1$, $s|_{p_2} = \ell_2\sigma_2$ and $t = s[r_1\sigma_1]_{p_1}$, $u = s[r_2\sigma_2]_{p_2}$. We show existence of a term v with $t \to^* v$ and $u \to^= v$ by analyzing the positions p_1 and p_2. If they are parallel then $t \to t[r_2\sigma_2]_{p_2} = u[r_1\sigma_1]_{p_1} \leftarrow u$. If they are not parallel then one is above the other and we conclude by Lemma 3.6. □

Then by Lemma 2.11 $\mathcal{R}$ is also confluent.

Example 3.8 (Cop #103)**.** Consider the TRS $\mathcal{R}$ consisting of the two rewrite rules

$$\mathsf{f}(\mathsf{f}(x,y),z) \to \mathsf{f}(x,\mathsf{f}(y,z)) \qquad\qquad \mathsf{f}(x,y) \to \mathsf{f}(y,x)$$

Note that rules of this shape are of special interest—they state that f is associative and commutative (AC). There are four non-trivial critical pairs:

$$\mathsf{f}(\mathsf{f}(x,\mathsf{f}(y,z)),v) \leftarrow\!\rtimes\!\rightarrow \mathsf{f}(\mathsf{f}(x,y),\mathsf{f}(z,v)) \qquad \mathsf{f}(x,\mathsf{f}(y,z)) \leftarrow\!\rtimes\!\rightarrow \mathsf{f}(z,\mathsf{f}(x,y))$$
$$\mathsf{f}(z,\mathsf{f}(x,y)) \leftarrow\!\rtimes\!\rightarrow \mathsf{f}(x,\mathsf{f}(y,z)) \qquad \mathsf{f}(\mathsf{f}(y,x),z) \leftarrow\!\rtimes\!\rightarrow \mathsf{f}(x,\mathsf{f}(y,z))$$

Because of the rewrite sequences

$$\begin{aligned}
&\mathsf{f}(\mathsf{f}(x,\mathsf{f}(y,z)),v) \to \mathsf{f}(x,\mathsf{f}(\mathsf{f}(y,z)),v) \to \mathsf{f}(x,\mathsf{f}(y,\mathsf{f}(z,v)))\\
&\mathsf{f}(\mathsf{f}(x,y),\mathsf{f}(z,v)) \to \mathsf{f}(x,\mathsf{f}(y,\mathsf{f}(z,v)))\\
&\mathsf{f}(\mathsf{f}(x,y),\mathsf{f}(z,v)) \to \mathsf{f}(\mathsf{f}(z,v),\mathsf{f}(x,y)) \to \mathsf{f}(\mathsf{f}(v,z),\mathsf{f}(x,y)) \to \mathsf{f}(v,\mathsf{f}(z,\mathsf{f}(x,y)))\\
&\qquad\qquad \to \mathsf{f}(\mathsf{f}(z,\mathsf{f}(x,y)),v) \to \mathsf{f}(\mathsf{f}(\mathsf{f}(x,y),z),v) \to \mathsf{f}(\mathsf{f}(x,\mathsf{f}(y,z)),v)\\
&\mathsf{f}(z,\mathsf{f}(x,y)) \to \mathsf{f}(\mathsf{f}(x,y),z) \to \mathsf{f}(x,\mathsf{f}(y,z))
\end{aligned}$$

$$\begin{aligned}
&\mathsf{f}(x,\mathsf{f}(y,z)) \to \mathsf{f}(\mathsf{f}(y,z),x) \to \mathsf{f}(y,\mathsf{f}(x,z))\\
&\qquad\qquad \to \mathsf{f}(\mathsf{f}(x,z),y) \to \mathsf{f}(\mathsf{f}(z,x),y) \to \mathsf{f}(z,\mathsf{f}(x,y))\\
&\mathsf{f}(\mathsf{f}(y,x),z) \to \mathsf{f}(\mathsf{f}(x,y),z) \to \mathsf{f}(x,\mathsf{f}(y,z))\\
&\mathsf{f}(x,\mathsf{f}(y,z)) \to \mathsf{f}(\mathsf{f}(y,z),x) \to \mathsf{f}(\mathsf{f}(z,y),x) \to \mathsf{f}(z,\mathsf{f}(y,x)) \to \mathsf{f}(\mathsf{f}(y,x),z)
\end{aligned}$$

$\mathcal{R}$ is strongly closed. Since it is also linear, we obtain confluence.

The next example shows how to apply the criterion to a TRS that does not fulfill the variable conditions.

Example 3.9 (Cop #394)**.** Consider the linear TRS $\mathcal{R}$ consisting of the following three rules:

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

There are four critical pairs modulo symmetry:

$$\mathsf{f}(y) \leftarrow\!\!\rtimes\!\!\rightarrow \mathsf{f}(x) \quad \mathsf{f}(\mathsf{g}(\mathsf{a})) \leftarrow\!\!\rtimes\!\!\rightarrow \mathsf{f}(x) \quad \mathsf{b} \leftarrow\!\!\rtimes\!\!\rightarrow \mathsf{b} \quad \mathsf{f}(\mathsf{g}(\mathsf{f}(x))) \leftarrow\!\!\rtimes\!\!\rightarrow \mathsf{b}$$

Using the second rule it is easy to see that all of them are strongly closed. Hence $\mathcal{R}$ is confluent.

In the next section we consider a criterion that drops the condition on $\mathcal{R}$ to be right-linear.

3.3 Parallel Closed Critical Pairs

The criterion from the previous section requires the TRS to be linear and while left-linearity is a common restriction, right-linearity is a rather unnatural one. Thus we turn our attention to criteria for left-linear systems that change the restriction on the joinability of critical pairs. The crucial observation is that in a non-right-linear system executing the upper step in a variable overlap can duplicate the redex below. Thus to join such a situation multiple steps might be necessary, all of which take place at parallel positions. Consequently we consider parallel rewriting. The following definition describes the new joinability condition.

Definition 3.10. A TRS $\mathcal{R}$ is *parallel closed* if every critical pair $s \leftarrow\!\!\rtimes\!\!\rightarrow t$ of $\mathcal{R}$ satisfies $s \twoheadrightarrow\!\!\!\!\!\!+_{\mathcal{R}}\ t$.

Together with left-linearity this guarantees the diamond property of the parallel rewrite relation.

Theorem 3.11 (Huet [45])**.** *If a TRS $\mathcal{R}$ is left-linear and parallel closed then $\twoheadrightarrow_{\mathcal{R}}$ has the diamond property.*

The proof of this theorem is much more involved than the one for strongly closed systems. The first observation is that we will now have to consider a peak of parallel steps, in order to show the diamond property of $\twoheadrightarrow$. In case the two parallel steps are orthogonal to each other, they simply commute by the well-known Parallel Moves Lemma. However, if they do interfere the assumption of the theorem only allows us to close a single critical pair to reduce the amount of interference. Thus we will have to use some form of induction on how much the patterns of the two parallel steps overlap. Figure 4 shows the setting for the overlapping case. The horizontal parallel step, described by the horizontally striped redexes, and the vertical step, described by the vertically striped redexes, overlap. Hence there is a critical pair, say the one obtained from overlapping the leftmost vertical redex with the leftmost horizontal redex. Then, by assumption there is a closing parallel step, which, since it takes place inside the critical pair, can be combined with the remaining horizontal redexes to obtain a new peak with less overlap, which can be closed by the induction hypothesis. When making this formal we identified two crucial choices. First the representation of the parallel rewrite relation and second the way to measure the amount of overlap between two parallel steps with the same source. Huet in his original proof heavily uses positions. That is, a parallel step is defined as multiple single steps that happen at parallel positions and for measuring overlap he takes the sum of the sizes of the subterms that are affected by both steps. More precisely, writing $\twoheadrightarrow_P$ for a parallel step that takes place at positions in a set P, for a peak $t \;{}_{P_1}\!\twoheadleftarrow s \twoheadrightarrow_{P_2} u$ he uses

$$\sum_{q \in Q} |s|_q|$$

where $Q = \{p_1 \in P_1 \mid p_2 \leqslant p_1 \text{ for some } p_2 \in P_2\} \cup \{p_2 \in P_2 \mid p_1 \leqslant p_2 \text{ for some } p_1 \in P_1\}$. This formulation is also adopted in the textbook by Baader and Nipkow [11]. Consequently, when starting the present formalization, we also adopted this definition. However, the book keeping required by working with sets of positions as well as formally reasoning about this measure in Isabelle

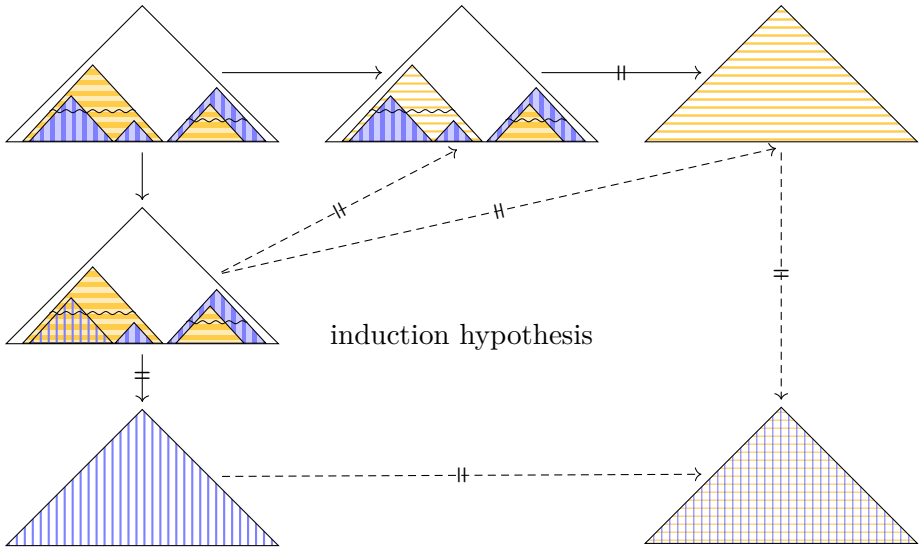


Figure 4: Overview of the proof of Theorem 3.11.

became so convoluted that it very much obscured the ingenuity and elegance of Huet's original idea while at the same time defeating our formalization efforts. Hence in the end we had to adopt a different approach.

Toyama [107], in the proof of his extension of Huet's result, does not use positions at all and instead relies on (multihole) contexts, which means a parallel step is then described by a context and a list of root steps that happen in the holes. To measure overlap he collects those redexes that are subterms of some redex in the other step, i.e., decorating the parallel rewrite relation with the redexes contracted in the step, for a peak $t \; {}_{t_1,\ldots,t_n}\!\!\twoheadleftarrow \; s \; \twoheadrightarrow_{u_1,\ldots,u_m} \; u$ Toyama's measure is

$$\sum_{r \in S} |r|$$

where $S = \{u_i \mid u_i \trianglelefteq t_j \text{ for some } t_j\} \cup \{t_j \mid t_j \trianglelefteq u_i \text{ for some } u_i\}$. However, this measure turns out to be problematic as shown in the following example.

Example 3.12 (Cop #503)**.** Consider the TRS consisting of the following five rewrite rules:

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

Then we have the peak $\mathsf{f}(\mathsf{b},\mathsf{b},\mathsf{a},\mathsf{a}) \xleftarrow{\mathsf{a},\mathsf{a},\mathsf{b},\mathsf{b}} \mathsf{f}(\mathsf{a},\mathsf{a},\mathsf{b},\mathsf{b}) \xrightarrow{\mathsf{f}(\mathsf{a},\mathsf{a},\mathsf{b},\mathsf{b})} \mathsf{f}(\mathsf{c},\mathsf{c},\mathsf{c},\mathsf{c})$. The measure of this peak according to the definition above is 2, since $S = \{\mathsf{a},\mathsf{b}\} \cup \varnothing$. Now after splitting off one of the four critical steps—it does not matter which one—and closing the corresponding critical pair, we arrive at

$$\begin{array}{ccc} \mathsf{f}(\mathsf{a},\mathsf{a},\mathsf{b},\mathsf{b}) & \longrightarrow & \mathsf{f}(\mathsf{c},\mathsf{c},\mathsf{c},\mathsf{c}) \\ \downarrow & \nearrow\!\!\!\nearrow & \\ \mathsf{f}(\mathsf{b},\mathsf{a},\mathsf{b},\mathsf{b}) & & \\ \Downarrow & & \\ \mathsf{f}(\mathsf{b},\mathsf{b},\mathsf{a},\mathsf{a}) & & \end{array}$$

The measure of the new peak $\mathsf{f}(\mathsf{b},\mathsf{b},\mathsf{a},\mathsf{a}) \xleftarrow{\mathsf{a},\mathsf{b},\mathsf{b}} \mathsf{f}(\mathsf{b},\mathsf{a},\mathsf{b},\mathsf{b}) \xrightarrow{\mathsf{b},\mathsf{a},\mathsf{b},\mathsf{b}} \mathsf{f}(\mathsf{c},\mathsf{c},\mathsf{c},\mathsf{c})$ is still 2 since $S = \{\mathsf{a},\mathsf{b}\} \cup \{\mathsf{a},\mathsf{b}\}$.

Note that using multisets instead of sets does not help, since then the measure of the initial peak is 4 ($S = \{\mathsf{a},\mathsf{a},\mathsf{b},\mathsf{b}\}$) and of the new peak, after closing the critical pair, it is 7 since $S = \{\mathsf{a},\mathsf{b},\mathsf{b}\} \uplus \{\mathsf{b},\mathsf{a},\mathsf{b},\mathsf{b}\}$ (and even if we take into account that three of the redexes are counted twice we still get 4). The problem is that in the new peak the redex at position 1 of the closing step is counted again, because b is a subterm of one the redexes of the other step. Hence it is crucial to only count redexes at overlapping positions.

To remedy this situation we will collect all *overlapping* redexes of a peak in a multiset. These multisets will then be compared by $\rhd_{\mathsf{mul}}$, the multiset extension of the proper superterm relation. We start by characterizing parallel rewrite steps using multihole contexts.

Definition 3.13. We write

$$s \xrightarrow{C,a_1,\ldots,a_n}_{\mathcal{R}} t$$

if $s = C[a_1,\ldots,a_n]$ and $t = C[b_1,\ldots,b_n]$ for some $b_1,\ldots,b_n$ with $a_i \to^{\epsilon}_{\mathcal{R}} b_i$ for all $1 \leqslant i \leqslant n$.

To save space we sometimes abbreviate the list of terms $a_1,\ldots,a_n$ by $\overline{a}$ and denote the step by

$$s \xrightarrow{C,\overline{a}}_{\mathcal{R}} t$$

leaving the length of the list of redexes implicit. The following expected correspondence is easily shown by induction.

Lemma 3.14. *We have $s \twoheadrightarrow_{\mathcal{R}} t$ if and only if $s \xrightarrow{C,\overline{s}}_{\mathcal{R}} t$ for some C and $\overline{s}$.* □

Now we can formally measure the overlap between two parallel rewrite steps by collecting those redexes that are below some redex in the other step.

Definition 3.15. The *overlap* between two co-initial parallel rewrite steps is defined by the following equations

$$\blacktriangle\left(\xleftarrow{\square,a} s \xrightarrow{\square,b}\right) = \{s\}$$

$$\blacktriangle\left(\xleftarrow{C,a_1,\ldots,a_c} s \xrightarrow{\square,b}\right) = \{a_1,\ldots,a_c\}$$

$$\blacktriangle\left(\xleftarrow{\square,a} s \xrightarrow{D,b_1,\ldots,b_d}\right) = \{b_1,\ldots,b_d\}$$

$$\blacktriangle\left(\xleftarrow{f(C_1,\ldots,C_n),\overline{a}} f(s_1,\ldots,s_n) \xrightarrow{f(D_1,\ldots,D_n),\overline{b}}\right) = \bigcup_{i=1}^{n} \blacktriangle\left(\xleftarrow{C_i,\overline{a}_i} s_i \xrightarrow{D_i,\overline{b}_i}\right)$$

where $\overline{a}_1,\ldots,\overline{a}_n = \overline{a}$ and $\overline{b}_1,\ldots,\overline{b}_n = \overline{b}$ are partitions of $\overline{a}$ and $\overline{b}$ such that the length of $\overline{a}_i$ and $\overline{b}_i$ matches the number of holes in C_i and D_i, for all $1 \leqslant i \leqslant n$.

Given functionality to do the partitioning, this definition is straightforward to write down in Isabelle, the formal text is shown in Listing 5. Here ***mset*** converts a list to a multiset and # is used in Isabelle/HOL for denoting (operations on) multisets, e.g. $\{\#\}$ denotes the empty multiset. We also use slightly different pattern matching compared to Definition 3.15: the case where both contexts are the hole is covered by the first equation of Listing 5, while the cases for variables where added to make the function total.

Example 3.16. Applying this definition for the two peaks from Example 3.12 yields

$$\blacktriangle\left(\xleftarrow{\mathsf{f}(\square,\square,\square,\square),\mathsf{a},\mathsf{a},\mathsf{b},\mathsf{b}} \mathsf{f}(\mathsf{a},\mathsf{a},\mathsf{b},\mathsf{b}) \xrightarrow{\square,\mathsf{f}(\mathsf{a},\mathsf{a},\mathsf{b},\mathsf{b})}\right) = \{\mathsf{a},\mathsf{a},\mathsf{b},\mathsf{b}\}$$

$$\blacktriangle\left(\xleftarrow{\mathsf{f}(\mathsf{b},\square,\square,\square),\mathsf{a},\mathsf{b},\mathsf{b}} \mathsf{f}(\mathsf{b},\mathsf{a},\mathsf{b},\mathsf{b}) \xrightarrow{\mathsf{f}(\square,\square,\square,\square),\mathsf{b},\mathsf{a},\mathsf{b},\mathsf{b}}\right) = \{\mathsf{a},\mathsf{b},\mathsf{b}\}$$

and $\{\mathsf{a},\mathsf{a},\mathsf{b},\mathsf{b}\} \rhd_{\mathsf{mul}} \{\mathsf{a},\mathsf{b},\mathsf{b}\}$ as desired.

Note that our definition of ▲ is in fact an over-approximation of the actual overlap between the steps. That is because we do not split redexes into the left-hand side of the applied rule and a substitution but take the redex as a whole. The following example illustrates the effect.

```
▲(MHole, [l]) (C, ls) = mset ls
▲(C, ls) (MHole, [l]) = mset ls
▲(MFun f Cs, ls) (MFun g Ds, ls') =
  ⋃#{# ▲(Cs ! i, partition_holes ls Cs ! i)
       (Ds ! i, partition_holes ls' Ds ! i)
     . i ∈# mset [0..<length Cs]#}
▲(MVar x, []) (C, ls) = {#}
▲(C, ls) (MVar x, []) = {#}
```

Listing 5: Definition of overlap between two parallel steps.

Example 3.17. Consider the rewrite system consisting of the two rules

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

and the peak $\mathsf{a} \leftarrow \mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{b})$. We have

$$\blacktriangle\left(\overset{\square,\mathsf{f}(\mathsf{a})}{\longleftarrow\!\!\shortparallel} \mathsf{f}(\mathsf{a}) \overset{\mathsf{f}(\square),\mathsf{a}}{\shortparallel\!\!\longrightarrow}\right) = \{\mathsf{a}\}$$

although the two steps do not overlap—the step to the right takes place completely in the substitution of the one to the left (in fact the rewrite system in question is orthogonal).

However, since we are dealing with parallel rewriting, no problems arise from this over-approximation. This changes when extending the results to development steps, see Section 3.7 for further discussion.

The following properties of ▲ turned out to be crucial in our proof of Theorem 3.11.

Lemma 3.18. *For a peak* $\overset{C,\overline{a}}{\longleftarrow\!\!\shortparallel}\ s\ \overset{D,\overline{b}}{\shortparallel\!\!\longrightarrow}$ *the following properties of* ▲ *hold.*

- *If* $s = f(s_1, \ldots, s_n)$ *with* $C = f(C_1, \ldots, C_n)$ *and* $D = f(D_1, \ldots, D_n)$ *then*

$$\blacktriangle\left(\overset{C_i,\overline{a}_i}{\longleftarrow\!\!\shortparallel}\ s_i\ \overset{D_i,\overline{b}_i}{\shortparallel\!\!\longrightarrow}\right) \subseteq \blacktriangle\left(\overset{C,\overline{a}}{\longleftarrow\!\!\shortparallel}\ s\ \overset{D,\overline{b}}{\shortparallel\!\!\longrightarrow}\right)$$

  for all $1 \leqslant i \leqslant n$.

- *The overlap is bounded by* $\overline{a}$*, i.e., we have*

$$\{\overline{a}\} \rhd^{=}_{\mathsf{mul}} \blacktriangle\left(\overset{C,\overline{a}}{\longleftarrow\!\!\shortparallel}\ s\ \overset{D,\overline{b}}{\shortparallel\!\!\longrightarrow}\right)$$

- *The overlap is symmetric, i.e.,* $\blacktriangle\left(\overset{C,\overline{a}}{\leftarrow\!\!\shortparallel}\ s\ \overset{D,\overline{b}}{\shortparallel\!\!\rightarrow}\right) = \blacktriangle\left(\overset{D,\overline{b}}{\leftarrow\!\!\shortparallel}\ s\ \overset{C,\overline{a}}{\shortparallel\!\!\rightarrow}\right)$. □

There is one more high-level difference between the formalization and the paper proof. In the original proof one needs to combine the closing step for the critical pair with the remainder of the original step in order to obtain a new peak, to which the induction hypothesis can then be applied. This reasoning can be avoided, by using an additional induction on the source of the peak. Then the case where neither of the two parallel steps is a root step (and thus a single step) can be discharged by the induction hypothesis of that induction.

The following technical lemma tells us that a parallel rewrite step starting from $s\sigma$ is either inside s, i.e., we can split off a critical pair, or we can do the step completely inside σ.

Lemma 3.19. *Let s be a linear term. If $s\sigma \overset{C,s_1,\ldots,s_n}{\xrightarrow{\quad\shortparallel\quad}}_{\mathcal{R}} t$ then either $t = s\tau$ for some substitution τ such that $x\sigma \twoheadrightarrow x\tau$ for all $x \in \mathcal{V}\mathsf{ar}(s)$ or there exist a context D, a non-variable term s', a rule $\ell \to r \in \mathcal{R}$, a substitution τ, and a multihole context C' such that $s = D[s']$, $s'\sigma = \ell\tau$, $D\sigma[r\tau] = C'[s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n]$ and $t = C'[t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n]$ for some $1 \leqslant i \leqslant n$.*

Proof. By induction on s. If $s = x$ for a variable x set $\tau = \{x \mapsto t\}$. Otherwise $s = f(s'_1, \ldots, s'_n)$, proceed by case analysis on C. If $C = \Box$ then $s = \ell\tau$ and $t = r\tau$ for some $\ell \to r \in \mathcal{R}$. Set $s' = s$ and $D = \Box$. Otherwise we obtain the desired result for all $s'_1, \ldots, s'_n$ by the induction hypothesis. If one of the steps in the arguments contains a critical pair so does the whole step. Otherwise, since s is linear, the parallel steps in the arguments can be combined to a step on the whole term. □

We are now ready to prove the main result of this section. To ease presentation, the following proof does use the condition that the left-hand sides of rewrite rules are not variables. By employing additional technical case analyses this restriction can be easily dropped. We refer to the formalization for details.

Proof of Theorem 3.11. Assume $t \overset{C,\overline{a}}{\leftarrow\!\!\shortparallel}\ s\ \overset{D,\overline{b}}{\shortparallel\!\!\rightarrow} u$. We show $t \twoheadrightarrow v \twoheadleftarrow u$ for some term v by well-founded induction on the overlap between the two parallel steps using the order $\rhd_{\mathsf{mul}}$ and continue by induction on s with respect to $\rhd$. If $s = x$ for some variable x then $t = u = x$. So let $s = f(s_1, \ldots, s_n)$. We distinguish four cases.

(a) If $C = f(C_1, \ldots, C_n)$ and $D = f(D_1, \ldots, D_n)$ then $t = f(t_1, \ldots, t_n)$ and $u = f(u_1, \ldots, u_n)$ and we obtain partitions $\overline{a}_1, \ldots, \overline{a}_n = \overline{a}$ and $\overline{b}_1, \ldots, \overline{b}_n = \overline{b}$ of $\overline{a}$ and $\overline{b}$ with

$$t_i \xleftarrow{C_i,\overline{a}_i} s_i \xrightarrow{D_i,\overline{b}_i} u_i$$

for all $1 \leqslant i \leqslant n$. Then, since we have

$$\blacktriangle\left(\xleftarrow{C_i,\overline{a}_i} s_i \xrightarrow{D_i,\overline{b}_i}\right) \subseteq \blacktriangle\left(\xleftarrow{C,\overline{a}} s \xrightarrow{D,\overline{b}}\right)$$

by Lemma 3.18 and thus also

$$\blacktriangle\left(\xleftarrow{C,\overline{a}} s \xrightarrow{D,\overline{b}}\right) \rhd^{=}_{\mathsf{mul}} \blacktriangle\left(\xleftarrow{C_i,\overline{a}_i} s_i \xrightarrow{D_i,\overline{b}_i}\right)$$

we can apply the inner induction hypothesis and obtain terms v_i with $t_i \twoheadrightarrow v_i \twoheadleftarrow u_i$ for all $1 \leqslant i \leqslant n$ and thus we have $t \twoheadrightarrow f(v_1, \ldots, v_n) \twoheadleftarrow u$.

(b) If $C = D = \Box$ then both steps are root steps and thus single rewrite steps and we can write $t = r_1\sigma_1 \xleftarrow{\epsilon} \ell_1\sigma_1 = s = \ell_2\sigma_2 \xrightarrow{\epsilon} r_2\sigma_2 = u$. Hence, since $\ell_1\sigma_1 = \ell_2\sigma_2$, there is a critical pair $r'_1\mu \leftarrow\!\bowtie\!\rightarrow r'_2\mu$ for variable disjoint variants $\ell'_1 \to r'_1$, $\ell'_2 \to r'_2$ of $\ell_1 \to r_1$, $\ell_2 \to r_2$ with μ a most general unifier of ℓ'_1 and ℓ'_2. Then by assumption $r'_1\mu \twoheadrightarrow r'_2\mu$ and by closure under substitutions also $t = r_1\sigma_1 \twoheadrightarrow r_2\sigma_2 = u$.

(c) If $C = f(C_1, \ldots, C_n)$ and $D = \Box$ then the step to the right is a single root step and we write

$$t = f(t_1, \ldots, t_n) \xleftarrow{C,\overline{a}} s = \ell\sigma \xrightarrow{\epsilon} r\sigma = u$$

Since ℓ is linear by assumption, we can apply Lemma 3.19 and either obtain τ with $t = \ell\tau$ and $x\sigma \twoheadrightarrow x\tau$ for all $x \in \mathcal{V}\mathsf{ar}(\ell)$ or a critical pair.

- In the first case define

$$\delta(x) = \begin{cases} \tau(x) & \text{if } x \in \mathcal{V}\mathsf{ar}(\ell) \\ \sigma(x) & \text{otherwise} \end{cases}$$

We have $t = \ell\tau = \ell\delta$ by definition of δ and hence $t \twoheadrightarrow r\delta$ by a single root step. Moreover we have $u = r\sigma \twoheadrightarrow r\delta$ since $x\sigma \twoheadrightarrow x\delta$ for all variables $x \in \mathcal{V}\mathsf{ar}(r)$. This holds because either $x \in \mathcal{V}\mathsf{ar}(\ell)$ and then $x\sigma \twoheadrightarrow x\tau = x\delta$ or $x \notin \mathcal{V}\mathsf{ar}(\ell)$ and then $x\sigma = x\delta$.

- In the second case Lemma 3.19 yields a context E, a non-variable term ℓ'', a rule $\ell' \to r' \in \mathcal{R}$, a substitution τ, and a multihole context C' such that $E\sigma[r'\tau] = C'[a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_c]$, $t = C'[a'_1, \ldots, a'_{i-1}, a'_{i+1}, \ldots, a'_c]$, $\ell = E[\ell'']$, and $\ell''\sigma = \ell'\tau$ for some $1 \leqslant i \leqslant c$. Since $\ell''\sigma = \ell'\tau$ there is a critical pair $E\mu[r'\mu] \leftarrow\!\rtimes\!\rightarrow r\mu$ and by assumption $E\mu[r'\mu] \twoheadrightarrow r\mu$ and thus also $E\sigma[r'\tau] \twoheadrightarrow r\sigma$. That is, we obtain a new peak

$$t \xleftarrow{C',\overline{a}'} E\sigma[r'\tau] \twoheadrightarrow r\sigma$$

with $\overline{a}' = a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_c$. Since

$$\blacktriangle\left(\xleftarrow{C,\overline{a}} s \xrightarrow{\Box,\ell\sigma}\right) = \{a_1, \ldots, a_c\} \rhd_{\mathsf{mul}} \{a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_c\} \rhd^{=}_{\mathsf{mul}} \blacktriangle\left(\xleftarrow{C',\overline{a'}} E\sigma[r'\tau] \twoheadrightarrow\right)$$

by Lemma 3.18, we can apply the induction hypothesis and obtain v with $t \twoheadrightarrow v \twoheadleftarrow r\sigma = u$.

(d) The final case, where $D = f(D_1, \ldots, D_n)$ and $C = \Box$, is completely symmetric. □

Finally, by Lemma 2.11 and Lemma 2.33 we obtain confluence of $\to_{\mathcal{R}}$.

Example 3.20 (Cop #504)**.** Consider the TRS $\mathcal{R}$ consisting of the following three rewrite rules:

$$\begin{aligned} x + y &\to y + x \\ (x + y) * z &\to (x * z) + (y * z) \\ (y + x) * z &\to (x * z) + (y * z) \end{aligned}$$

Since the four critical pairs of $\mathcal{R}$

$$\begin{array}{ll} (y + x) * z \leftarrow\!\rtimes\!\rightarrow (x * z) + (y * z) & (y * z) + (x * z) \leftarrow\!\rtimes\!\rightarrow (x * z) + (y * z) \\ (x + y) * z \leftarrow\!\rtimes\!\rightarrow (x * z) + (y * z) & (x * z) + (y * z) \leftarrow\!\rtimes\!\rightarrow (y * z) + (x * z) \end{array}$$

are parallel closed, $\mathcal{R}$ is confluent.

3.4 Almost Parallel Closed Critical Pairs

In this section we consider two extensions to Huet's result due to Toyama [107]. The first one allows us to weaken the joining condition for some critical pairs.

When carefully examining the proof of Theorem 3.11 one realizes that in the case where both steps of the peak are single root steps, i.e., the case where $C = D = \Box$, the induction hypothesis does not need to be applied, since closing the critical pair immediately closes the whole peak. This suggests that the joining condition can be weakened for overlays. A first idea could be to take $\leftarrow\!\bowtie\!\rightarrow \subseteq \twoheadrightarrow \cdot \twoheadleftarrow$ since then we would still have the diamond property in the overlay case. However, Toyama realized that one can do even better by weakening the diamond property to strong confluence. The following definition captures the new conditions.

Definition 3.21. A TRS $\mathcal{R}$ is *almost parallel closed* if $s \twoheadrightarrow \cdot {}^*\!\leftarrow t$ for all overlays $s \leftarrow\!\bowtie\!\rightarrow t$ and $s \twoheadrightarrow t$ for all inner critical pairs $s \leftarrow\!\rtimes\!\rightarrow t$.

Using exactly the same proof structure as before we could now prove strong confluence of $\twoheadrightarrow$ for left-linear almost parallel closed systems. However, considering Toyama's second extension of Theorem 3.11, we will prove the theorem in the more general setting of commutation.

Theorem 3.22 (Toyama [107])**.** *Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be left-linear TRSs. If we have $s \twoheadrightarrow_2 \cdot {}^*_1\!\leftarrow t$ for all critical pairs $s\ {}_1\!\leftarrow\!\rtimes\!\rightarrow_2 t$ and additionally $s \twoheadrightarrow_1 t$ for all inner critical pairs $s\ {}_2\!\leftarrow\!\rtimes\!\rightarrow_1 t$ then $\twoheadrightarrow_1$ and $\twoheadrightarrow_2$ strongly commute.*

Proof Adaptations. We only highlight the differences to the proof of Theorem 3.11 and refer to the formalization for the full proof details. Assume

$$t\ {}_1\!\xleftarrow{C,\overline{a}}\!\!\!\shortmid\ s\ \shortmid\!\!\!\xrightarrow{D,\overline{b}}{}_2\ u$$

We show $t \twoheadrightarrow_2 v\ {}^*_1\!\leftarrow u$ for some term v. We apply the same inductions and case analyses as before. The cases $C = f(C_1, \ldots, C_n)$, $D = f(D_1, \ldots, D_n)$ and $C = D = \Box$ require no noteworthy adaptation. The main difference is that now the cases $D = f(D_1, \ldots, D_n)$, $C = \Box$ and $C = f(C_1, \ldots, C_n)$, $D = \Box$ become asymmetric for the critical pair case—the corresponding diagrams are shown in Figure 5. First, suppose $C = f(C_1, \ldots, C_n)$ and $D = \Box$, write

$$t = f(t_1, \ldots, t_n)\ {}_1\!\xleftarrow{C,\overline{a}}\!\!\!\shortmid\ s = \ell\sigma \xrightarrow{\epsilon}_2 r\sigma = u$$

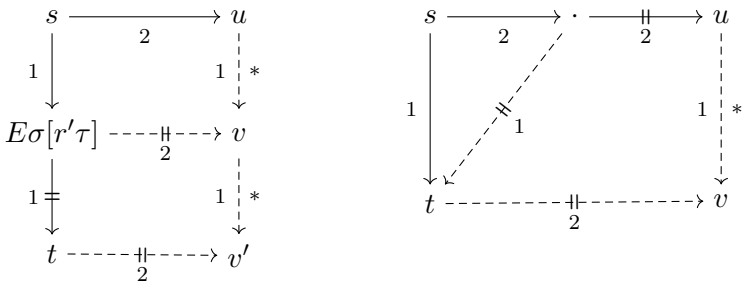


Figure 5: Asymmetry in the proof of Theorem 3.22.

and assume there is a critical pair according to Lemma 3.19. That is, we obtain $E\mu[r'\mu]\ {}_1\!\leftarrow\!\rtimes\!\rightarrow_2 r\mu$ with $E\sigma[r'\tau] \twoheadrightarrow_1 t$ and by assumption we obtain a v such that $E\sigma[r'\tau] \twoheadrightarrow_2 v\ {}^*_1\!\leftarrow r\sigma$. Then using the same reasoning as before, for the new peak

$$t\ {}_1\!\xleftarrow{C',\overline{a}'} E\sigma[r'\tau] \twoheadrightarrow_2 v$$

we have

$$\blacktriangle\left(\xleftarrow{C,\overline{a}} s \xrightarrow{\Box,\ell\sigma}\right) \rhd_{\mathsf{mul}} \blacktriangle\left(\xleftarrow{C',\overline{a'}} E\sigma[r'\tau] \twoheadrightarrow\right)$$

and can apply the induction hypothesis to obtain a v' with $t \twoheadrightarrow_2 v'\ {}^*_1\!\leftarrow v$, which combined with $u = r\sigma \rightarrow_1^* v$ concludes this case.

In the second case, i.e., when $D = f(D_1, \ldots, D_n)$ and $C = \Box$, observe that the critical pair we obtain is an inner critical pair between $\mathcal{R}_2$ and $\mathcal{R}_1$, since $D \neq \Box$. Thus, after applying the assumption for critical pairs ${}_2\!\leftarrow\!\rtimes\!\rightarrow_1$, the proof is the same as for Theorem 3.11. □

Instantiating $\mathcal{R}_1$ and $\mathcal{R}_2$ with the same TRS $\mathcal{R}$ yields the corresponding result for confluence.

Corollary 3.23 (Toyama [107])**.** *If the TRS $\mathcal{R}$ is left-linear and almost parallel closed then $\twoheadrightarrow_{\mathcal{R}}$ is strongly confluent.*

Proof. Immediate from the definition of almost parallel closed, Theorem 3.22 and the fact that $s \leftarrow\!\rtimes\!\rightarrow t$ if and only if $s \leftarrow\!\bowtie\!\rightarrow t$ or $s \leftarrow\!\rtimes\!\rightarrow t$. □

Example 3.24. Recall the rewrite system from Example 3.12. One easily verifies that all its critical pairs are almost parallel closed, and hence it is confluent.

3.5 Critical Pair Closing Systems

The key idea in this section, originally due to Oyamaguchi and Hirokawa [82], is to consider the subsystem of rewrite rules employed in the joining sequences of critical pairs. Existence of such a subsystem is a necessary condition for commutation of left-linear TRSs. It can be strengthened to sufficient conditions by imposing restrictions like linearity or termination. The resulting criteria generalize parallel closedness and strong closedness.

Definition 3.25. Let $\mathcal{R}$ and $\mathcal{S}$ be TRSs. A subsystem $\mathcal{C}$ of $\mathcal{R} \cup \mathcal{S}$ is *critical-pair-closing* if

$$(\mathcal{R}{\leftarrow}{\rtimes}{\rightarrow}\mathcal{S}) \cup (\mathcal{R}{\leftarrow}{\ltimes}{\rightarrow}\mathcal{S}) \subseteq {\rightarrow}_{\mathcal{S}\cap\mathcal{C}}^{*} \cdot {}_{\mathcal{R}\cap\mathcal{C}}^{*}{\leftarrow}$$

Existence of a critical-pair-closing system generalizes the critical pair lemma.

Theorem 3.26. *Left-linear TRSs that admit a critical-pair-closing system locally commute.*

Proof. Immediate from Lemma 3.3 and the definition of critical-pair-closing system. □

Additionally requiring right-linearity allows for a result based on strong commutation.

Theorem 3.27. *Linear TRSs $\mathcal{R}$ and $\mathcal{S}$ commute if the inclusion*

$$(\mathcal{R}\cap\mathcal{C}{\leftarrow}{\rtimes}{\rightarrow}\mathcal{S}\cap\mathcal{C}) \cup (\mathcal{R}\cap\mathcal{C}{\leftarrow}{\ltimes}{\rightarrow}\mathcal{S}\cap\mathcal{C}) \subseteq {\rightarrow}_{\mathcal{S}\cap\mathcal{C}}^{=} \cdot {}_{\mathcal{R}\cap\mathcal{C}}^{*}{\leftarrow}$$

holds for a critical-pair-closing system $\mathcal{C}$ for $\mathcal{R}$ and $\mathcal{S}$.

Proof. The proof works by analyzing peaks between $\mathcal{R}$, $\mathcal{S}$, $\mathcal{R} \cap \mathcal{C}$, and $\mathcal{S} \cap \mathcal{C}$, eventually showing that $\rightarrow^{=} \cdot \rightarrow^{*}$ has the diamond property. We have

(a) ${}_{\mathcal{R}\cap\mathcal{C}}^{*}{\leftarrow} \cdot {\rightarrow}_{\mathcal{S}\cap\mathcal{C}}^{*} \subseteq {\rightarrow}_{\mathcal{S}\cap\mathcal{C}}^{*} \cdot {}_{\mathcal{R}\cap\mathcal{C}}^{*}{\leftarrow}$

(b) $\mathcal{R}{\leftarrow} \cdot {\rightarrow}\mathcal{S} \subseteq {\rightarrow}_{\mathcal{S}}^{=} \cdot {\rightarrow}_{\mathcal{S}\cap\mathcal{C}}^{*} \cdot {}_{\mathcal{R}\cap\mathcal{C}}^{*}{\leftarrow} \cdot {}_{\mathcal{R}}^{=}{\leftarrow}$

(c) $\mathcal{R}\cap\mathcal{C}{\leftarrow} \cdot {\rightarrow}\mathcal{S} \subseteq {\rightarrow}_{\mathcal{S}}^{=} \cdot {\rightarrow}_{\mathcal{S}\cap\mathcal{C}}^{*} \cdot {}_{\mathcal{R}\cap\mathcal{C}}^{*}{\leftarrow}$

(d) $\mathcal{S}\cap\mathcal{C}{\leftarrow} \cdot {\rightarrow}\mathcal{R} \subseteq {\rightarrow}_{\mathcal{R}}^{=} \cdot {\rightarrow}_{\mathcal{R}\cap\mathcal{C}}^{*} \cdot {}_{\mathcal{S}\cap\mathcal{C}}^{*}{\leftarrow}$

(e) $\mathcal{R}\cap\mathcal{C}^*\!\leftarrow \cdot \to_{\mathcal{S}} \subseteq \to_{\overline{\mathcal{S}}}^{=} \cdot \to_{\mathcal{S}\cap\mathcal{C}}^* \cdot \mathcal{R}\cap\mathcal{C}^*\!\leftarrow$

(f) $\mathcal{S}\cap\mathcal{C}^*\!\leftarrow \cdot \to_{\mathcal{R}} \subseteq \to_{\overline{\mathcal{R}}}^{=} \cdot \to_{\mathcal{R}\cap\mathcal{C}}^* \cdot \mathcal{S}\cap\mathcal{C}^*\!\leftarrow$

(g) $\mathcal{R}\cap\mathcal{C}^*\!\leftarrow \cdot \overline{\mathcal{R}}\!\leftarrow \cdot \to_{\overline{\mathcal{S}}} \cdot \to_{\mathcal{S}\cap\mathcal{C}}^* \subseteq \to_{\overline{\mathcal{S}}}^{=} \cdot \to_{\mathcal{S}\cap\mathcal{C}}^* \cdot \mathcal{R}\cap\mathcal{C}^*\!\leftarrow \cdot \overline{\mathcal{R}}\!\leftarrow$

For item (a) we show that $\mathcal{R}\cap\mathcal{C}$ and $\mathcal{S}\cap\mathcal{C}$ strongly commute. This follows from the usual peak analysis, using right-linearity to avoid duplication of redexes in case one step takes place in the substitution of the other, like in the proof of Lemma 3.6, and the assumption on critical pairs between $\mathcal{R}\cap\mathcal{C}$ and $\mathcal{S}\cap\mathcal{C}$ in case of a critical overlap.

For item (b) the same peak analysis yields $\mathcal{R}\!\leftarrow \cdot \to_{\mathcal{S}} \subseteq \to_{\overline{\mathcal{S}}}^{=} \cdot \overline{\mathcal{R}}^{=}\!\leftarrow$ in case the steps happen at parallel positions or one is in the substitution of the other and, since $\mathcal{C}$ is critical-pair-closing, $\mathcal{R}\!\leftarrow \cdot \to_{\mathcal{S}} \subseteq \to_{\mathcal{S}\cap\mathcal{C}}^* \cdot \mathcal{R}\cap\mathcal{C}^*\!\leftarrow$ in case of a critical overlap. Items (c) and (d) are straightforward variations of item (b), where the joining steps from the right can be merged into one sequence, since the step to the left is restricted to $\mathcal{C}$.

Items (e) and (f) follow from items (c) and (d) respectively, using an induction on the length of the rewrite sequence to the left and item (a), as shown in Figure 6(a). Note that if the closing $\to_{\overline{\mathcal{S}}}^{=}$-step, obtained from item (c), is empty applying the induction hypothesis is omitted.

Item (g) can then be shown from items (b), (e), (f), and (a) by closing the peak as depicted in Figure 6(b). Again, if any of the $\to^{=}$-steps are empty the respective parts of the diagram disappear.

Finally we conclude commutation using Lemma 2.17 and the two inclusions $\to_{\mathcal{R}} \subseteq \to_{\overline{\mathcal{R}}}^{=} \cdot \to_{\mathcal{R}\cap\mathcal{C}}^* \subseteq \to_{\mathcal{R}}^*$ and $\to_{\mathcal{S}} \subseteq \to_{\overline{\mathcal{S}}}^{=} \cdot \to_{\mathcal{S}\cap\mathcal{C}}^* \subseteq \to_{\mathcal{S}}^*$. □

The corresponding corollary for confluence generalizes Huet's result on strongly closed TRSs.

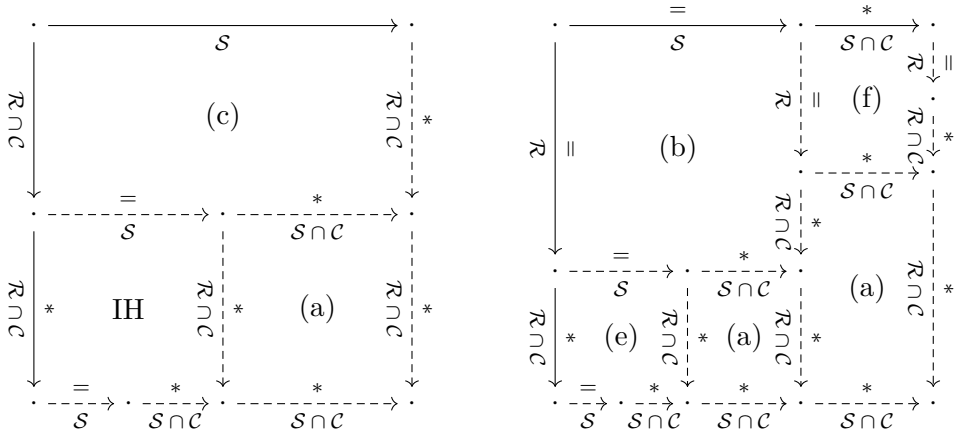
Corollary 3.28. *A linear TRS $\mathcal{R}$ is confluent if it admits a strongly closed critical-pair-closing system $\mathcal{C}$.* □

Example 3.29 (Cop #760)**.** Consider the linear $\mathcal{R}$ TRS consisting of the following three rules:

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

There are two critical pairs

$$\mathsf{f}(\mathsf{f}(x)) \leftarrow\!\rtimes\!\rightarrow \mathsf{g}(x) \qquad \mathsf{g}(x) \leftarrow\!\rtimes\!\rightarrow \mathsf{f}(\mathsf{f}(x))$$



(a) Proof of Theorem 3.27(e). (b) Proof of Theorem 3.27(g).

Figure 6: Proof of Theorem 3.27.

Since $\mathsf{g}(x) \to \mathsf{f}(x) \to \mathsf{f}(\mathsf{f}(x))$ by the first two rules, they constitute a critical-pair-closing system. Moreover they do not admit critical pairs and hence $\mathcal{R}$ is confluent by Corollary 3.28. Note that $\mathcal{R}$ is not strongly closed, because that would require a join $\mathsf{g}(x) \to^{=} \cdot {}^{*}\!\leftarrow \mathsf{f}(\mathsf{f}(x))$. To see that a join of this shape does not exist, first observe that the only terms reachable from $\mathsf{g}(x)$ by $\to^{=}$ are of size 2, namely $\mathsf{f}(x)$ and $\mathsf{g}(x)$ itself. However $|s| > 2$ for all s with $\mathsf{f}(\mathsf{f}(x) \to^{*} s$, since $|\ell| \leqslant |r|$ for all rules $\ell \to r$ in $\mathcal{R}$.

Next we turn to a variation of parallel closedness that requires $\mathcal{C}$ to be terminating.

Theorem 3.30. *Left-linear TRSs $\mathcal{R}$ and $\mathcal{S}$ commute if the inclusions*

$$\mathcal{R}{\leftarrow}{\bowtie}{\rightarrow}\mathcal{S} \subseteq {\twoheadrightarrow}_{\mathcal{S}\cap\mathcal{C}} \cdot {}_{\mathcal{R}\cap\mathcal{C}}^{*}{\leftarrow} \qquad \textit{and} \qquad \mathcal{R}{\leftarrow}{\ltimes}{\rightarrow}\mathcal{S} \subseteq {\to}_{\mathcal{S}\cap\mathcal{C}}^{*} \cdot \mathcal{R}\cap\mathcal{C}{\twoheadleftarrow}$$

hold for some terminating critical-pair-closing system $\mathcal{C}$ for $\mathcal{R}$ and $\mathcal{S}$.

Note that, since $\mathcal{C}$ is required to be a critical-pair-closing system, the theorem demands ${\mathcal{R}}{\leftarrow}{\bowtie}{\rightarrow}\mathcal{S} \subseteq {\to}_{\mathcal{S}\cap\mathcal{C}}^{*} \cdot {}_{\mathcal{R}\cap\mathcal{C}}^{*}{\leftarrow}$ for overlays. Before proving it we illustrate its power on an example.

Example 3.31 (Cop #62)**.** Consider the left-linear TRS $\mathcal{R}$ from [83]:

$$
\begin{array}{rrcl@{\qquad}rrcl}
1\colon & x - 0 & \to & x & 2\colon & 0 - x & \to & 0 \\
3\colon & \mathsf{s}(x) - \mathsf{s}(y) & \to & x - y & 4\colon & x < 0 & \to & \mathsf{false} \\
5\colon & 0 < \mathsf{s}(y) & \to & \mathsf{true} & 6\colon & \mathsf{s}(x) < \mathsf{s}(y) & \to & x < y \\
7\colon & \mathsf{if}(\mathsf{true}, x, y) & \to & x & 8\colon & \mathsf{if}(\mathsf{false}, x, y) & \to & y \\
9\colon & \mathsf{gcd}(x, 0) & \to & x & 10\colon & \mathsf{gcd}(0, x) & \to & x \\
11\colon & \mathsf{gcd}(x, y) & \to & \mathsf{gcd}(y, \mathsf{mod}(x, y)) & 12\colon & \mathsf{mod}(x, 0) & \to & x \\
13\colon & \mathsf{mod}(0, y) & \to & 0 & & & & \\
14\colon & \mathsf{mod}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{if}(x < y, \mathsf{s}(x), \mathsf{mod}(x - y, \mathsf{s}(y))) & & & &
\end{array}
$$

Let $\mathcal{C} = \{9, 10, 12, 13\}$. The system $\mathcal{C}$ is critical-pair-closing for $\mathcal{R}$ and obviously terminating. Since $\mathcal{R}$ admits no inner critical pairs it is confluent by Theorem 3.30.

To prove Theorem 3.30 we show the diamond property of $\to^* \cdot \mathrel{+\!\!\!\Rightarrow}$.

Lemma 3.32. *Let $\mathcal{R}$ and $\mathcal{S}$ be left-linear TRSs satisfying the assumptions of Theorem 3.30. Then we have*

(a) $\mathcal{R} \cap \mathcal{C}^*\!\leftarrow \cdot \to^*_{\mathcal{S} \cap \mathcal{C}} \subseteq \to^*_{\mathcal{S} \cap \mathcal{C}} \cdot \mathcal{R} \cap \mathcal{C}^*\!\leftarrow$

(b) $\mathcal{R} \mathrel{+\!\!\!\Leftarrow} \cdot \mathrel{+\!\!\!\Rightarrow}_{\mathcal{S}} \subseteq \to^*_{\mathcal{S} \cap \mathcal{C}} \cdot \mathrel{+\!\!\!\Rightarrow}_{\mathcal{S}} \cdot \mathcal{R} \mathrel{+\!\!\!\Leftarrow} \cdot \mathcal{R} \cap \mathcal{C}^*\!\leftarrow$

(c) $\mathcal{R} \cap \mathcal{C} \mathrel{+\!\!\!\Leftarrow} \cdot \mathrel{+\!\!\!\Rightarrow}_{\mathcal{S}} \subseteq \to^*_{\mathcal{S} \cap \mathcal{C}} \cdot \mathrel{+\!\!\!\Rightarrow}_{\mathcal{S}} \cdot \mathcal{R} \cap \mathcal{C}^*\!\leftarrow$

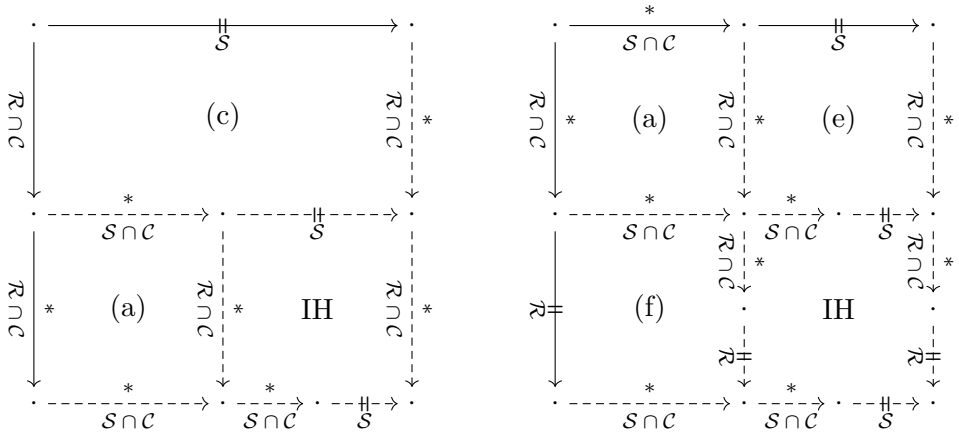
(d) $\mathcal{S} \cap \mathcal{C} \mathrel{+\!\!\!\Leftarrow} \cdot \mathrel{+\!\!\!\Rightarrow}_{\mathcal{R}} \subseteq \to^*_{\mathcal{R} \cap \mathcal{C}} \cdot \mathrel{+\!\!\!\Rightarrow}_{\mathcal{R}} \cdot \mathcal{S} \cap \mathcal{C}^*\!\leftarrow$

(e) $\mathcal{R} \cap \mathcal{C}^*\!\leftarrow \cdot \mathrel{+\!\!\!\Rightarrow}_{\mathcal{S}} \subseteq \to^*_{\mathcal{S} \cap \mathcal{C}} \cdot \mathrel{+\!\!\!\Rightarrow}_{\mathcal{S}} \cdot \mathcal{R} \cap \mathcal{C}^*\!\leftarrow$

(f) $\mathcal{S} \cap \mathcal{C}^*\!\leftarrow \cdot \mathrel{+\!\!\!\Rightarrow}_{\mathcal{R}} \subseteq \to^*_{\mathcal{R} \cap \mathcal{C}} \cdot \mathrel{+\!\!\!\Rightarrow}_{\mathcal{R}} \cdot \mathcal{S} \cap \mathcal{C}^*\!\leftarrow$

(g) $\mathcal{R} \mathrel{+\!\!\!\Leftarrow} \cdot \mathcal{R} \cap \mathcal{C}^*\!\leftarrow \cdot \to^*_{\mathcal{S} \cap \mathcal{C}} \cdot \mathrel{+\!\!\!\Rightarrow}_{\mathcal{S}} \subseteq \to^*_{\mathcal{S} \cap \mathcal{C}} \cdot \mathrel{+\!\!\!\Rightarrow}_{\mathcal{S}} \cdot \mathcal{R} \mathrel{+\!\!\!\Leftarrow} \cdot \mathcal{R} \cap \mathcal{C}^*\!\leftarrow$

Proof Sketch. We only give the outline of the proof structure and refer to the formalization for full details. Item (a) follows from local commutation of $\mathcal{R} \cap \mathcal{C}$ and $\mathcal{S} \cap \mathcal{C}$, termination of $\mathcal{C}$, and a commutation version of Newman's Lemma. Items (b), (c), and (d) are proved by an induction on the amount of overlap between the steps in the peak, similar to the proof of Theorem 3.22.



(a) Proof of Lemma 3.32(e). (b) Proof of Lemma 3.32(g).

Figure 7: Proof of Lemma 3.32.

Items (e) and (f) follow from items (c) and (d) using well-founded induction on the source of the peak with respect to $\to_{\mathcal{C}}^{+}$, see Figure 7(a). The proof for item (g) also proceeds by well-founded induction on the source of the peak with respect to $\to_{\mathcal{C}}^{+}$ applying item (b) if both $\mathcal{C}$-sequences are empty and items (a), (e), and (f) if they are not, as depicted in Figure 7(b). □

Now the main theorem follows easily.

Proof of Theorem 3.30. From Lemma 3.32(g) and the two obvious inclusions $\to_{\mathcal{R}} \subseteq \to_{\mathcal{R}\cap\mathcal{C}}^{*} \cdot \twoheadrightarrow_{\mathcal{R}} \subseteq \to_{\mathcal{R}}^{*}$ and $\to_{\mathcal{S}} \subseteq \to_{\mathcal{S}\cap\mathcal{C}}^{*} \cdot \twoheadrightarrow_{\mathcal{S}} \subseteq \to_{\mathcal{S}}^{*}$ we obtain the commutation of $\mathcal{R}$ and $\mathcal{S}$ from Lemma 2.17. □

3.6 Certificates

To facilitate checking of confluence proofs generated by automatic tools based on Corollary 3.7, Corollary 3.23, and Theorem 3.30, we extended the CPF to represent such proofs. Since in order to check the closing conditions for the critical pairs, CeTA has to compute them anyway, we do not require them in the certificate. We just require the claim that the system is strongly or almost parallel closed, together with a bound on the length of the rewrite sequences to

join the critical pairs, which is necessary to ensure termination of the certifier. In case of Theorem 3.30 the certificate contains the critical-pair-closing system $\mathcal{C}$ together with a termination proof.[2] Certificates for commutation are not yet supported, since currently no tool produces them, and CPF does not contain a specification for commutation proofs.

3.7 Summary

In this chapter we presented the formalization of three classical criteria for confluence and commutation of (left-)linear rewrite systems. Building on top of IsaFoR we formalized proofs that linear strongly closed systems, and left-linear (almost) parallel closed systems are confluent (commute). The major difference to the original paper proof is our definition of the overlap between two parallel steps that are represented via multihole contexts. Finally we extended the criteria using critical-pair-closing systems.

As expected IsaFoR provided invaluable support on the one hand, e.g. by its theories on critical pairs and multihole contexts, and on the other hand, it was also extended with new facts about lists, multisets, multihole contexts, etc.

Concerning future work, another important extension of the results of Huet and Toyama due to van Oostrom [78] is using multisteps $\multimap$ which allow nested non-overlapping redexes. This extension not only strengthens Huet's criterion in the first-order world but also makes it applicable to higher-order rewriting, where using parallel steps fails due to β-reduction.

Definition 3.33. A TRS $\mathcal{R}$ is *development closed* if all critical pairs $s \leftarrow\!\rtimes\!\rightarrow t$ of $\mathcal{R}$ satisfy $s \multimap_{\mathcal{R}} t$. It is *almost development closed* if $s \multimap \cdot {}^*\!\leftarrow t$ for all overlays $s \leftarrow\!\bowtie\!\rightarrow t$ and $s \multimap t$ for all inner critical pairs $s \leftarrow\!\bowtie\!\rightarrow t$.

Theorem 3.34 (van Oostrom [78])**.** *Every left-linear and almost development closed TRS is confluent.* □

A similar extension is possible for the results on critical-pair-closing systems. However, although the paper proofs superficially look very similar, and do employ similar ideas, obtaining a formalized proof will require serious effort. In fact neither our representation of (parallel) rewrite steps, nor our definition of ▲, nor the idea of using an induction on the source of the peak to avoid

[2] CeTA contains support for a wide range of termination criteria that is easy to reuse.

reasoning about combining steps, carry over. To make the concepts that are hard to formalize in a proof assistant, e.g. measuring the amount of overlap between two multisteps or the descendants of a multistep, Hirokawa and Middeldorp [43] suggested to use proof terms to obtain a rigorous proof (and at the same time extended the result to commutation). This is a step forward but more is needed to obtain a formalized proof, also for the extension to higher-order systems. In particular, we anticipate the extensive use of sets of positions (in [43]) to be problematic without alternative notions. We plan to employ residual theory [105, Section 8.7] and to develop a notion of overlap for multisteps similar to Definition 3.15 to close the gap.

Chapter 4

Rule Labeling

In mathematics you don't understand things.
You just get used to them.
John von Neumann

The rule labeling heuristic aims to establish confluence of (left-)linear term rewrite systems via decreasing diagrams. In this chapter we present a formalization of a confluence criterion based on the interplay of relative termination and the rule labeling in Isabelle. Moreover, we report on the integration of this result into CeTA, facilitating the checking of confluence certificates based on decreasing diagrams.

Decreasing diagrams [76] provide a complete characterization of confluence for abstract rewrite systems whose convertibility classes are countable. This confluence criterion states that a labeled abstract rewrite system is confluent if each of its local peaks can be *joined decreasingly* (see Figure 8(a), where $\vee\alpha$ indicates that any label below α may be used, and $>$ is a well-founded order on the labels. We will introduce these notions formally in Section 4.1). As a criterion for abstract rewrite systems, decreasing diagrams can be applied to first- and higher-order rewriting, including term rewriting and the λ-calculus. Van Oostrom [79] presented the rule labeling heuristic, which labels rewrite steps by their rules and yields a confluence criterion for linear term rewrite systems by checking decreasingness of the critical peaks. The rule labeling has successfully been implemented by Aoto [2] and Hirokawa and Middeldorp [42].

We consider a recent powerful confluence result for term rewrite systems, exploiting relative termination and decreasing diagrams that makes the rule labeling applicable to left-linear system. Our aim was to formalize [115, Corollary 16] for a specific labeling culminating in Theorem 4.6. Actually the formal proof of Theorem 4.6 is obtained by instantiating the more general result of Theorem 4.27 appropriately. Hence, confluence proofs according to

these theorems are now checkable by CeTA. We achieved this via the following steps:

- Build on local decreasingness for abstract rewrite systems [23, 28, 113].
- Perform a detailed analysis of how local peaks can be joined for linear and left-linear term rewrite systems (Section 4.2.1).
- Use the detailed analysis of local peaks to formalize the notion of a labeling and a confluence result for term rewrite systems parametrized by a labeling (Section 4.2.2). In this way it is ensured that the formalization is reusable for other labelings stemming from [115].
- Instantiate the result from Section 4.2.2 to obtain concrete confluence results. We demonstrate how this instantiation can be done, culminating in a formal proof of Theorem 4.27 (Section 4.3).
- Finally we made our formalization executable to check proof certificates: we suitably extended CPF to represent proofs according to Theorem 4.27 and implemented dedicated check functions in our formalization, enabling CeTA to inspect, i.e., certify, such confluence proofs (Section 4.3.3).

The remainder of this chapter is organized as follows. Additional preliminaries are introduced in the next section. Based on results for abstract rewriting and the notion of a labeling, Section 4.2 formulates conditions that limit the attention to critical peaks rather than arbitrary local peaks in the case of first-order term rewriting. Section 4.3 instantiates these results with concrete labelings to obtain corollaries that ensure confluence.

4.1 Preliminaries

The decreasing diagrams technique is based on labeling each rewrite step such that all local peaks can be joined decreasingly. In order to assign such labels to steps we will need to decorate them with more information. If $\ell \to r \in \mathcal{R}$, p is a position, and σ is a substitution we call the triple $\pi = \langle p, \ell \to r, \sigma \rangle$ a *redex pattern*, and write p_π, ℓ_π, r_π, σ_π for its position, left-hand side, right-hand side, and substitution, respectively. We write $\to^\pi$ (or $\to^{p_\pi, \ell_\pi \to r_\pi, \sigma_\pi}$) for a rewrite step at position p_π using the rule $\ell_\pi \to r_\pi$ and the substitution σ_π. A redex pattern π *matches* a term t if $t|_{p_\pi} = \ell_\pi\sigma_\pi$, in which case $t|_{p_\pi}$ is called the

redex. Let π_1 and π_2 be redex patterns that match a common term. They are called *parallel*, written $\pi_1 \parallel \pi_2$, if $p_{\pi_1} \parallel p_{\pi_2}$. If $P = \{\pi_1, \pi_2, \ldots, \pi_n\}$ is a set of pairwise parallel redex patterns matching a term t, we denote by $t \mathrel{+\!\!\!\Rightarrow}^P t'$ the *parallel rewrite step* from t to t' by P, i.e., $t \to^{\pi_1} \cdot \to^{\pi_2} \cdots \to^{\pi_n} t'$.

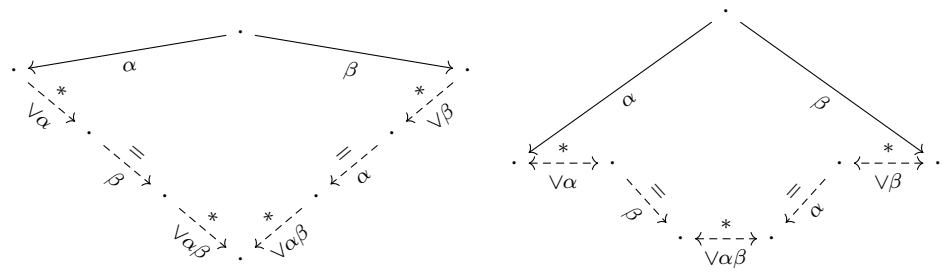
On the abstract rewriting level we label steps by presenting the rewrite relation as a family of labeled steps. Let I be an index set. We write $\{\to_\alpha\}_{\alpha\in I}$ to denote the ARS $\to$ where $\to$ is the union of $\to_\alpha$ for all $\alpha \in I$. Let $\{\to_\alpha\}_{\alpha\in I}$ be an ARS and let $>$ and $\geqslant$ be relations on I. Two relations $>$ and $\geqslant$ are called *compatible* if $\geqslant \cdot > \cdot \geqslant \subseteq >$. Given a relation $\succcurlyeq$ we write $\to_{\curlyvee\alpha_1 \cdots \alpha_n}$ for the union of all $\to_\beta$ with $\alpha_i \succcurlyeq \beta$ for some $1 \leqslant i \leqslant n$. Similarly, $\curlyvee S$ is the set of all β such that $\alpha \succcurlyeq \beta$ for some $\alpha \in S$. We call α and β *extended locally decreasing* (for $>$ and $\geqslant$) if ${}_\alpha\!\leftarrow \cdot \to_\beta \subseteq \leftrightarrow^*_{\curlyvee\alpha} \cdot \to^{=}_{\curlywedge\beta} \cdot \leftrightarrow^*_{\curlyvee\alpha\beta} \cdot {}^{=}_{\curlywedge\alpha}\!\leftarrow \cdot \leftrightarrow^*_{\curlyvee\beta}$. If there exist a well-founded order $>$ and a preorder $\geqslant$, such that $>$ and $\geqslant$ are compatible, and α and β are extended locally decreasing for all $\alpha, \beta \in I$ then the ARS $\{\to_\alpha\}_{\alpha\in I}$ is *extended locally decreasing* (for $>$ and $\geqslant$). We call an ARS *locally decreasing* (for $>$) if it is extended locally decreasing for $>$ and $=$, where the latter is the identity relation. In the sequel, we often refer to extended locally decreasing as well as to locally decreasing just by decreasing, whenever the context clarifies which concept is meant or the exact meaning is irrelevant. In the literature the above condition is referred to as *the conversion version of decreasing diagrams*, Figure 8(b). In its *valley version*, Figure 8(a), the condition reads as follows: ${}_\alpha\!\leftarrow \cdot \to_\beta \subseteq \to^*_{\curlyvee\alpha} \cdot \to^{=}_{\curlywedge\beta} \cdot \to^*_{\curlyvee\alpha\beta} \cdot {}^*_{\curlyvee\alpha\beta}\!\leftarrow \cdot {}^{=}_{\curlywedge\alpha}\!\leftarrow \cdot {}^*_{\curlyvee\beta}\!\leftarrow$. In the sequel we always refer to the conversion version of decreasing diagrams, except when stated otherwise explicitly. The main result on decreasing diagrams is due to van Oostrom.

Theorem 4.1 (van Oostrom [79])**.** *Locally decreasing ARSs are confluent.* □

In the remainder of this section we are concerned with the formalization of the following variation of the result from [41, Theorem 2]:

Lemma 4.2. *Every extended locally decreasing ARS is confluent.*

It is interesting to note that in [42], the journal version of [41], extended local decreasingness is avoided by employing the *predecessor labeling*. Originally the authors used the *source labeling*, wherein a step $s \to t$ is labeled by s, and $\geqslant = \to^*$ for the weak order. Consequently, any rewrite step $s' \to t'$ with $s \to^* s'$ has a label s' with $s \geqslant s'$. In the predecessor labeling, a rewrite step can be labeled by an arbitrary predecessor, i.e., we have $s \to_u t$ if $u \to^* s$.



(a) Valley version. (b) Conversion version.

Figure 8: Locally decreasing diagrams.

Now, if $s \to_u t$ and $s \to^* s' \to t'$, we may label the step from s' to t' by u, the same label as $s \to_u t$. In this way, the need for the weak comparison $\geqslant$, and hence *extended* local decreasingness, is avoided. As the proof of Lemma 4.2 demonstrates, this idea works in general, i.e., extended local decreasingness can be traded for assigning more than one label to each rewrite step. In the context of automation, however, assigning just one (computable) label to every rewrite step is very attractive, as confluence tools must show critical peaks decreasing and confluence certifiers must check the related proof certificates.

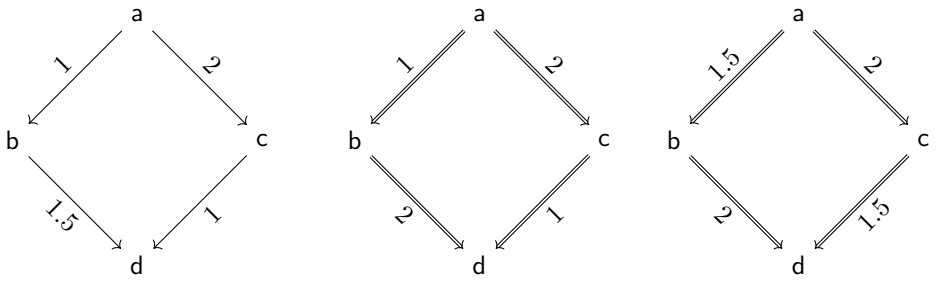
In the rest of this section, we describe our formalized proof of Lemma 4.2. Given an ARS that is extended locally decreasing for $>$ and $\geqslant$, the proof in [41] constructs a single order $\succ$ on sets of labels and establishes local decreasingness of the ARS for $>$. We do the same, but use a simplified proof for the following key lemma, which allows us to use $>$ directly.

Lemma 4.3. *Every extended locally decreasing ARS is locally decreasing.*

The idea is to set $\Rightarrow_\alpha = \to_{\diagup\!\!\diagup\alpha}$ and show that $\Rightarrow$ is locally decreasing.[1] This establishes the claim because $\bigcup_{\alpha\in I} \Rightarrow_\alpha = \bigcup_{\alpha\in \diagup\!\!\diagup I} \to_\alpha = \bigcup_{\alpha\in I} \to_\alpha$, and therefore, $\Rightarrow = \{\Rightarrow_\alpha\}_{\alpha\in I}$ and $\to = \{\to_\alpha\}_{\alpha\in I}$ are the same ARS, labeled in two different ways. The next example demonstrates some peculiarities of this approach.

Example 4.4. Consider the ARS $\{\to_\alpha\}_{\alpha\in\{1,\,1.5,\,2\}}$ where $\to_1 = \{(\mathsf{a},\mathsf{b}),(\mathsf{c},\mathsf{d})\}$, $\to_{1.5} = \{(\mathsf{b},\mathsf{d})\}$, and $\to_2 = \{(\mathsf{a},\mathsf{c})\}$. This ARS is extended locally decreasing for

[1] Compared to [41], we use $\diagup\!\!\diagup\alpha$ instead of $C_\alpha = (\diagup\!\!\diagup\alpha) \setminus (\vee\alpha)$.



(a) Extended locally decreasing. (b) Locally decreasing. (c) Locally decreasing.

Figure 9: (Extended) locally decreasing peaks.

the orders $> = \{(x, y) \mid x, y \in \mathbb{Q}_{\geqslant 0} \text{ such that } x - y \geqslant 1\}$ and $\geqslant_{\mathbb{Q}}$, as depicted in Figure 9(a). We have

$$\underset{2}{\Rightarrow} = \{(\mathsf{a},\mathsf{b}),(\mathsf{c},\mathsf{d}),(\mathsf{a},\mathsf{c}),(\mathsf{b},\mathsf{d})\}$$
$$\underset{1.5}{\Longrightarrow} = \{(\mathsf{b},\mathsf{d}),(\mathsf{a},\mathsf{b}),(\mathsf{c},\mathsf{d})\}$$
$$\underset{1}{\Rightarrow} = \{(\mathsf{a},\mathsf{b}),(\mathsf{c},\mathsf{d})\}$$

where e.g. $\to_{1.5} \subseteq \Rightarrow_2$ since $2 \geqslant_{\mathbb{Q}} 1.5$. Consequently, $\Rightarrow = \Rightarrow_1 \cup \Rightarrow_{1.5} \cup \Rightarrow_2$. To establish local decreasingness of the related ARS $\{\Rightarrow_\alpha\}_{\alpha\in\{1,1.5,2\}}$ the peak $\mathsf{b} \;{}_1\!\Leftarrow \mathsf{a} \Rightarrow_2 \mathsf{c}$ (emerging from $\mathsf{b} \;{}_1\!\leftarrow \mathsf{a} \to_2 \mathsf{c}$) must be considered, which can be closed in a locally decreasing fashion via $\mathsf{b} \Rightarrow_2 \mathsf{d} \;{}_1\!\Leftarrow \mathsf{c}$ (based on $\mathsf{b} \to_{1.5} \mathsf{d} \;{}_1\!\leftarrow \mathsf{c}$), as in Figure 9(b). Note that $\mathsf{b} \Rightarrow_{1.5} \mathsf{d} \;{}_1\!\Leftarrow \mathsf{c}$ would not establish decreasingness for $>$. However, the construction also admits the peak $\mathsf{b} \;{}_{1.5}\!\Leftarrow \mathsf{a} \Rightarrow_2 \mathsf{c}$, for which there is no peak $\mathsf{b} \;{}_{1.5}\!\leftarrow \mathsf{a} \to_2 \mathsf{c}$ in the original ARS, as it does not contain the step $\mathsf{b} \;{}_{1.5}\!\leftarrow \mathsf{a}$. Still, this peak can be closed locally decreasing for $>$, see Figure 9(c), based on the steps in the diagram of Figure 9(a).

Proof of Lemma 4.3. We assume the ARS $\{\to_\alpha\}_{\alpha\in I}$ is extended locally decreasing for $>$ and $\geqslant$ and establish local decreasingness of the ARS $\{\Rightarrow_\alpha\}_{\alpha\in I}$ for $>$ by showing

$$\underset{\alpha}{\Leftarrow} \cdot \underset{\beta}{\Rightarrow} \subseteq \underset{\vee\alpha}{\overset{*}{\Leftrightarrow}} \cdot \underset{\beta}{\overset{=}{\Rightarrow}} \cdot \underset{\vee\alpha\beta}{\overset{*}{\Longleftrightarrow}} \cdot \underset{\alpha}{\overset{=}{\Leftarrow}} \cdot \underset{\vee\beta}{\overset{*}{\Leftrightarrow}} \tag{4.1}$$

for $\alpha, \beta \in I$. Assume that $x \;{}_\alpha\!\Leftarrow \cdot \Rightarrow_\beta y$. By the definition of $\Rightarrow$, there are $\alpha' \leqslant \alpha$ and $\beta' \leqslant \beta$ such that $x \;{}_{\alpha'}\!\leftarrow \cdot \to_{\beta'} y$. Because $\to$ is extended locally

decreasing, this implies that

$$x \underset{\vee\alpha'}{\overset{*}{\longleftrightarrow}} \cdot \underset{\vee/\beta'}{\overset{=}{\longrightarrow}} \cdot \underset{\vee\alpha'\beta'}{\overset{*}{\longleftarrow}} \cdot \underset{\vee/\alpha'}{\overset{=}{\longleftarrow}} \cdot \underset{\vee\beta'}{\overset{*}{\longleftrightarrow}} y$$

Consider a label $\gamma \in \vee\alpha'$, i.e., $\gamma < \alpha'$. By compatibility, this implies $\gamma < \alpha$, i.e., $\gamma \in \vee\alpha$. Consequently, $\leftrightarrow_{\vee\alpha'} \subseteq \Leftrightarrow_{\vee\alpha}$. Similarly, if $\gamma \in \vee/\beta'$ then $\gamma \in \vee/\beta$, because $\geqslant$ is transitive, and hence $\to^{=}_{\vee/\beta'} \subseteq \Rightarrow^{=}_{\beta}$. Continuing in this fashion we obtain

$$x \underset{\vee\alpha}{\overset{*}{\Longleftrightarrow}} \cdot \underset{\beta}{\overset{=}{\Longrightarrow}} \cdot \underset{\vee\alpha\beta}{\overset{*}{\Longleftrightarrow}} \cdot \underset{\alpha}{\overset{=}{\Longleftarrow}} \cdot \underset{\vee\beta}{\overset{*}{\Longleftrightarrow}} y$$

This establishes (4.1). Consequently, $\Rightarrow$ is locally decreasing. □

Proof of Lemma 4.2. Now confluence of extended locally decreasing ARSs immediately follows from Lemma 4.3 and Theorem 4.1. □

4.2 Formalized Confluence Results

This section builds upon the result for ARSs from the previous section to prepare for confluence criteria for TRSs. First we recall the rule labeling [79], which maps each rewrite step to a natural number based on the rewrite rule applied in the step.

Definition 4.5. Let $i \colon \mathcal{R} \to \mathbb{N}$ be an index mapping, which associates to every rewrite rule a natural number. The function $l^i(s \to^\pi t) = i(\ell_\pi \to r_\pi)$ is called *rule labeling*. Labels due to the rule labeling are compared by $>_\mathbb{N}$ and $\geqslant_\mathbb{N}$.

Our aim is to formalize the following theorem, which is one of the main results of [115].

Theorem 4.6 (Valley version of decreasing diagrams)**.** *A left-linear TRS is confluent if its duplicating rules terminate relative to its other rules and all its critical peaks are decreasing for the rule labeling.*

Before preparing for its proof, we illustrate the technique on an example.

Example 4.7 (Cop #20)**.** Consider the TRS $\mathcal{R}$ from [34, Example 2]. It consisting of the following five rewrite rules, where we indicate the index mapping i for the rule labeling in the subscripts:

$$\mathsf{hd}(x : y) \to_0 x \qquad \mathsf{inc}(x : y) \to_0 \mathsf{s}(x) : \mathsf{inc}(y) \qquad \mathsf{nats} \to_0 \mathsf{0} : \mathsf{inc}(\mathsf{nats})$$
$$\mathsf{tl}(x : y) \to_0 y \qquad \mathsf{inc}(\mathsf{tl}(\mathsf{nats})) \to_1 \mathsf{tl}(\mathsf{inc}(\mathsf{nats}))$$

The rules give rise to one critical peak that is labeled by l^i as

$$\mathsf{inc}(\mathsf{tl}(0 : \mathsf{inc}(\mathsf{nats}))) \;{}_0\!\leftarrow \mathsf{inc}(\mathsf{tl}(\mathsf{nats})) \rightarrow_1 \mathsf{tl}(\mathsf{inc}(\mathsf{nats}))$$

This peak can be closed using the valley

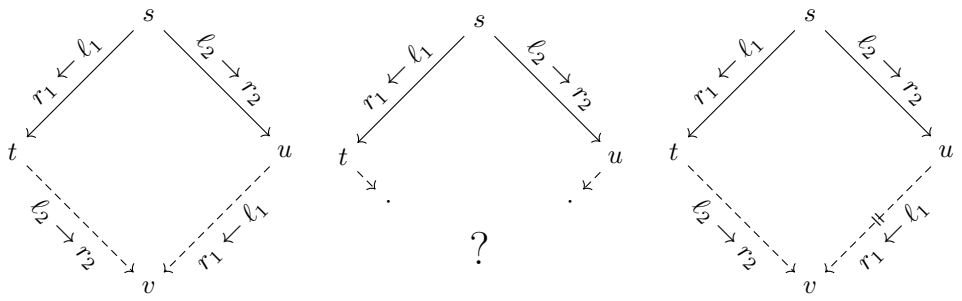
$$\begin{aligned}\mathsf{inc}(\mathsf{tl}(0 : \mathsf{inc}(\mathsf{nats}))) &\rightarrow_0 \mathsf{inc}(\mathsf{inc}(\mathsf{nats}))\\ &\;{}_0\!\leftarrow \mathsf{tl}(\mathsf{s}(0) : \mathsf{inc}(\mathsf{inc}(\mathsf{nats})))\\ &\;{}_0\!\leftarrow \mathsf{tl}(\mathsf{inc}(0 : \mathsf{inc}(\mathsf{nats})))\\ &\;{}_0\!\leftarrow \mathsf{tl}(\mathsf{inc}(\mathsf{nats}))\end{aligned}$$

Clearly all labels in the valley are dominated by label 1 in the peak. Hence the only critical peak of $\mathcal{R}$ is decreasing for the given rule labeling. Since $\mathcal{R}$ does not contain duplicating rules, the relative termination condition vacuously holds, and consequently $\mathcal{R}$ is confluent by Theorem 4.6.

To support other confluence results, in the formalization we did not follow the easiest path, i.e., suit the definitions and lemmas directly to Theorem 4.6. Rather, we adopted the approach from [115], where all results are established via *labeling functions* satisfying some abstract properties. Apart from avoiding a monolithic proof, this has the advantage that similar proofs need not be repeated for different labeling functions. Instead it suffices to establish that the concrete labeling functions satisfy some abstract conditions. In this approach decreasingness is established in three steps. The first step comprises joinability results for local peaks (Section 4.2.1). The second step (Section 4.2.2) formulates abstract conditions with the help of *labeling functions* that admit a finite characterization of decreasingness of local peaks. Finally, based on the previous two steps, the third step (Section 4.3) then obtains confluence results by instantiating the abstract labeling functions with concrete ones, e.g. the rule labeling. So only the third step needs to be adapted when formalizing new labeling functions, as steps one and two are unaffected. The content of this section is part of the theory `Decreasing_Diagrams2.thy` in IsaFoR.

4.2.1 Local Peaks

As IsaFoR already supported Knuth-Bendix' criterion [98], it contained results for joinability of local peaks and the critical pair lemma. However, large parts of the existing formalization could not be reused directly as the established



(a) Parallel peak. (b) Function peak. (c) Variable peak.

Figure 10: Three kinds of local peaks.

results lacked information required for ensuring decreasingness. For instance, to obtain decreasingness for the rule labeling in case of a variable peak, the rewrite rules employed in the joining sequences are crucial, but the existing formalization only states that such a local peak is joinable. On the other hand, the existing notion of critical pairs from IsaFoR could be reused as the foundation for critical peaks. Since the computation of critical pairs requires a formalized unification algorithm, extending IsaFoR admitted focusing on the tasks related to decreasingness.

Local peaks can be characterized based on the positions of the diverging rewrite steps. Either the positions are parallel, called a *parallel peak*, or one position is above the other. In the latter situation we further distinguish whether the lower position is at a function position, called a *function peak*, or at/below a variable position of the other rule's left-hand side, called a *variable peak*. More precisely, for a local peak

$$t = s[r_1\sigma_1]_p \leftarrow s[\ell_1\sigma_1]_p = s = s[\ell_2\sigma_2]_q \rightarrow s[r_2\sigma_2]_q = u \tag{4.2}$$

there are three possibilities (modulo symmetry):

(a) $p \parallel q$ (parallel peak),

(b) $q \leqslant p$ and $p\backslash q \in \mathsf{Pos}_{\mathcal{F}}(\ell_2)$ (function peak),

(c) $q \leqslant p$ and $p\backslash q \notin \mathsf{Pos}_{\mathcal{F}}(\ell_2)$ (variable peak).

```
local_peaks R =
  {((s, r1, p1, σ1, True, t), (s, r2, p2, σ2, True, u)) | s t u r1 r2 p1 p2 σ1 σ2.
   (s, t) ∈ rstep_r_p_s R r1 p1 σ1 ∧ (s, u) ∈ rstep_r_p_s R r2 p2 σ2}

parallel_peak R p =
  (p ∈ local_peaks R ∧
   (let ((s, r1, p1, σ1, b, t), (s, r2, p2, σ2, d, u)) = p in p1 ⊥ p2))

function_peak R p =
  (p ∈ local_peaks R ∧
   (let ((s, r1, p1, σ1, b, t), (s, r2, p2, σ2, d, u)) = p in
    ∃r. p1 <#> r = p2 ∧ r ∈ poss (fst r1) ∧ is_Fun (fst r1 |_ r)))

variable_peak R p =
  (p ∈ local_peaks R ∧
   (let ((s, r1, p1, σ1, b, t), (s, r2, p2, σ2, d, u)) = p in
    ∃r. p1 <#> r = p2 ∧ ¬ (r ∈ poss (fst r1) ∧ is_Fun (fst r1 |_ r))))
```

Listing 6: Characterization of local peaks.

For the situation of a left-linear TRS these cases are visualized in Figure 10. It is easy to characterize parallel, function, and variable peaks in Isabelle, see Listing 6, but it requires tedious notation. For instance the ***local_peaks*** of a TRS $\mathcal{R}$ are represented as the set of all pairs $(s, r_1, p_1, \sigma_1, \mathsf{True}, t)$ and $(s, r_2, p_2, \sigma_2, \mathsf{True}, u)$, with rewrite steps $s \to^{p_1, r_1, \sigma_1} t$ and $s \to^{p_2, r_2, \sigma_2} u$. The rewrite steps are formalized via the existing IsaFoR notion of ***rstep_r_p_s***, which represents a step $s \to_{\mathcal{R}}^{p, \ell \to r, \sigma} t$ as $(s,\ t) \in$ ***rstep_r_p_s*** $\mathcal{R}\ (l,\ r)\ p\ \sigma$. Here True and False are used to recall the direction of a step, i.e., $s \to t$ and $t \leftarrow s$, respectively. This distinction is important for steps that are part of conversions, because conversions may mix forward and backward steps. In the definitions of ***function_peak*** and ***variable_peak*** the symbol <#> is used, which is the IsaFoR operation for concatenating positions.

As the definition of function and variable peaks is asymmetric the five cases of local peaks can be reduced to the above three by mirroring those peaks. Then local peaks can be characterized as in Listing 7. Next we elaborate on the three cases.

$p \in$ ***local_peaks*** $\mathcal{R} \Longrightarrow$
parallel_peak $\mathcal{R}$ $p \vee$ ***variable_peak*** $\mathcal{R}$ $p \vee$ ***variable_peak*** $\mathcal{R}$ (***snd*** p, ***fst*** p)
$\vee$ ***function_peak*** $\mathcal{R}$ $p \vee$ ***function_peak*** $\mathcal{R}$ (***snd*** p, ***fst*** p)

Listing 7: Cases of local peaks.

Case 1: Parallel Peaks Figure 10(a) shows the shape of a local peak where the steps take place at parallel positions. For a peak $t \,{}^{\pi_1}\!\leftarrow s \rightarrow^{\pi_2} u$ with $\pi_1 \parallel \pi_2$ we established that $t \rightarrow^{\pi_2} v \,{}^{\pi_1}\!\leftarrow u$, where opposing steps apply the same rule/substitution at the same position. The proof is straightforward and based on a decomposition of the terms into a context and the redex.

Case 2: Function Peaks In general joining function peaks may involve rules not present in the divergence, as indicated by the question mark in Figure 10(b). To reduce the burden of joining infinitely many function peaks to joining the, in case of a finite TRS finitely many, critical peaks, we formalized that every function peak is an instance of a critical peak.

Lemma 4.8. *Let* $t \,{}^{p,\ell_1\to r_1,\sigma_1}\!\leftarrow s \rightarrow^{q,\ell_2\to r_2,\sigma_2} u$ *with* $qq' = p$ *and* $q' \in \mathsf{Pos}_{\mathcal{F}}(\ell_2)$. *Then there are a critical peak* $\ell_2\mu[r_1\mu]_{q'} \leftarrow \ell_2\mu \rightarrow r_2\mu$, *a context* C, *and a substitution* τ *where* $s = C[\ell_2\mu\tau]$, $t = C[(\ell_2\mu[r_1\mu]_{q'})\tau]$, *and* $u = C[r_2\mu\tau]$. □

This fact is folklore, see e.g. [105, Lemma 2.7.12]. We remark that it was already present (multiple times) in IsaFoR, but concealed in larger proofs, e.g. the formalization of orthogonality [67], and never stated explicitly.

As we do not enforce that the variables of a rewrite rule's right-hand side are contained in its left-hand side, such rules are also included in the critical peak computation.

Case 3: Variable Peaks Variable overlaps, Figure 10(c), can again be joined by the rules involved in the diverging step.[2] We only consider the case that $\ell_2 \to r_2$ is left-linear, as our main result assumes left-linearity. More precisely, if q' is the unique position in $\mathsf{Pos}_{\mathcal{V}}(\ell_2)$ such that $qq' \leqslant p$, $x = \ell_2|_{q'}$, and $|r_2|_x = n$ then we have $t \rightarrow_{\ell_2\to r_2} v$, which is similar to the case for parallel peaks, as the redex $\ell_2\sigma$ becomes $\ell_2\tau$ but is not destroyed, and $u \rightarrow^n_{\ell_1\to r_1} v$. To obtain

[2] This includes rules having a variable as left-hand side.

this result we reason via parallel rewriting. The notions of parallel rewriting described so far do not keep track of, for example, the applied rules. Thus we use a different formulation of parallel steps, which record the information (position, rewrite rule, substitution) of each rewrite step, i.e., the rewrite relation is decorated with the contracted redex patterns.

Definition 4.9. For a TRS $\mathcal{R}$ the *parallel rewrite relation* $\mathrel{+\!\!\!\Rightarrow}$ is defined by the following inference rules.

$$\frac{}{x \overset{\varnothing}{\mathrel{+\!\!\!\Rightarrow}} x} \qquad \frac{\ell \to r \in \mathcal{R}}{l\sigma \overset{\{\langle \epsilon, \ell \to r, \sigma\rangle\}}{\mathrel{+\!\!\!\Rightarrow}} r\sigma} \qquad \frac{s_1 \overset{P_1}{\mathrel{+\!\!\!\Rightarrow}} t_1 \quad \cdots \quad s_n \overset{P_n}{\mathrel{+\!\!\!\Rightarrow}} t_n}{f(s_1,\ldots,s_n) \overset{(1P_1)\cup\cdots\cup(nP_n)}{\mathrel{+\!\!\!\Rightarrow}} f(t_1,\ldots,t_n)}$$

Here for a set of redex patterns $P = \{\pi_1, \ldots, \pi_m\}$ by iP we denote the set $\{i\pi_1, \ldots, i\pi_m\}$ with $i\pi = \langle ip, \ell \to r, \sigma\rangle$ for $\pi = \langle p, \ell \to r, \sigma\rangle$.

To use this parallel rewrite relation for closing variable peaks we formalized the following auxiliary facts.[3]

Lemma 4.10. *The following properties of the parallel rewrite relation hold:*

(a) *For all terms s we have $s \mathrel{+\!\!\!\Rightarrow}^{\varnothing} s$.*

(b) *If $s \mathrel{+\!\!\!\Rightarrow}^{\varnothing} t$ then $s = t$.*

(c) *If $s \mathrel{+\!\!\!\Rightarrow}^{P} t$ and $q \in \mathsf{Pos}(u)$ then $u[s]_q \mathrel{+\!\!\!\Rightarrow}^{qP} u[t]_q$.*

(d) *We have $s \to^{\pi} t$ if and only if $s \mathrel{+\!\!\!\Rightarrow}^{\{\pi\}} t$.*

(e) *If $\sigma(x) \to^{\pi} \tau(x)$ and $\sigma(y) = \tau(y)$ for all $y \in \mathcal{V}$ with $y \neq x$ then $t\sigma \mathrel{+\!\!\!\Rightarrow}^{P} t\tau$ with $\ell_{\pi'} \to r_{\pi'} = \ell_{\pi} \to r_{\pi}$ for all $\pi' \in P$.*

(f) *If $s \mathrel{+\!\!\!\Rightarrow}^{\{\pi\}\cup P} t$ then there is a term u with $s \mathrel{+\!\!\!\Rightarrow}^{\{\pi\}} u \mathrel{+\!\!\!\Rightarrow}^{P} t$.*

(g) *If $s \mathrel{+\!\!\!\Rightarrow}^{\{\pi_1,\ldots,\pi_n\}} t$ then $s \to^{\pi_1} \cdots \to^{\pi_n} t$.* □

[3] In a typical paper proof, these facts would be considered as self-evident. However, Isabelle is not so easily convinced, so a proof is required. So rather than being an interesting result, Lemma 4.10 serves to illustrate the level of detail that formalizations often require.

$$\frac{}{(s,\ [])\ \in\ \textbf{\textit{conv}}\ \mathcal{R}}$$

$$\frac{(s,\ t)\ \in\ \textbf{\textit{rstep\_r\_p\_s}}\ \mathcal{R}\ r\ p\ \sigma \qquad (t,\ ts)\ \in\ \textbf{\textit{conv}}\ \mathcal{R}}{(s,\ (s,\ r,\ p,\ \sigma,\ \mathsf{True},\ t)\ \#\ ts)\ \in\ \textbf{\textit{conv}}\ \mathcal{R}}$$

$$\frac{(s,\ t)\ \in\ \textbf{\textit{rstep\_r\_p\_s}}\ \mathcal{R}\ r\ p\ \sigma \qquad (s,\ ts)\ \in\ \textbf{\textit{conv}}\ \mathcal{R}}{(t,\ (s,\ r,\ p,\ \sigma,\ \mathsf{False},\ t)\ \#\ ts)\ \in\ \textbf{\textit{conv}}\ \mathcal{R}}$$

Listing 8: Conversions.

In principle the statements of Lemma 4.10 follow from the definitions using straightforward induction proofs, building upon each other in the order they are listed. However, the additional bookkeeping, required to correctly propagate the information attached to the rewrite relation, makes them considerably more involved than for the previous, agnostic notion of parallel rewriting.

Now for reasoning about variable peaks as in Figure 10(c) we decompose $u = s[r_2\sigma]_q$ and $v = s[r_2\tau]_q$ where $\sigma(y) = \tau(y)$ for all $y \in \mathcal{V} \setminus \{x\}$ and $\sigma(x) \to_{p\setminus qq',\ell_1\to r_1} \tau(x)$. From the latter by item (e) we obtain $r_2\sigma \mathrel{+\!\!\!\twoheadrightarrow^P} r_2\tau$, where all redex patterns in P use $\ell_1 \to r_1$. Then by item (c) we get $s[r_2\sigma]_q \mathrel{+\!\!\!\twoheadrightarrow^{qP}} s[r_2\tau]_q$ and finally $s[r_2\sigma]_q \to^n_{\ell_1\to r_1} s[r_2\tau]_q$ with $n = |qP| = |P|$ by item (g).

4.2.2 Local Decreasingness

This section presents a confluence result, see Corollary 4.16, based on decreasingness of the critical peaks. Abstract conditions, via the key notion of a labeling, will ensure that parallel peaks and variable peaks are decreasing. Furthermore, these conditions imply that decreasingness of the critical peaks implies decreasingness of the function peaks. For establishing (extended) local decreasingness, a label must be attached to rewrite steps. To facilitate the computation of labels, in the formalization conversions provide information about the intermediate terms, applied rules, etc. In Listing 8 the definition of conversions as an inductive set is provided. The first case states that the empty conversion starting from a term s is a conversion. The second case states that if $s \to_{\mathcal{R}} t$, using rule r at position p with substitution σ, and there is a conversion starting from t, where the list ts collects the details on the conversion, then

there also is a conversion starting from s and the details for this conversion are collected in the list where the tuple $(s, r, p, \sigma, \mathsf{True}, t)$ is added to the list ts. While the first two cases of Listing 8 amount to an inductive definition of rewrite sequences, together with the third case (which considers a step $t \leftarrow s$ and a conversion starting from s), conversions are obtained. Here True and False are used to recall the direction of a step, as in Listing 6.

Furthermore, labels are computed by a *labeling function*, having local information about the rewrite step, such as source and target term, applied rewrite rule, position, and substitution, it is expected to label. For reasons of readability we continue using mathematical notation (e.g., $\to^*$ for sequences and $\leftrightarrow^*$ for conversions, etc.) with all information implicit but remark that the formalization works on sequences and conversions with explicit information as in Listing 8.

Definition 4.11. A *labeling* is a function l from rewrite steps to a set of labels such that for all contexts C and substitutions σ the following properties are satisfied:

- It is compatible with the strict order on labels: if $l(s \to^{\pi_1} t) > l(u \to^{\pi_2} v)$ then $l(C[s\sigma] \to^{C[\pi_1\sigma]} C[t\sigma]) > l(C[u\sigma] \to^{C[\pi_2\sigma]} C[v\sigma])$.
- It is compatible with the weak order on labels: if $l(s \to^{\pi_1} t) \geqslant l(u \to^{\pi_2} v)$ then $l(C[s\sigma] \to^{C[\pi_1\sigma]} C[t\sigma]) \geqslant l(C[u\sigma] \to^{C[\pi_2\sigma]} C[v\sigma])$.
- It is compatible with directions: $l(s \to^{\pi} t) = l(t \,{}^{\pi}\!\leftarrow s)$.

Here $C[\pi\sigma]$ denotes $\langle qp, \ell \to r, \tau\sigma\rangle$ for $\pi = \langle p, \ell \to r, \tau\rangle$ and $C|_q = \Box$.

As the co-domain of a labeling is a set of labels, a labeling according to Definition 4.11 maps a single rewrite step to a single label, which is different from how (some) ARSs are labeled in Section 4.1. The rule labeling is a labeling in our sense.

In the presence of a labeling, conversions can be labeled at any time. This avoids lifting many notions (such as rewrite steps, local peaks, rewrite sequences, etc.) and results from rewriting to labeled rewriting.

In the next definition a labeling is applied to conversions via the equations

- $l(t \leftrightarrow^0 t) = \varnothing$,
- $l(s \to^{\pi} t \leftrightarrow^* u) = \{l(s \to^{\pi} t)\} \cup l(t \leftrightarrow^* u)$, and

- $l(s \,{}^{\pi}\!\leftarrow t \leftrightarrow^* u) = \{l(s \,{}^{\pi}\!\leftarrow t)\} \cup l(t \leftrightarrow^* u)$.

Definition 4.12. A local peak $t \,{}^{\pi_1}\!\leftarrow s \rightarrow^{\pi_2} u$ is *extended locally decreasing* (for a labeling l and orders $>$ and $\geqslant$) if there is a local diagram, i.e., $t \leftrightarrow^* t' \rightarrow^{=} t'' \leftrightarrow^* u'' \,{}^{=}\!\leftarrow u' \leftrightarrow^* u$ such that its labels are extended locally decreasing, i.e.,

- $l(t \leftrightarrow^* t') \subseteq \vee l(t \,{}^{\pi_1}\!\leftarrow s)$,
- $l(t' \rightarrow^{=} t'') \subseteq \vee\!\!/ l(s \rightarrow^{\pi_2} u)$,
- $l(t'' \leftrightarrow^* u'') \subseteq \vee l(t \,{}^{\pi_1}\!\leftarrow s \rightarrow^{\pi_2} u)$,
- $l(u'' \,{}^{=}\!\leftarrow u') \subseteq \vee\!\!/ l(t \,{}^{\pi_1}\!\leftarrow s)$, and
- $l(u' \leftrightarrow^* u) \subseteq \vee l(s \rightarrow^{\pi_2} u)$.

Following [113], we separate (local) diagrams (where rewriting is involved) from decreasingness (where only the labels are involved). In contrast to the valley version of decreasing diagrams, as employed in [68], where a sequence $\rightarrow^*$ can be decomposed into $\rightarrow^*_{\vee\alpha} \cdot \rightarrow^{=}_{\vee\!\!/\beta} \cdot \rightarrow^*_{\vee\alpha\beta}$ based on the sequence of labels, this is no longer possible for the conversion version (as there is no guarantee that the step $\rightarrow^{=}_{\vee\!\!/\beta}$ is oriented correctly). Hence we made the decomposition explicit for the conversion version.

The corresponding predicate in IsaFoR is given in Listing 9 where extended local decreasingness (***eldc***) of a local peak p is expressed via the existence of conversions cl_1, cl_2, cr_1, cr_2 and possibly empty steps sl and sr that close the divergence caused by the local peak p in the shape of a local diagram (***ldc_trs***).[4] Here ***get_target*** gets the target of a rewrite step and ***first*** and ***last*** get the first and last element of a conversion or rewrite sequence, respectively. Moreover the labels of the underlying conversions are required to be extended locally decreasing (***eld_conv***, ***ELD_1***). Here ***ds*** $(\preccurlyeq)$ S is the notation for $\curlyvee S$ and r is the pair of relations $(>, \geqslant)$.

Then a function peak is extended locally decreasing if the critical peaks are.

Lemma 4.13. *Let l be a labeling and let all critical peaks of a TRS $\mathcal{R}$ be extended locally decreasing for l. Then every function peak of $\mathcal{R}$ is extended locally decreasing for l.*

[4] The conversions cl_2 and cr_2 could be merged into a single conversion. However, in proofs the symmetric version permits to reason about one side and obtain the other side for free.

```
eldc R l r p =
  (∃ cl1 sl cl2 cr1 sr cr2. ldc_trs R p cl1 sl cl2 cr1 sr cr2 ∧
     eld_conv l r p cl1 sl cl2 cr1 sr cr2)

ldc_trs R p cl1 sl cl2 cr1 sr cr2 =
  (p ∈ local_peaks R ∧ cl1 ∈ conv R ∧ sl ∈ seq R ∧ cl2 ∈ conv R ∧
   cr1 ∈ conv R ∧ sr ∈ seq R ∧ cr2 ∈ conv R ∧
   get_target (fst p) = first cl1 ∧ last cl1 = first sl ∧
   last sl = first cl2 ∧ get_target (snd p) = first cr1 ∧
   last cr1 = first sr ∧ last sr = first cr2 ∧ last cl2 = last cr2)

eld_conv l r p cl1 sl cl2 cr1 sr cr2 =
  (ELD_1 r (l (fst p)) (l (snd p)) (map l (snd cl1)) (map l (snd sl))
    (map l (snd cl2)) ∧
   ELD_1 r (l (snd p)) (l (fst p)) (map l (snd cr1)) (map l (snd sr))
    (map l (snd cr2)))

ELD_1 r β α σ1 σ2 σ3 =
  (set σ1 ⊆ ds (fst r) {β} ∧ length σ2 ≤ 1 ∧ set σ2 ⊆ ds (snd r) {α} ∧
   set σ3 ⊆ ds (fst r) {α, β})
```

Listing 9: Extended local decreasingness.

Proof. As every function peak is an instance of a critical peak (see Lemma 4.8), the result follows from l being a labeling (Definition 4.11). □

The notion of compatibility (between a TRS and a labeling) admits a finite characterization of extended local decreasingness.

Definition 4.14. Let l be a labeling. We call l *compatible* with a TRS $\mathcal{R}$ if all parallel peaks and all variable peaks of $\mathcal{R}$ are extended locally decreasing for l.

The key lemma establishes that if l is compatible with a TRS, then all local peaks are extended locally decreasing.

Lemma 4.15. *Let l be a labeling that is compatible with a TRS $\mathcal{R}$. If the critical peaks of $\mathcal{R}$ are extended locally decreasing for l, then all local peaks of $\mathcal{R}$ are extended locally decreasing for l.*

Proof. The cases of variable and parallel peaks are taken care of by compatibility. The case of function peaks follows from the assumption in connection with Lemma 4.13. The symmetric cases for function and variable peaks can be resolved by mirroring the local diagrams. □

Representing a TRS $\mathcal{R}$ over the signature $\mathcal{F}$ and variables $\mathcal{V}$ as the ARS over objects $\mathcal{T}(\mathcal{F}, \mathcal{V})$ and relations $\to_\alpha = \{(s,t) \mid s \to^\pi t \text{ and } l(s \to^\pi t) = \alpha\}$ for some labeling l, Lemma 4.2 immediately applies to TRSs. To this end, extended local decreasingness formulated via explicit rewrite sequences with labeling functions is mapped to extended local decreasingness on families of abstract rewrite relations. Finally, we are ready to formalize a confluence result for first-order term rewriting.

Corollary 4.16. *Let l be a labeling compatible with a TRS $\mathcal{R}$, and let $>$ and $\geqslant$ be a well-founded order and a compatible preorder, respectively. If the critical peaks of $\mathcal{R}$ are extended locally decreasing for l and $>$ and $\geqslant$ then $\mathcal{R}$ is confluent.* □

Concrete confluence criteria are then obtained as instances of the above result by instantiating l. For example, Theorem 4.6 is obtained by showing that its relative termination assumption in combination with the rule labeling yields a compatible labeling for left-linear TRSs.

4.3 Checkable Confluence Proofs

In this section we instantiate Corollary 4.16 to obtain concrete confluence results, first for linear TRSs in Section 4.3.1, and then for left-linear TRSs in Section 4.3.2. Afterwards we discuss the design of the certificates, checkable by CeTA, in Section 4.3.3. The formalization corresponding to the first two subsections is part of `Decreasing_Diagrams2.thy` in IsaFoR, whereas the executable check-functions from Section 4.3.3 can be found in the theory `Rule_Labeling_Impl.thy`.

4.3.1 Linear Term Rewrite Systems

The rule labeling admits a confluence criterion for linear TRSs based on the formalization established so far.

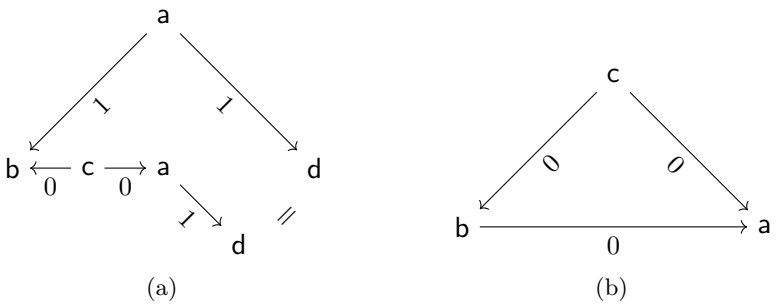


Figure 11: Decreasingness of critical peaks in Example 4.18.

Lemma 4.17. *Let $\mathcal{R}$ be a linear TRS.*

(a) *The rule labeling is a labeling.*

(b) *Parallel peaks are extended locally decreasing for the rule labeling.*

(c) *Variable peaks of $\mathcal{R}$ are extended locally decreasing for the rule labeling.*

(d) *The rule labeling is compatible with $\mathcal{R}$.*

(e) *If all critical peaks of $\mathcal{R}$ are extended locally decreasing for the rule labeling, then $\mathcal{R}$ is confluent.*

Proof. Item (a) follows from Definitions 4.11 and 4.5. For items (b) and (c) we employ the analysis of parallel and variable peaks from Section 4.2.1. Item (d) is then a consequence of items (b) and (c). Finally, item (e) amounts to an application of Corollary 4.16. □

We demonstrate the rule labeling on a simple example.

Example 4.18 (Cop #761)**.** Consider the TRS consisting of the following rules, where subscripts indicate labels for the rule labeling:

$$\mathsf{a} \to_1 \mathsf{b} \qquad \mathsf{a} \to_1 \mathsf{d} \qquad \mathsf{b} \to_0 \mathsf{a} \qquad \mathsf{c} \to_0 \mathsf{a} \qquad \mathsf{c} \to_0 \mathsf{b}$$

The critical peaks are decreasing for the rule labeling as depicted in Figure 11.

4.3.2 Left-linear Term Rewrite Systems

That a locally confluent terminating left-linear TRS is confluent can be established in the flavor of Lemma 4.17. The restriction to left-linearity arises from the lack of considering non-left-linear variable peaks in Section 4.2.1. As the analysis of such a peak would not give further insights we pursue another aim in this section, i.e., the mechanized proof of Theorem 4.6.

It is well known that the rule labeling l^i is in general not compatible with left-linear TRSs [42]. Thus, to obtain extended local decreasingness for variable peaks, in Theorem 4.6 the additional relative termination assumption is exploited. To this end, we use the source labeling. Note that for the rule labeling alone extended local decreasingness directly implies local decreasingness, as $\geqslant_{\mathbb{N}}$ is the reflexive closure of $>_{\mathbb{N}}$.

Definition 4.19. The *source labeling* maps a rewrite step to its source, i.e., $l^{\mathsf{src}}(s \to^{\pi} t) = s$. Labels due to the source labeling are compared using the orders $\to^{+}_{\mathcal{R}_{\mathsf{d}}/\mathcal{R}_{\mathsf{nd}}}$ and $\to^{*}_{\mathcal{R}}$.

The relative termination assumption of Theorem 4.6 makes all variable peaks of a left-linear TRS extended locally decreasing for the source labeling. These variable peaks might not be extended locally decreasing for the rule labeling, as the step $u \twoheadrightarrow_{\{\ell_1 \to r_1\}} v$ in Figure 10(c) yields $u \to^{n} v$ for n possibly larger than one. Hence we introduce weaker versions of decreasingness and compatibility.

Definition 4.20. A diagram of the shape $t \,{}_{\alpha}\!\leftarrow s \to^{\ell_2 \to r_2}_{\beta} u$, $t \to_{\diagdown\!/\beta} v \,{}^{n}_{\diagdown\!/\alpha}\!\leftarrow u$ is called *weakly extended locally decreasing* if $n \leqslant 1$ whenever r_2 is linear, see Figure 12. We call a labeling l *weakly compatible* with a TRS $\mathcal{R}$ if parallel and variable peaks are weakly extended locally decreasing for l.

Following [115], the aim is to establish that the lexicographic combination of a compatible labeling with a weakly compatible labeling (e.g., $l^{\mathsf{src}} \times l^i$) is compatible with a left-linear TRS. While weak extended local decreasingness could also be defined in the spirit of extended local decreasingness (with a more complicated join sequence or involving conversions), the chosen formulation suffices to establish the result, eases the definition, and simplifies proofs. As this notion is only used to reason about parallel and variable peaks, which need not be processed by automatic tools, the increased generality would not benefit automation.

Based on the peak analysis of Section 4.2.1, the following results are formalized (such properties must be proved for each labeling function):

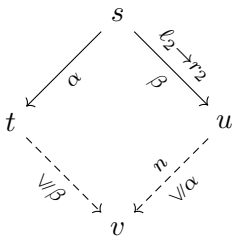


Figure 12: Weakly extended locally decreasing diagram.

Lemma 4.21. *Let $\mathcal{R}$ be a left-linear TRS.*

(a) *Parallel peaks are weakly extended locally decreasing for the rule labeling.*

(b) *Variable peaks of $\mathcal{R}$ are weakly extended locally decreasing for the rule labeling.*

(c) *The rule labeling is weakly compatible with $\mathcal{R}$.*

Proof. Items (a) and (b) are established by labeling the rewrite steps in the corresponding diagrams based on the characterizations of parallel and variable peaks of Figures 10(a) and 10(c), respectively. Item (c) follows from (a) and (b) together with Lemma 4.17(a). □

Similar results are formalized for the source labeling.

Lemma 4.22. *Let $\mathcal{R}$ be a left-linear TRS whose duplicating rules terminate relative to the other rules.*

(a) *The source labeling is a labeling.*

(b) *Parallel peaks are extended locally decreasing for the source labeling.*

(c) *Variable peaks of $\mathcal{R}$ are extended locally decreasing for the source labeling.*

(d) *The source labeling is compatible with $\mathcal{R}$.*

Proof. Closure of rewriting under contexts and substitutions immediately yields item (a). Items (b) and (c) are established along the lines of the proof of Lemma 4.21. Finally, item (d) follows from items (b) and (c) in combination with Definition 4.14. □

Using this lemma, we proved the following results for the lexicographic combination of the source labeling with another labeling.

Lemma 4.23. *Let $\mathcal{R}$ be a left-linear TRS whose duplicating rules terminate relative to the other rules and l be a labeling weakly compatible with $\mathcal{R}$. Then $l^{\mathsf{src}} \times l$ is a labeling compatible with $\mathcal{R}$.* □

For reasons of readability we have left the orders $>$ and $\geqslant$ that are required for (weak) compatibility implicit and just mention that the lexicographic extension as detailed in [115] preserves the required properties. Finally, we prove Theorem 4.6.

Proof of Theorem 4.6. From Lemma 4.21(c) in combination with Lemma 4.23 we obtain that $l^{\mathsf{src}} \times l^i$ is a labeling compatible with a left-linear TRS, provided the relative termination assumption is satisfied. By assumption, the critical peaks are extended locally decreasing for the rule labeling l^i. As along a rewrite sequence labels with respect to l^{src} never increase, the critical peaks are extended locally decreasing for $l^{\mathsf{src}} \times l^i$. We conclude the proof by an application of Corollary 4.16. □

Actually a stronger result than Theorem 4.6 has been established, as $l^{\mathsf{src}} \times l^i$ might show more critical peaks decreasing than l^i alone, like in the following example.

Example 4.24 (Cop #763)**.** Consider the TRS $\mathcal{R}$ consisting of the single rule

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

That $\mathcal{R}$ is terminating can easily be shown by modern termination provers (for instance, applying the dependency pair transformation and a tcap-based dependency graph approximation suffices). Also note that $\mathcal{R}_{\mathsf{d}} = \mathcal{R}$, $\mathcal{R}_{\mathsf{nd}} = \varnothing$, and $\to^+_{\mathcal{R}_{\mathsf{d}}/\mathcal{R}_{\mathsf{nd}}} = \to^+_{\mathcal{R}}$. There is one critical peak:

$$\mathsf{f}(\mathsf{f}(\mathsf{g}(\mathsf{f}(x), \mathsf{f}(x)))) \leftarrow \mathsf{f}(\mathsf{f}(\mathsf{f}(x))) \to \mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{f}(x)), \mathsf{f}(\mathsf{f}(x))))$$

Apart from reordering the steps from the right, the only valley for this peak is:

$$\begin{aligned}\mathsf{f}(\mathsf{f}(\mathsf{g}(\mathsf{f}(x), \mathsf{f}(x)))) &\to \mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{g}(\mathsf{f}(x), \mathsf{f}(x))), \mathsf{f}(\mathsf{g}(\mathsf{f}(x), \mathsf{f}(x)))))\\ &\leftarrow \mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{g}(\mathsf{f}(x), \mathsf{f}(x))), \mathsf{f}(\mathsf{f}(x)))) \leftarrow \mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{f}(x)), \mathsf{f}(\mathsf{f}(x))))\end{aligned}$$

Since there is only one rule in $\mathcal{R}$ all steps get the same label when applying the rule labeling. Consequently, labeling this diagram decreasingly using the rule labeling cannot succeed, because the sequence from the right consists of two steps. However, using the source labeling, and hence also $l^{\mathsf{src}} \times l^i$, the diagram is clearly decreasing, because all terms in the valley are reachable from $\mathsf{f}(\mathsf{f}(\mathsf{f}(x)))$ using $\to^+_{\mathcal{R}_\mathsf{d}/\mathcal{R}_\mathsf{nd}}$.

Regarding a variant of Theorem 4.6 based on the conversion version of decreasing diagrams the statement "*As along a rewrite sequence labels with respect to* l^{src} *never increase, ...*" in the above proof is no longer appropriate, as rewrite sequences are replaced by conversions. Consequently, in contrast to the valley version, the source labeling might destroy decreasingness, as shown by the next example.

Example 4.25 (Example 4.18 continued)**.** Although the critical diagram in Figure 11(a) is decreasing for l^i it is not decreasing for $l^{\mathsf{src}} \times l^i$ as the label of the step $\mathsf{a} \to_{(\mathsf{a},1)} \mathsf{b}$ is not larger than the label of the step $\mathsf{b}\ {}_{(\mathsf{c},0)}\!\leftarrow \mathsf{c}$. The problem is that $\mathsf{a} \to^* \mathsf{c}$ does not hold.

To forbid the situation highlighted in Example 4.25, the property that the labels of the conversions are smaller or equal to the source of the local peak must be ensured.

Definition 4.26. A diagram of the shape $t_1 \leftarrow s \to t_n$, $t_1 \leftrightarrow^* t_2 \leftrightarrow^* \cdots \leftrightarrow^* t_n$ has the *fan property* if $s \to^* t_i$ for $1 \leqslant i \leqslant n$.

The fan property is sketched in Figure 13, where the solid arcs indicate the diagram and the dashed arcs the additional conditions.[5] The fan property is related to, but slightly different from source decreasingness [44], which demands that, using the source labeling, every peak $\leftarrow s \to$ is connected by a conversion in which all labels are smaller than s. Our formalization covers the following result, which is new in theory and IsaFoR:

Theorem 4.27 (Conversion version of decreasing diagrams)**.** *A left-linear TRS is confluent if its duplicating rules terminate relative to its other rules and all its critical peaks have local diagrams that both have the fan property and are decreasing for the rule labeling.*

[5] Note that $s \to^* t_i$ implies $s \to^* t_{i-1}$ and $s \to^* t_{i+1}$ if $t_{i-1} \leftarrow t_i \to t_{i+1}$ and so the indicated reductions suffice.

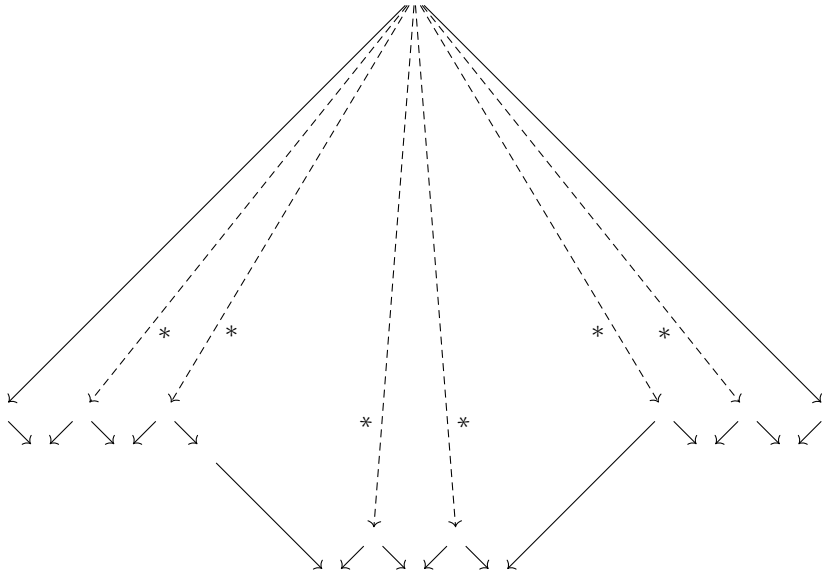


Figure 13: The fan property.

Proof. That $l^{\mathsf{src}} \times l^i$ is a labeling, is obtained as in the proof of Theorem 4.6. The fan property ensures that a critical peak that is decreasing for the rule labeling l^i is also decreasing for $l^{\mathsf{src}} \times l^i$. We conclude by an application of Corollary 4.16. □

The following example demonstrates that the fan property is necessary for Theorem 4.27 to be correct. Note that the TRS in Example 4.25 is confluent and can hence only motivate the fan property but cannot show its indispensability.

Example 4.28 (Cop #762)**.** Consider the TRS $\mathcal{R}$ consisting of the following rules, where subscripts indicate labels for the rule labeling:

$$\mathsf{a} \to_2 \mathsf{b} \quad \mathsf{f}(\mathsf{a},\mathsf{b}) \to_1 \mathsf{f}(\mathsf{a},\mathsf{a}) \quad \mathsf{f}(\mathsf{b},\mathsf{a}) \to_1 \mathsf{f}(\mathsf{a},\mathsf{a}) \quad \mathsf{f}(\mathsf{a},\mathsf{a}) \to_1 \mathsf{c} \quad \mathsf{g}(x) \to_0 \mathsf{f}(x,x)$$

Then $\mathcal{R}_{\mathsf{d}}/\mathcal{R}_{\mathsf{nd}}$ is easily seen to be terminating, for example using the lexicographic path order with a quasi-precedence that equates a and b and has $\mathsf{g} > \mathsf{f} > \mathsf{c}$. There are four critical peaks, all of which are decreasing with

respect to the given rule labeling:

$$\mathsf{f}(\mathsf{a},\mathsf{b}) \;{}_2{\leftarrow}\; \mathsf{f}(\mathsf{a},\mathsf{a}) \to_1 \mathsf{c} \qquad \mathsf{f}(\mathsf{a},\mathsf{b}) \to_1 \mathsf{f}(\mathsf{a},\mathsf{a}) \to_1 \mathsf{c}$$
$$\mathsf{f}(\mathsf{b},\mathsf{a}) \;{}_2{\leftarrow}\; \mathsf{f}(\mathsf{a},\mathsf{a}) \to_1 \mathsf{c} \qquad \mathsf{f}(\mathsf{b},\mathsf{a}) \to_1 \mathsf{f}(\mathsf{a},\mathsf{a}) \to_1 \mathsf{c}$$
$$\mathsf{f}(\mathsf{b},\mathsf{b}) \;{}_2{\leftarrow}\; \mathsf{f}(\mathsf{a},\mathsf{b}) \to_1 \mathsf{f}(\mathsf{a},\mathsf{a}) \qquad \mathsf{f}(\mathsf{b},\mathsf{b}) \;{}_0{\leftarrow}\; \mathsf{g}(\mathsf{b}) \;{}_2{\leftarrow}\; \mathsf{g}(\mathsf{a}) \to_0 \mathsf{f}(\mathsf{a},\mathsf{a})$$
$$\mathsf{f}(\mathsf{b},\mathsf{b}) \;{}_2{\leftarrow}\; \mathsf{f}(\mathsf{b},\mathsf{a}) \to_1 \mathsf{f}(\mathsf{a},\mathsf{a}) \qquad \mathsf{f}(\mathsf{b},\mathsf{b}) \;{}_0{\leftarrow}\; \mathsf{g}(\mathsf{b}) \;{}_2{\leftarrow}\; \mathsf{g}(\mathsf{a}) \to_0 \mathsf{f}(\mathsf{a},\mathsf{a})$$

Nevertheless, $\mathcal{R}$ is not confluent: $\mathsf{f}(\mathsf{b},\mathsf{b}) \leftarrow \mathsf{f}(\mathsf{a},\mathsf{b}) \leftarrow \mathsf{f}(\mathsf{a},\mathsf{a}) \to \mathsf{c}$ is a conversion between two distinct normal forms. Note that the local diagrams for the final two critical peaks violate the fan property: $\mathsf{g}(\mathsf{a})$ is neither reachable from $\mathsf{f}(\mathsf{a},\mathsf{b})$ nor from $\mathsf{f}(\mathsf{b},\mathsf{a})$.

Finally, we remark that in the formalization Theorem 4.6 is obtained by instantiating Theorem 4.27. To this end, we formalized that the fan property holds vacuously whenever the local diagram is a valley. The direct proof of Theorem 4.6 on page 74 does not have a correspondence in the formalization but it already conveys the proof idea of the more complex proof of Theorem 4.27.

4.3.3 Certificates

Next we discuss the design of the certificates for confluence proofs via Theorem 4.27, i.e., how they are represented in CPF, and the executable checker to verify them. A minimal certificate could just claim that the considered rewrite system can be shown decreasing via the rule labeling. However, this is undecidable, even for locally confluent systems [42]. Hence the certificate contains the following entries: the TRS $\mathcal{R}$, the index function i, (candidates for) the joining conversions for each critical peak, an upper bound on the number of rewrite steps required to check the fan property, and, in case $\mathcal{R}$ is not right-linear, a relative termination proof for $\mathcal{R}_{\mathsf{d}}/\mathcal{R}_{\mathsf{nd}}$. The labels in the joining conversions are not required in the certificate, since CeTA has to check, i.e., compute them anyway. The same holds for the critical peaks. Note that the (complex) reasoning required for parallel and variable peaks does not pollute the certificates. The outline of a certificate for a confluence proof according to Theorem 4.27 is shown in Figure 14.[6] Besides elements for struc-

[6] In Figure 14 some boilerplate nodes and details are omitted—a full certificate in CPF for Example 4.18 is available at `http://cl-informatik.uibk.ac.at/experiments/2016/rule_labeling/rule_labeling_conv.proof.xml`.

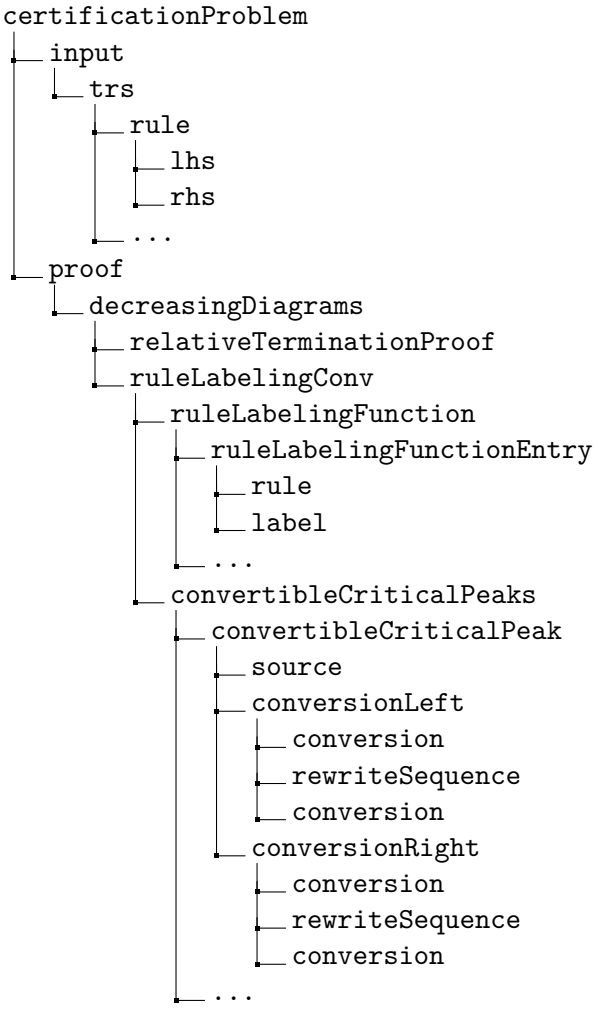


Figure 14: Structure of a Rule Labeling Certificate in CPF.

turing the certificate (`decreasingDiagrams`, `ruleLabelingConv`), additions to CPF, for representing proofs via Theorem 4.27, are the elements to represent the rule labeling (`ruleLabelingFunction`) and the joining conversions `convertibleCriticalPeaks`. Conversions and rewrite sequences themselves were already representable in CPF (`conversion`, `rewriteSequence`) and easy

to reuse. To check such a certificate CeTA performs the following steps:

(a) Parse the certificate. To parse the CPF elements that were newly introduced we extended CeTA's parser accordingly.

(b) Check the proof for relative termination. Luckily, CeTA already supports a wide range of relative termination techniques, so that here we just needed to make use of existing machinery.

(c) Compute all critical peaks of the rewrite system specified in the certificate.

(d) For each computed critical peak, find and check a decreasing joining conversion given in the certificate. To check decreasingness (see Listing 9) we require the decomposition of the joining conversions to be explicit in the certificate. As the confluence tools that generate certificates might use different renamings than CeTA when computing critical pairs, the conversions given in the certificate are subject to a variable renaming. Thus, after computing the critical peaks, CeTA has to consider the joining conversions modulo renaming of variables. Then checking that they form a local diagram and that the labels are extended locally decreasing is straightforward.

(e) Check the fan property. The certificate contains an upper bound on the number of rewrite steps required to reach the terms in the conversions from the source of the peak. This ensures termination of CeTA when checking the existence of suitable rewrite sequences.

CeTA also supports checking decreasingness using the valley version of decreasing diagrams, i.e., certifying applications of Theorem 4.6. In that case splitting the joining sequences in the certificate is not required: for every critical peak just two rewrite sequences need to be provided. CeTA can automatically find a split if one exists: given two natural numbers α and β and a sequence σ of natural numbers, is there a split $\sigma = \sigma_1\sigma_2\sigma_3$ such that $\sigma_1 \subseteq \vee\alpha$, $\sigma_2 \subseteq \vee\!\!/\beta$ with length of σ_2 at most one, and $\sigma_3 \subseteq \vee\alpha\beta$? The checker employs a simple, greedy approach. That is, we pick the maximal prefix of σ with labels smaller α as σ_1. If the next label is less or equal to β we take it as σ_2 and otherwise we take the empty sequence for σ_2. Finally, the remainder of the sequence is σ_3. A straightforward case analysis shows that this approach is complete, i.e., otherwise no such split exists.

4.4 Assessment

First we detail why Theorem 4.6 is an adequate candidate for formalization and certification. On the one hand, regarding the aspect of automation, it is easily implementable as the relative termination requirement can be outsourced to external relative termination provers and the rule labeling heuristic has already been implemented successfully [2, 41]. Furthermore, it is a powerful criterion as demonstrated by the experimental evaluation in Section 8.3. On the other hand, regarding the aspect of formalization, it is challenging because it involves the combination of different labeling functions (in the sense of [115]). Hence, in our formalization Theorem 4.6 is not established directly, but obtained as a corollary of more general results. In particular Lemma 4.23 is based on a more general result, which allows different labeling functions to be combined lexicographically. This paves the way for reusing the formalization described here when tackling the remaining criteria in [115], which are based on more flexible combinations of labeling functions, and use labelings besides the source labeling, Lemma 4.22, and the rule labeling, Lemma 4.17. For example, labels can also be defined based on the path to a rewrite step, or the redex that is being contracted. In order to certify the corresponding proofs, we will also have to extend the CPF format with encodings of those labelings.

The required characterization of closing local peaks, see Figure 10, provides full information about the rewrite steps involved in the joining sequences. As this characterization is the basis for many confluence criteria—not necessarily relying on decreasing diagrams—this result aids future certification efforts. We anticipate that the key result for closing variable peaks for the left-linear case, see Section 4.2.1, does not rely on the annotated version of parallel rewriting, but as [115] also supports labelings based on parallel rewriting, the developed machinery should be useful for targeting further confluence results from. Needless to say, parallel rewriting is handy on its own. The formalization described in this article covers a significant amount of the results presented in [115]. As explained, additional concepts, e.g. the annotated version of parallel rewriting were formalized with the remaining criteria in mind. However, for some results that are not covered yet, e.g. persistence [5, 25], we anticipate that even formalizing the preliminaries requires significant effort.

Next we discuss the usefulness of existing formalizations for this work. The existing machinery of IsaFoR provided invaluable support. We regard our efforts to establish an annotated version of parallel rewriting not as a shortcoming of

IsaFoR, but as a useful extension to it. On the contrary, we could employ many results from IsaFoR without further ado, e.g., completeness of the unification algorithm to compute critical peaks, plain rewriting to connect parallel steps with single steps, and the support for relative termination. That Lemma 4.8 occurred several times in IsaFoR can be traced to textbook proofs, e.g. [11], where this result is not made explicit either. Instead it is established in the scope of a larger proof of the critical pair lemma. Still, in later proofs the result is used as if it would have been established explicitly. In IsaFoR these proofs have been duplicated, but as formalization papers typically come with code refactoring these deficiencies have been fixed. Ultimately our aim in the formalization was to follow paper proofs as closely as possible. The benefit of this choice is that this way, shortcomings in existing proofs can be identified and eradicated. As our formalization covers and combines results from various sources, the notions used in the papers had to be connected. As already mentioned, while different notations are typically identified in paper proofs, in the formalization this step has to be made explicit. To avoid this drawback in the future our recommendation is to strive for more standard notation, also in paper proofs.

Finally, differences to [115] are addressed. The concepts of an L-labeling and an LL-labeling from [115] have been generalized to the notion of a labeling *compatible* with a TRS, while weak-LL-labelings are represented via *weakly compatible* labelings here.[7] This admits the formulation of the abstract conditions such that a labeling ensures confluence (see Corollary 4.16) independent from the TRS being (left-)linear. Furthermore we present a generalization of Theorem 4.6 to the conversion version of decreasing diagrams, namely Theorem 4.27.

4.5 Summary

In this chapter we presented the formalization of a result establishing confluence of left-linear term rewrite systems based on relative termination and the rule labeling. While our formalization admits stronger results in order to prepare for further results from [115], we targeted Theorem 4.6, whose statement,

[7] The definitions of L-labelings and LL-labelings spell out the shape of the standard joining valley for a variable peak for linear and left-linear TRSs, respectively, and then impose restrictions on the occurring labels that ensure compatibility.

in contrast to its proof, does not require the complex interplay of relative termination and the rule labeling. Hence this criterion is easily implementable for automated confluence tools, admitting the use of external termination provers. Our formalization subsumes the original criterion for the rule labeling, see Lemma 4.17(e), which is applicable to linear systems only. Dealing with non-right-linear systems required an analysis of non-right-linear variable peaks, and of the interplay with the relative termination condition. Furthermore, whereas plain rule labeling can be proved correct by decreasing diagrams, the involvement of the source labeling means that extended decreasing diagrams are required. Hence the proof of Theorem 4.6 is significantly more involved than the one of Lemma 4.17(e).

Despite the fact that any confluence proof by the conversion version of decreasing diagrams can be completed into a confluence proof by the valley version using the same labels [79, Theorem 3], conversions can be significantly shorter than valleys [79, Example 8]. Regarding the conversion version of decreasing diagrams, in automated tools the main obstacle is finding suitable conversions. Even though simple heuristics [42, Section 4] have been proposed to limit the explosion in the search space when considering conversions, most automated confluence provers still favor the valley version. While those heuristics suffice for the rule labeling, Theorem 4.27 shows that in a more complex setting, conversions must satisfy additional restrictions, which make the search for suitable conversions even more challenging.

Chapter 5

Redundant Rules

Not that I have anything much against redundancy.
But I said that already.
Larry Wall (199702271735.JAA04048@wall.org)

In this chapter we present a remarkably simple technique based on the removal and addition of redundant rewrite rules, i.e., rules that can be simulated by other rules, when (dis)proving confluence of term rewrite systems. We demonstrate how automatic confluence provers benefit from the addition as well as the removal of redundant rules. Due to their simplicity, our transformations were easy to formalize in a proof assistant and are thus amenable to certification. Experimental results in Section 8.3 show the surprising gain in power.

Example 5.1 (Cop #442)**.** Consider the TRS $\mathcal{R}$ consisting of the two rewrite rules

$$\mathsf{f}(\mathsf{f}(x)) \to x \qquad\qquad \mathsf{f}(x) \to \mathsf{f}(\mathsf{f}(x))$$

The two non-trivial critical pairs

$$\mathsf{f}(\mathsf{f}(\mathsf{f}(x))) \leftarrow\!\rtimes\!\rightarrow x \qquad\qquad x \leftarrow\!\rtimes\!\rightarrow \mathsf{f}(\mathsf{f}(\mathsf{f}(x)))$$

are obviously joinable

$$\mathsf{f}(\mathsf{f}(\mathsf{f}(x))) \to_{\mathcal{R}} \mathsf{f}(x) \to_{\mathcal{R}} \mathsf{f}(\mathsf{f}(x)) \to_{\mathcal{R}} x$$

but not by a multistep, see Definition 2.34. Consequently, the result of van Oostrom [78] on development closed critical pairs does not apply. After adding the rewrite rule $\mathsf{f}(x) \to x$ to $\mathcal{R}$, we obtain four new critical pairs

$$\mathsf{f}(x) \leftarrow\!\rtimes\!\rightarrow x \qquad x \leftarrow\!\rtimes\!\rightarrow \mathsf{f}(x) \qquad \mathsf{f}(\mathsf{f}(x)) \leftarrow\!\rtimes\!\rightarrow x \qquad x \leftarrow\!\rtimes\!\rightarrow \mathsf{f}(\mathsf{f}(x))$$

The new rule ensures that $\mathsf{f}^n(x) \mathrel{-\!\!\!\circledcirc} x$ for all $n \geqslant 0$ and thus confluence of the extension follows from Theorem 3.34.

First we establish that $\mathsf{f}^n(x) \mathrel{-\!\!\!\circledcirc} x$ by induction on n. The claim is trivially true for $n = 0$. Given $\mathsf{f}^{n-1}(x) \mathrel{-\!\!\!\circledcirc} x$, we can take substitutions σ and σ' that map x to $\mathsf{f}^{n-1}(x)$ and x, respectively, and obtain $\mathsf{f}(x)\sigma \mathrel{-\!\!\!\circledcirc} x\sigma'$, i.e., $\mathsf{f}^n(x) \mathrel{-\!\!\!\circledcirc} x$.

For each of the critical pairs $s \leftarrow\!\!\rtimes\!\!\rightarrow t$, we have either $s \mathrel{-\!\!\!\circledcirc} t$ or $s \leftarrow\!\!\bowtie\!\!\rightarrow t$ and $s \mathrel{\circledcirc\!\!-} t$, which implies $s \mathrel{-\!\!\!\circledcirc} \cdot {}^*\!\leftarrow t$. Therefore the TRS is almost development closed and thus confluent. Since the new rule can be simulated by the original rules, $\mathsf{f}(x) \to_{\mathcal{R}} \mathsf{f}(\mathsf{f}(x)) \to_{\mathcal{R}} x$, also $\mathcal{R}$ is confluent.

Confluence of the TRS in Example 5.1 is hard to show directly: ACP 0.51, CoLL-Saigawa 1.1, and CSI 1.1 all fail without the technique of this chapter, but all three can prove confluence of the extended TRS. Below we explain how such extensions can be found automatically.

The next example shows that also proving non-confluence may become easier after adding rules.

Example 5.2 (Cop # 216)**.** Consider the TRS $\mathcal{R}$ [22] consisting of the eight rewrite rules

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

All critical pairs are *deeply*[1] joinable but $\mathcal{R}$ is not confluent. Two of the critical pairs are

$$\mathsf{b} \leftarrow\!\!\rtimes\!\!\rightarrow \mathsf{f}(\mathsf{h}(\mathsf{g}(\mathsf{a})), \mathsf{g}(x)) \qquad \mathsf{c} \leftarrow\!\!\rtimes\!\!\rightarrow \mathsf{f}(\mathsf{h}(x), \mathsf{g}(\mathsf{h}(\mathsf{a})))$$

After adding them as rules

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

new critical pairs are obtained, one of which is

$$\mathsf{b} \leftarrow\!\!\rtimes\!\!\rightarrow \mathsf{c}$$

[1] A critical pair $s \leftarrow\!\!\rtimes\!\!\rightarrow t$ is deeply joinable if $u \downarrow v$ for any two reducts u of s and v of t. The example defeats any non-confluence check based on proving non-joinability of peaks starting from critical peaks.

Since b and c are different normal forms, the extension is obviously non-confluent. Since the new rules can be simulated by the original rules, also $\mathcal{R}$ is non-confluent. Of the three tools mentioned, ACP shows non-confluence by first deriving the rule $\mathsf{g}(\mathsf{a}) \to \mathsf{a}$, which can also be simulated by existing rules, giving rise to a critical pair that extends to a non-joinable peak:

$$\mathsf{b} \leftarrow \mathsf{f}(\mathsf{g}(\mathsf{a}), \mathsf{g}(\mathsf{a})) \to \mathsf{f}(\mathsf{g}(\mathsf{a}), \mathsf{a}) \to^* \mathsf{c}$$

CoLL-Saigawa, and CSI (without the techniques from this chapter) fail.

The remainder of the chapter is structured as follows. In the next section we describe the theory underlying the addition and removal of rules, and Section 5.2 is devoted to its integration into CeTA. In Section 5.3 we sketch several heuristics for finding redundant rules implemented in CSI.

5.1 Theory

In this section we present the easy theory behind the use of redundant rules for proving confluence. For adding such rules we use the following folklore result.

Lemma 5.3. *If $\ell \to_{\mathcal{R}}^* r$ for every rule $\ell \to r$ from $\mathcal{S}$ then $\to_{\mathcal{R}}^* = \to_{\mathcal{R} \cup \mathcal{S}}^*$.*

Proof. The inclusion $\to_{\mathcal{R}}^* \subseteq \to_{\mathcal{R} \cup \mathcal{S}}^*$ is obvious. For the reverse direction it suffices to show that $s \to_{\mathcal{R}}^* t$ whenever $s \to_{\mathcal{S}} t$. The latter ensures the existence of a position p in s, a rewrite rule $\ell \to r$ in $\mathcal{S}$, and a substitution σ such that $s|_p = \ell\sigma$ and $t = s[r\sigma]_p$. We obtain $\ell \to_{\mathcal{R}}^* r$ from the assumption of the lemma. Closure of $\to_{\mathcal{R}}^*$ under contexts and substitutions yields the desired $s \to_{\mathcal{R}}^* t$. □

Corollary 5.4. *If $\ell \to_{\mathcal{R}}^* r$ for every rule $\ell \to r$ from $\mathcal{S}$ then $\mathcal{R}$ is confluent if and only if $\mathcal{R} \cup \mathcal{S}$ is confluent.*

Proof. We obtain $\to_{\mathcal{R}}^* = \to_{\mathcal{R} \cup \mathcal{S}}^*$ from the preceding lemma. Hence we also have $\downarrow_{\mathcal{R}} = \downarrow_{\mathcal{R} \cup \mathcal{S}}$ and $\uparrow_{\mathcal{R}} = \uparrow_{\mathcal{R} \cup \mathcal{S}}$. Therefore

$$\uparrow_{\mathcal{R}} \subseteq \downarrow_{\mathcal{R}} \qquad \iff \qquad \uparrow_{\mathcal{R} \cup \mathcal{S}} \subseteq \downarrow_{\mathcal{R} \cup \mathcal{S}} \qquad \square$$

Definition 5.5. A rule $\ell \to r \in \mathcal{R}$ is *redundant* if $\ell \to_{\mathcal{R} \setminus \{\ell \to r\}}^* r$.

By Corollary 5.4, if $\ell \to r \in \mathcal{R}$ is redundant, then $\mathcal{R}$ is confluent if and only if $\mathcal{R} \setminus \{\ell \to r\}$ is confluent. In other words, removing a redundant rule does not affect confluence of a TRS. For removing rules while reflecting[2] confluence (or adding rules while reflecting non-confluence) it suffices that the left- and right-hand side are convertible with respect to the remaining rules.

Lemma 5.6. *If $\ell \leftrightarrow^*_{\mathcal{R}} r$ for every rule $\ell \to r$ from $\mathcal{S}$ then $\leftrightarrow^*_{\mathcal{R}\cup\mathcal{S}} = \leftrightarrow^*_{\mathcal{R}}$.*

Proof. The inclusion $\leftrightarrow^*_{\mathcal{R}} \subseteq \leftrightarrow^*_{\mathcal{R}\cup\mathcal{S}}$ is obvious. For the reverse direction it suffices to show that $s \leftrightarrow^*_{\mathcal{R}} t$ whenever $s \to_{\mathcal{S}} t$. The latter ensures the existence of a position p in s, a rewrite rule $\ell \to r$ in $\mathcal{S}$, and a substitution σ such that $s|_p = \ell\sigma$ and $t = s[r\sigma]_p$. We obtain $\ell \leftrightarrow^*_{\mathcal{R}} r$ from the assumption of the lemma. Closure of $\leftrightarrow^*_{\mathcal{R}}$ under contexts and substitutions yields the desired $s \leftrightarrow^*_{\mathcal{R}} t$. □

Corollary 5.7. *If $\mathcal{R}$ is confluent and $\ell \leftrightarrow^*_{\mathcal{R}} r$ for every rule $\ell \to r$ from $\mathcal{S}$ then $\mathcal{R} \cup \mathcal{S}$ is confluent.*

Proof. From the preceding lemma and the confluence of $\mathcal{R}$ we obtain

$$\leftrightarrow^*_{\mathcal{R}\cup\mathcal{S}} \;=\; \leftrightarrow^*_{\mathcal{R}} \;\subseteq\; \downarrow_{\mathcal{R}} \;\subseteq\; \downarrow_{\mathcal{R}\cup\mathcal{S}}$$

Hence $\mathcal{R} \cup \mathcal{S}$ is confluent. □

Example 5.8 (Cop #20)**.** Reconsider the TRS from Example 4.7. Its five rules are

$$\begin{array}{lll} \mathsf{hd}(x : y) \to x & \mathsf{inc}(x : y) \to \mathsf{s}(x) : \mathsf{inc}(y) & \mathsf{nats} \to \mathsf{0} : \mathsf{inc}(\mathsf{nats}) \\ \mathsf{tl}(x : y) \to y & \mathsf{inc}(\mathsf{tl}(\mathsf{nats})) \to \mathsf{tl}(\mathsf{inc}(\mathsf{nats})) & \end{array}$$

We have shown confluence of this system using the rule labeling, i.e., decreasing diagrams. However, simply removing the last rule would make confluence obvious, since the remaining four rules constitute an orthogonal TRS. And indeed, because of the following joining sequences, the last rule is superfluous and can be dropped:

$$\mathsf{inc}(\mathsf{tl}(\mathsf{nats})) \to \mathsf{inc}(\mathsf{tl}(\mathsf{0} : \mathsf{inc}(\mathsf{nats}))) \to \mathsf{inc}(\mathsf{inc}(\mathsf{nats}))$$
$$\mathsf{tl}(\mathsf{inc}(\mathsf{nats})) \to \mathsf{tl}(\mathsf{inc}(\mathsf{0} : \mathsf{inc}(\mathsf{nats}))) \to \mathsf{tl}(\mathsf{s}(\mathsf{0}) : \mathsf{inc}(\mathsf{inc}(\mathsf{nats}))) \to \mathsf{inc}(\mathsf{inc}(\mathsf{nats}))$$

[2] We are interested in transformations that reflect (rather than preserve) confluence, because our goal is automation, and it is natural to work from the conclusion for finding proofs.

Some other examples from [34] can be dealt with in a similar fashion: In [34, Example 1] the first rule is joinable using the other rules, and the remaining system is orthogonal. The same argument (with a different joining conversion) applies to [34, Example 5].

Corollary 5.7 can also be beneficial when dealing with non-left-linear systems, as demonstrated by the following example.

Example 5.9 (Cop #233)**.** Consider the TRS from [103] consisting of the four rewrite rules

$$\begin{aligned} \mathsf{f}(x,x) &\to \mathsf{f}(\mathsf{g}(x),\mathsf{g}(x)) & \mathsf{f}(x,y) &\to \mathsf{f}(\mathsf{h}(x),\mathsf{h}(y)) \\ \mathsf{g}(x) &\to \mathsf{p}(x) & \mathsf{h}(x) &\to \mathsf{p}(x) \end{aligned}$$

Because of the conversion

$$\begin{aligned} \mathsf{f}(x,x) \to \mathsf{f}(\mathsf{h}(x),\mathsf{h}(x)) \to \mathsf{f}(\mathsf{p}(x),\mathsf{g}(x)) \to \mathsf{f}(\mathsf{p}(x),\mathsf{p}(x)) \\ \leftarrow \mathsf{f}(\mathsf{g}(x),\mathsf{p}(x)) \leftarrow \mathsf{f}(\mathsf{g}(x),\mathsf{g}(x)) \end{aligned}$$

we can remove the first rule. Since the resulting TRS is orthogonal and the removed rule is convertible using the other rules, also the original TRS is confluent.

It can also be beneficial to both add and remove rules. In particular adding a redundant rule can help with removing other, problematic rules, as shown in the following example.

Example 5.10 (Cop #412)**.** Consider the TRS consisting of the three rewrite rules

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

After adding the rule $\mathsf{f}(x,y) \to \mathsf{a}$, which is justified by the rewrite sequence $\mathsf{f}(x,y) \to \mathsf{f}(\mathsf{g}(x),\mathsf{g}(x)) \to \mathsf{a}$, we can remove the first two original rules, due to the following conversions:

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

The resulting TRS is orthogonal and hence confluent. Since the added rule can be simulated by the original rules, and the removed rules are convertible using the new rule, also the original TRS is confluent.

While adding (or removing) rules using Corollary 5.4 is always safe in the sense that we cannot lose confluence, it is easy to see that the reverse direction of Corollary 5.7 does not hold in general. That is, removing convertible rules can make a confluent TRS non-confluent as for example witnessed by the two TRSs $\mathcal{R} = \{\mathsf{a} \to \mathsf{b},\ \mathsf{a} \to \mathsf{c}\}$ and $\mathcal{S} = \{\mathsf{b} \to \mathsf{a}\}$. Clearly $\mathcal{R}$ is not confluent, $\mathcal{S} \cup \mathcal{R}$ is confluent, and $\mathsf{b} \leftrightarrow^*_{\mathcal{R}} \mathsf{a}$.

We give one more example, showing that the removal of redundant rules can considerably speed up finding a confluence proof.

Example 5.11 (Cop #424)**.** Consider the TRS consisting of the following two rules:

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

This TRS is confluent by the simultaneous critical pair criterion of Okui [75][3]. Alas, there are 58 simultaneous critical pairs and indeed ACP 0.51, which implements Okui's criterion, does not terminate in ten minutes. While 58 looks small, the simultaneous critical pairs become quite big. For example, with $t = \mathsf{g}(\mathsf{f}^3(\mathsf{g}(x, \mathsf{f}(x))), \mathsf{f}^4(\mathsf{g}(x, \mathsf{f}(x))))$, one of the simultaneous critical pairs is

$$\mathsf{f}^3(\mathsf{g}(\mathsf{f}(\mathsf{g}(\mathsf{f}(t), \mathsf{f}(\mathsf{f}(t))), \mathsf{f}(\mathsf{f}(\mathsf{g}(\mathsf{f}(t), \mathsf{f}(\mathsf{f}(t)))))))) \mathrel{\reflectbox{$\to$}\!\!\bowtie\!\!\to} \mathsf{f}^5(\mathsf{g}(\mathsf{f}^3(x), \mathsf{f}^4(x)))$$

and testing joinability using development steps is very expensive. In general, if one takes the rules $\mathsf{f}(x) \to \mathsf{g}(x, \mathsf{f}(x))$ and $\mathsf{f}^n(\mathsf{f}(x)) \to \mathsf{f}^n(\mathsf{g}(x, \mathsf{f}(x)))$, then the number and size of the simultaneous critical pair will grow exponentially in n. However, Corollary 5.7 is applicable—the second rule can be simulated by the first rule in one step—and showing confluence of the first rule is trivial.

5.2 Formalization and Certification

The redundant rules technique is particularly well suited for certification for the following reasons. First, since the theory we use is elementary, formalizing it in a proof assistant is entirely straightforward. Moreover the generated

[3]Note that the given TRS is feebly orthogonal [80]. The key observation here is that any simultaneous critical pair arises from a peak between a development step and a plain rewrite step. By the orthogonalization procedure from [80], we can obtain an equivalent peak of two orthogonal development steps, which is joinable by two development steps, thus satisfying Okui's criterion.

proofs, while simple in nature, can become very large, which makes checking them infeasible by hand, but easy for a machine. Finally, as demonstrated in Section 8.3, the existing certifiable confluence techniques heavily benefit from our transformations.

To add support for our transformations to CeTA we formalized the results from Section 5.1 in Isabelle and integrated them into IsaFoR. The theory `Redundant_Rules.thy` contains the theoretical results, whose formalization, directly following the paper proof, requires a mere 100 lines of Isabelle, stressing the simplicity of the transformations.

We extended CPF for representing proofs using addition and removal of redundant rules and implemented dedicated check functions in the theory `Redundant_Rules_Impl.thy`, enabling CeTA to certify such (non-)confluence proofs. A certificate for (non-)confluence of a TRS $\mathcal{R}$ by an application of the redundant rules transformation consists of three parts:

- the modified TRS $\mathcal{R}'$,
- a certificate for the (non-)confluence of $\mathcal{R}'$, and
- a justification for redundancy of the added and removed rules. Here for the rules that were added, i.e., all $\ell \to r$ in $\mathcal{S} = \mathcal{R}' \setminus \mathcal{R}$, we simply require a bound on the length of the derivations showing $\ell \to^*_{\mathcal{R}} r$. This bound is necessary to ensure termination of the check function. For the deleted rules in a non-confluence certificate, i.e., all $\ell \to r$ in $\mathcal{S} = \mathcal{R} \setminus \mathcal{R}'$, the same bound is used for $\ell \to^*_{\mathcal{R}'} r$. For a confluence proof one can either give explicit conversions $\ell \leftrightarrow^*_{\mathcal{R}'} r$ or rely on the bound again, which then has to ensure $\ell \downarrow_{\mathcal{R}'} r$.

Implementing check functions for such a certificate is then straightforward. We simply compute $\mathcal{S} \setminus \mathcal{R}$ and $\mathcal{R} \setminus \mathcal{S}$ and use the given bound and conversions to ensure redundancy.

Whereas for certification we only need to check that the modified rules really are redundant, the question of how to automatically find suitable rules for addition and deletion is more intricate. In the next section we discuss and evaluate our implementation of several possible approaches in the confluence prover CSI, see also Chapter 8.

5.3 Heuristics

CSI features a powerful strategy language, see Section 8.2, which allows to combine confluence techniques in a modular and flexible manner, making it easy to test different strategies that exploit redundant rules. We use the following four heuristics to add and remove rules.

Joining Sequences (js) Our first strategy is to add (minimal) joining sequences of critical pairs as rules, i.e., in Corollary 5.4 we choose

$$\mathcal{S} \subseteq \{s \to u, t \to u \mid s \leftarrow\!\rtimes\!\rightarrow t \text{ with } s \to_{\mathcal{R}}^* u \text{ and } t \to_{\mathcal{R}}^* u\}$$

The underlying idea here is that critical peaks become joinable in a single step, which is advantageous for other confluence criteria, for example rule labeling. This heuristic solves e.g. Example 5.2.

Rewriting Right-Hand Sides (rhs) The second strategy for obtaining redundant rules to add, is to rewrite right-hand sides of rules, i.e., in Corollary 5.4 set

$$\mathcal{S} = \{\ell \to t \mid \ell \to r \in \mathcal{R} \text{ and } r \to_{\mathcal{R}} t\}$$

Again the motivation is to produce shorter joining sequences for critical pairs, facilitating the use of other confluence criteria. For instance, in Example 5.1 this heuristic derives the desired rule.

Forward Closures (fc) The final strategy for adding rules is based on instantiating the right-hand side before rewriting it.

Definition 5.12. Given two variable disjoint rewrite rules $\ell_1 \to r_1$ and $\ell_2 \to r_2$ of a TRS $\mathcal{R}$, a function position p in r_1, and an mgu σ of $r_1|_p$ and ℓ_2, the rewrite rule $\ell_1\sigma \to r_1\sigma[r_2\sigma]_p$ is a *forward closure* of $\mathcal{R}$. We write $\mathsf{FC}(\mathcal{R})$ for the extension of $\mathcal{R}$ with all its forward closures.

We use this process in both directions, i.e., we also extend the left-hand sides of rules—more precisely in Corollary 5.4 we use

$$\mathcal{S} = \mathsf{FC}(\mathcal{R}) \cup \mathsf{FC}(\mathcal{R}^{-1})^{-1}$$

Example 5.13 (Cop #47)**.** Consider the TRS $\mathcal{R}$ due to Klop [52] consisting of the three rules

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

Because of the rewrite sequence

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

we also have $\mathsf{c} \to \mathsf{g}(\mathsf{c}) \to^* \mathsf{g}(\mathsf{a})$ and since a and $\mathsf{g}(\mathsf{a})$ are not joinable, which can be shown using tree automata techniques [27], $\mathcal{R}$ is not confluent. However, finding this conversion is non-trivial and indeed ACP 0.51, CoLL-Saigawa 1.1, and CSI without redundant rules fail. Using forward closures to derive redundant rules we find

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

and hence $\mathsf{a} \leftarrow\!\rtimes\!\rightarrow \mathsf{g}(\mathsf{a}) \in \mathsf{CP}(\mathsf{FC}^4(\mathcal{R}))$.

To see why including $\mathsf{FC}(\mathcal{R}^{-1})^{-1}$ is beneficial, consider the following example.

Example 5.14 (Cop #46)**.** The TRS consisting of the rules

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

due to Huet [45] is not confluent because

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

is a peak connecting two distinct normal forms. There is an overlap between $\mathsf{f}(\mathsf{c},\mathsf{c}) \to \mathsf{b} \in \mathsf{FC}(\mathcal{R}^{-1})^{-1}$ and $\mathsf{f}(x,x) \to \mathsf{a} \in \mathcal{R}$, resulting in the critical pair $\mathsf{a} \leftarrow\!\rtimes\!\rightarrow \mathsf{b}$. Note that considering $\mathsf{FC}(\mathcal{R})$ alone does not yield any progress in this example.

Deleting Rules (del) For removing rules we search for rules whose left- and right-hand sides are joinable, i.e., in Corollary 5.7 set

$$\mathcal{S} = \{\ell \to r \mid \ell \downarrow_{\mathcal{R}} r\}$$

This decision is motivated by simplicity of implementation and the fact that for confluent TRSs, joinability and convertibility coincide. Removing rules can benefit confluence proofs by eliminating critical pairs. Since our strategy here is a simple greedy one that removes as many rules as possible, we also lose confluence in some cases.

In the case of adding rules we also discard rules that can be simulated by other rules in a single step. Without this refinement, the gain in power would become smaller, and even disappear for CSI's full strategy.

We also implemented and tested three other strategies, which did not yield any additional proofs:

- Inspired by Example 5.1 we tried to add rules specifically for making rewrite systems (almost) development closed. That is, we used

  $$\mathcal{S} = \{s \to t \mid s \leftarrow\!\!\rtimes\!\!\rightarrow t \text{ with } s \to_{\mathcal{R}}^{*} t \text{ and not } s \twoheadrightarrow_{\mathcal{R}} t\}$$

  in Corollary 5.4. All examples gained by this strategy can also be handled by (js) or (rhs).

- To help with systems containing AC-like rules we tried to add inverted reversible rules, by setting

  $$\mathcal{S} = \{r \to \ell \mid \ell \to r \in \mathcal{R} \text{ with } r \to_{\mathcal{R}}^{*} \ell\}$$

  in Corollary 5.4. Again we gained no additional proofs compared to (js) and (rhs).

- When removing rules we also tried to search for conversions that are not valleys, by using rules in the reverse direction when searching for a join. More precisely, we tried

  $$\mathcal{S} = \{\ell \to r \mid \ell \downarrow_{\mathcal{R} \cup \mathcal{R}^{-1}} r\}$$

  in Corollary 5.7, but this variation only lost examples compared to (del).

5.4 Summary

In this chapter we demonstrated how a very simple technique, namely adding and removing redundant rules, can boost the power of automated confluence

provers. It is easy to implement and we believe that also confluence tools other than CSI could benefit from such transformations, not only increasing their power, but also simplifying the generated proofs. Moreover the technique is well-suited for certification, resulting in more trustworthy proofs. Interestingly we observed that most of the systems gained by (del), see also Section 8.3, become orthogonal by removing redundant rules. This might be due to the fact that when designing example TRSs for new techniques, one often works by systematically making existing criteria non-applicable and removing rules can undo this effort.

Chapter 6

Confluence of the Lambda Calculus

A man provided with paper, pencil, and rubber,
and subject to strict discipline,
is in effect a universal machine.

Alan Turing

In the previous chapters we were concerned with certifiable confluence analysis for first-order rewriting. Now we turn our attention to higher-order formalisms. Before discussing automatable techniques for proving confluence of higher-order rewrite systems in the next chapter, we stay in the realm of formalization and discuss a formalized confluence proof for the quintessential higher-order rewrite system: the λ-calculus. Anticipating future work, the presented formalization could serve as the basis for formalizing results for higher-order rewriting.

In this chapter we give a short proof of the Church-Rosser property for the λ-calculus enjoying two distinguishing features: firstly, it employs the Z-property, resulting in a short and elegant proof; and secondly, it is formalized in the nominal higher-order logic available for Isabelle/HOL.

Dehornoy proved confluence for the rule of self-distributivity $xyz \to xz(yz)$[1] by means of a novel method [19], the idea being to give a map $\bullet$ that is monotone with respect to $\to^*$ and that yields for each object an upper bound on all objects reachable from it in a single step. Later, this method was extracted and dubbed the Z-property [20], and applied to prove confluence of various rewrite systems, in particular the λ-calculus.

Here we present our Isabelle/HOL formalization of part of the above mentioned work, in particular that the λ-calculus with β-reduction is confluent since it enjoys the Z-property and that the latter property is equivalent to an abstract version of Takahashi's confluence method [104]. We achieve a rigorous

[1] Confluence of this single-rule term rewrite system is hard: presently no tool can prove it automatically.

```
nominal_datatype term =
  Var name
| App term term
| Abs x::name t::term binds x in t
```

Listing 10: Nominal λ-terms.

treatment of λ-terms modulo α-equivalence by employing Nominal Isabelle [109], a nominal higher-order logic based on Isabelle/HOL. The formalization is available from the archive of formal proofs [26].

6.1 Nominal Lambda Terms

In our formalization λ-terms are represented by the nominal data type shown in Listing 10, where the annotation "**binds** x **in** t" indicates that the equality of such abstraction terms is up to renaming of x in t, i.e., terms are modulo α using an automatic quotient construction. For the sake of readability we will use standard notation in the remainder of this chapter: we drop the Var constructor and write x instead of Var x, denote application by juxtaposition, i.e., write $s\ t$ instead of App $s\ t$, and use $\lambda x.\ t$ instead of Abs $x\ t$.

Instead of using the variable convention, when defining functions on λ-terms, we use Nominal Isabelle's *freshness constraints*. A freshness constraint is written $x \sharp t$ and states that x does not occur in t, or equivalently, that x is fresh for t. In principle it is always possible to rename variables in terms, or any finitely supported structure, away from a given finite collection of variables. In order to relieve the user of doing so by hand, Nominal Isabelle provides infrastructure that takes care of appropriate renaming. For instance, the following rule for performing case analysis on nominal λ-terms is generated:

$$\begin{aligned}\forall c\, t\, P.\ &(\forall x.\ t = x \Longrightarrow P\ t)\\ &\wedge\ (\forall s\ u.\ t = s\ u \Longrightarrow P\ t)\\ &\wedge\ (\forall x\ s.\ x \sharp c \wedge t = \lambda x.\ s \Longrightarrow P\ t)\\ &\Longrightarrow P\ t\end{aligned}$$

Here c is a freshness context that can be instantiated to any structure that contains finitely many bound variables. Freshness constraints can then be

$y\,[x := s] = (\textbf{\textit{if}}\ x = y\ \textbf{\textit{then}}\ s\ \textbf{\textit{else}}\ y)$
$(t\ u)\,[x := s] = t\,[x := s]\ u\,[x := s]$
$y \sharp (x,\ s) \Longrightarrow (\lambda y.\ t)\,[x := s] = \lambda y.\ t\,[x := s]$

Listing 11: Capture-avoiding substitution in Nominal Isabelle.

used for defining nominal functions, giving rise to strong induction principles, similar to the rule above, with which one can reason like on paper, simply stating that bound and free variables are to be distinct, by instantiating c.

For instance, consider the definition of capture-avoiding substitution.

Definition 6.1. For terms s and t and a variable x, the *capture-avoiding substitution* of s for x in t is denoted by $t\,[x := s]$ and defined as follows:

$$t\,[x := s] = \begin{cases} s & \text{if } t = x \\ y & \text{if } t = y \text{ and } y \neq x \\ u\,[x := s]\ v\,[x := s] & \text{if } t = u\ v \\ \lambda y.\ u\,[x := s] & \text{if } t = \lambda y.\ u \text{ and } y \neq x \text{ and } y \sharp s \end{cases}$$

The formal definition as nominal function in Isabelle is show in Listing 11, where the last equation is only applicable when y is fresh for x and s, i.e., does not occur in s and is different from x. Note that on nominal terms these equations define a total function, because the freshness constraint can always be fulfilled by choosing an appropriate representative.

However, nominal functions do not come for free. After stating the defining equations, one is faced with proof obligations that ensure pattern-completeness, termination, equivariance, and well-definedness. With the help of some home-brewed Eisbach [63] methods we were able to handle those obligations automatically. We now illustrate the strong induction principle for nominal λ-terms on the Substitution Lemma [12, Lemma 2.1.16].

Lemma 6.2. *For all variables x and y and all terms s, t, and u we have*

$$x \sharp (y,\ u) \Longrightarrow t\,[x := s]\,[y := u] = t\,[y := u]\,[x := s\,[y := u]]$$

Proof. In principle the proof proceeds by induction on t. However, for the case of λ-abstractions we additionally want the bound variable to be fresh for s, u, x, and y. With Nominal Isabelle it is enough to indicate that the variables of

those terms should be avoided in order to obtain appropriately renamed bound variables. We will not mention this fact again in future proofs.

- In the base case $t = z$ for some variable z. If $z = x$ then we have $t\,[x := s]\,[y := u] = s\,[y := u]$ and $t\,[y := u]\,[x := s\,[y := u]] = s\,[y := u]$, since then $z \neq y$ and thus $z\,[y := u] = z$. Otherwise $z \neq x$. Now if $z = y$, then $t\,[x := s]\,[y := u] = u$ and $t\,[y := u]\,[x := s\,[y := u]] = u$, since $x \sharp u$. If $z \neq y$ then both ends of the equation reduce to z and we are done.

- In case of an application, we conclude by definition and using the induction hypothesis twice.

- Now for the interesting case. Let $t = \lambda z.\, v$ such that $z \sharp (s, u, x, y)$. Then

$$\begin{aligned}(\lambda z.\, v)\,[x := s]\,[y := u] &= \lambda z.\, v\,[x := s]\,[y := u] \\ &= \lambda z.\, v\,[y := u]\,[x := s\,[y := u]] \\ &= (\lambda z.\, v)\,[y := u]\,[x := s\,[y := u]]\end{aligned}$$

  The first equality holds by $z \sharp (s, u, x, y)$, while the second follows from the induction hypothesis. For the last step we need $z \sharp (s\,[y := u], u, x, y)$, where $z \sharp s\,[y := u]$ follows from $z \sharp (s, u, y)$ by a straightforward induction on s. □

Using substitution we can define β-reduction in the expected way.

Definition 6.3. We define a *β-step* inductively by the compatible closure [12, Definition 3.1.4] of the *β-rule* in a nominal version:

$$\frac{x \sharp t}{(\lambda x.\, s)\ t \to_\beta s\,[x := t]} \qquad \frac{s \to_\beta t}{s\ u \to_\beta t\ u} \qquad \frac{s \to_\beta t}{u\ s \to_\beta u\ t} \qquad \frac{s \to_\beta t}{\lambda x.\, s \to_\beta \lambda x.\, t}$$

The last three rules taken together yield closure under contexts, while the first rule employs a freshness constraint in order to define a root β-step with the help of capture-avoiding substitution. If we would drop the freshness constraint, the resulting induction principle would not be strong enough with respect to avoiding capture of bound variables.

The following standard "congruence properties" will be used freely in the remainder.

Lemma 6.4. *Let x be a variable and s, s', t, and t' be terms.*

(a) Z. (b) Triangle.

Figure 15: The Z and triangle properties.

- *If* $s \to_\beta^* s'$ *and* $t \to_\beta^* t'$ *then* $s\ t \to_\beta^* s'\ t'$
- *If* $s \to_\beta^* s'$ *then* $\lambda x.\ s \to_\beta^* \lambda x.\ s'$
- *If* $s \to_\beta^* s'$ *and* $t \to_\beta^* t'$ *then* $t[x := s] \to_\beta^* t'[x := s']$ □

They are proved along the lines of their textbook proofs [12, Lemma 3.1.6 and Proposition 3.1.16], the first two by induction on the length of the rewrite sequences and the last one by nominal induction on t followed by a nested nominal induction on the definition of β-steps, using Lemma 6.2.

Moreover we will make use of the easily proved fact that β-reduction is coherent with abstraction, i.e., steps in an abstraction must happen below that abstraction.

Lemma 6.5. *Let x be a variable and s and t be terms. If $\lambda x.\ s \to_\beta^* t$ then there is a term u with $t = \lambda x.\ u$ and $s \to_\beta^* u$.* □

6.2 The Z Property

We now present the Z-property for abstract rewriting, show that it implies confluence, and then instantiate it for the case of nominal λ-terms modulo α equipped with β-reduction.

Definition 6.6. A binary relation $\to$ on a set A has the *Z-property* if there is a map $\bullet : A \to A$ such that $a \to b$ implies $b \to^* a^\bullet$ and $a^\bullet \to^* b^\bullet$.

The Z-property is illustrated in Figure 15(a), which also explains the name. If a relation $\to$ has the Z-property then it is monotone, i.e., $a \to^* b$ implies

$a^\bullet \to^* b^\bullet$, which is straightforward to show by induction on the length of the rewrite sequence from a to b.

Lemma 6.7. *A relation that has the Z-property is confluent.*

Proof. We show semi-confluence. Assume $a \to^* c$ and $a \to d$. We show $d \downarrow c$ by case analysis on the rewrite sequence from a to c. If it is empty there is nothing to show. Otherwise there is a b with $a \to^* b$ and $b \to c$. Then by monotonicity we have $a^\bullet \to^* b^\bullet$. From $a \to d$ we have $d \to^* a^\bullet$ using the Z-property, so in total $d \to^* b^\bullet$. By applying the Z-property to $b \to c$ we also get $c \to^* b^\bullet$ and consequently $d \downarrow c$ as desired. □

There are two natural choices for functions on λ-terms that yield the Z-property for $\to_\beta$, namely the full-development function and the full-super-development function. The former maps a term to the result of contracting all residuals of redexes in it [12, Definition 13.2.7] and the latter also contracts upward-created redexes [87, Section 2.4]. Here we opt for the latter, which requires less case analysis. To define super-developments we use an auxiliary function, namely a variant of App with built-in β-reduction at the root.

Definition 6.8. The function $\cdot_\beta$ is defined by the following equations:

$$x \sharp u \Longrightarrow (\lambda x.\, s') \cdot_\beta u = s'[x := u]$$
$$x \cdot_\beta u = x\ u$$
$$(s\ t) \cdot_\beta u = s\ t\ u$$

Case analysis on the first argument shows that this function satisfies the following congruence-like property.

Lemma 6.9. *For all terms s, s', t, and t' if $s \to_\beta^* s'$ and $t \to_\beta^* t'$ then $s \cdot_\beta t \to_\beta^* s' \cdot_\beta t'$.* □

Definition 6.10. The *full-superdevelopment function* $\bullet$ on λ-terms is defined recursively as

$$x^\bullet = x$$
$$(\lambda x.\, t)^\bullet = \lambda x.\, t^\bullet$$
$$(s\ t)^\bullet = s^\bullet \cdot_\beta t^\bullet$$

Note the use of $\cdot_\beta$ to contract created redexes in the third equation. The following example illustrates the effect.

Example 6.11. Consider the rewrite sequence

$$(\lambda x.\, x)\ (\lambda y.\, y)\ z \to_\beta (\lambda y.\, y)\ z \to_\beta z$$

Since applying $\bullet$ to the starting term yields

$$\begin{aligned}((\lambda x.\, x)\ (\lambda y.\, y)\ z)^\bullet &= ((\lambda x.\, x)\ (\lambda y.\, y))^\bullet \cdot_\beta z^\bullet \\ &= ((\lambda x.\, x)^\bullet \cdot_\beta (\lambda y.\, y)^\bullet) \cdot_\beta z \\ &= ((\lambda x.\, x) \cdot_\beta (\lambda y.\, y)) \cdot_\beta z \\ &= (\lambda y.\, y) \cdot_\beta z \\ &= z\end{aligned}$$

it constitutes a full-superdevelopment. Note that it is not a development because the redex $(\lambda y.\, y)\ z$ is not present in the starting term of the reduction, but created by the first step.[2]

Below, we freely use the fact that $s^\bullet\ t^\bullet \to_\beta^= (s\ t)^\bullet$, which is shown by considering whether or not $s^\bullet$ is an abstraction.

The structure of the proof that $\to_\beta$ has the Z-property proceeds via two auxiliary results: the first expresses that each term self-expands to its full-superdevelopment, and the latter that applying $\bullet$ to the right-hand side of the β-rule, i.e., to the result of a substitution, "does more" than applying the map to its components first. Both are proved by structural induction.

Lemma 6.12. *For all terms t we have $t \to_\beta^* t^\bullet$.*

Proof. By induction on t.

- If $t = x$ then $t^\bullet = x$ and trivially $x \to_\beta^* x$.
- If $t = \lambda x.\, s$ then $t^\bullet = \lambda x.\, s^\bullet$. Since $s \to_\beta^* s^\bullet$ by the induction hypothesis we also have $\lambda x.\, s \to_\beta^* \lambda x.\, s^\bullet$.
- If $t = t_1\ t_2$ then we have $t_1\ t_2 \to_\beta^* t_1^\bullet\ t_2^\bullet$ by the induction hypothesis. If $t_1^\bullet$ is not an abstraction then $t^\bullet = t_1^\bullet\ \cdot_\beta\ t_2^\bullet = t_1^\bullet\ t_2^\bullet$ and we are done.

[2] Like developments, superdevelopments are finite [87]. In particular the rewrite sequence $\Omega \to_\beta \Omega \to_\beta \Omega$ for $\Omega = (\lambda x.\, x\ x)(\lambda x.\, x\ x)$ is not a superdevelopment, because the redex in the second step was not *upward*-created.

Otherwise $t_1^\bullet = \lambda x.\, s$ for some variable x and term s and we conclude by

$$\begin{aligned} t &= t_1\ t_2 \\ &\to_\beta^* (\lambda x.\, s)\ t_2^\bullet \\ &\to_\beta s\,[x := t_2^\bullet] \\ &= (\lambda x.\, s) \cdot_\beta t_2^\bullet \\ &= t^\bullet \end{aligned}$$

□

Lemma 6.13. *We have $t^\bullet\,[x := s^\bullet] \to_\beta^* t\,[x := s]^\bullet$ for all terms s, t and all variables x.*

Proof. We perform induction on t. The cases $t = x$ and $t = \lambda y.\, t'$ are straightforward. If $t = t_1\ t_2$ we continue by case analysis on $t_1^\bullet$.

- If $t_1^\bullet = \lambda y.\, u$, then $\lambda y.\, u\,[x := s^\bullet] = t_1^\bullet\,[x := s^\bullet] \to_\beta^* t_1\,[x := s]^\bullet$ by the induction hypothesis. Then, using Lemma 6.5, we obtain a term v such that $t_1\,[x := s]^\bullet = \lambda y.\, v$ and $u\,[x := s^\bullet] \to_\beta^* v$. We then have

$$\begin{aligned} (t_1\ t_2)^\bullet\,[x := s^\bullet] &= u\,[y := t_2^\bullet]\,[x := s^\bullet] \\ &= u\,[x := s^\bullet]\,[y := t_2^\bullet\,[x := s^\bullet]] \\ &= v\,[y := t_2^\bullet\,[x := s^\bullet]] \\ &\to_\beta^* v\,[y := t_2\,[x := s]^\bullet] \end{aligned}$$

  using Lemma 6.2 in the second, $u\,[x := s^\bullet] \to_\beta^* v$ in the third, and the induction hypothesis for t_2 in the final step. Since we also have $(t_1\ t_2)\,[x := s]^\bullet = (t_1\,[x := s]\ t_2\,[x := s])^\bullet = v\,[y := t_2\,[x := s]^\bullet]$ we conclude this case.

- If $t_1^\bullet$ is not an abstraction, then from the induction hypothesis we have

$$\begin{aligned} (t_1\ t_2)^\bullet\,[x := s^\bullet] &= t_1^\bullet\,[x := s^\bullet]\ t_2^\bullet\,[x := s^\bullet] \\ &\to_\beta^* t_1\,[x := s]^\bullet\ t_2\,[x := s]^\bullet \\ &\to_\beta^= (t_1\ t_2)\,[x := s]^\bullet \end{aligned}$$

□

Lemma 6.14. *The full-superdevelopment function $\bullet$ yields the Z-property for $\to_\beta$, i.e., if $s \to_\beta t$ then $t \to_\beta^* s^\bullet$ and $s^\bullet \to_\beta^* t^\bullet$ for all terms s and t.*

Proof. Assume $s \to_\beta t$. We continue by induction on the derivation of $\to_\beta$.

- If $s \to_\beta t$ is a root step then $s = (\lambda x.\, s')\ t'$ and $t = s'[x := t']$ for some s' and t'. Consequently $s^\bullet = s'^\bullet[x := t'^\bullet]$ and thus $t \to_\beta^* s^\bullet$ using Lemma 6.12 twice, and $s^\bullet \to_\beta^* t^\bullet$ by Lemma 6.13.
- In case the step happens below an abstraction write $s = \lambda x.\, s'$, $t = \lambda x.\, t'$, and $s' \to_\beta t'$. The induction hypothesis yields $t' \to_\beta^* s'^\bullet \to_\beta^* t'^\bullet$ and hence $\lambda x.\, t' \to_\beta^* \lambda x.\, s'^\bullet = (\lambda x.\, s')^\bullet$ and $\lambda x.\, s'^\bullet \to_\beta^* \lambda x.\, t'^\bullet = (\lambda x.\, t')^\bullet$.
- If the step happens in the left argument of an application then $s = s'\ u$ and $t = t'\ u$. From the induction hypothesis and Lemma 6.12 we have $t'\ u \to_\beta^* s'^\bullet\ u^\bullet \to_\beta^= (s\ u)^\bullet$. That also $(s'\ u)^\bullet \to_\beta^* (t'\ u)^\bullet$ follows directly from the induction hypothesis.
- The case where the step happens in the right argument of an application is symmetric. □

6.3 The Triangle Property

The Z-property is also closely related to the triangle property. In [104] confluence for the β-rule of the λ-calculus was proved by a method the novel idea of which was to define a function $\bullet$ that maps a given λ-term t to the result of a full development, i.e., of contracting all its redexes, and to show that any, not necessarily full, development from t can be extended by another development to $t^\bullet$. This property is known as the triangle property.

Definition 6.15. A binary relation $\to$ on A has the *triangle property* for a map $\bullet : A \to A$, and binary relation $\twoheadrightarrow$ on A, if $\to \subseteq \twoheadrightarrow \subseteq \to^*$ and $a \twoheadrightarrow b$ implies $b \twoheadrightarrow a^\bullet$.

Sometimes the additional condition $a \twoheadrightarrow a^\bullet$ is imposed which completes the triangle as shown in Figure 15(b) and explains the name.

Lemma 6.16. *A relation $\to$ has the Z-property for $\bullet$ if and only if it has the triangle property for $\bullet$ and some relation $\twoheadrightarrow$.*

Proof. First assume that $\to$ has the triangle property for $\bullet$ and $\twoheadrightarrow$. To show that $\to$ has the Z-property assume $a \to b$. Then by assumption we also have

$a \twoheadrightarrow b$ and hence $b \twoheadrightarrow a^\bullet$ and $a^\bullet \twoheadrightarrow b^\bullet$, by applying the triangle property twice, which together with $\twoheadrightarrow \subseteq \to^*$ yields the Z-property.

Now assume $\to$ has the Z-property for $\bullet$. We define the $\bullet$-*development* relation as $a \twoheadrightarrow b$ if $a \to^* b$ and $b \to^* a^\bullet$. Then $\to \subseteq \twoheadrightarrow \subseteq \to^*$ follows from the definition of $\twoheadrightarrow$ and the Z-property. The triangle itself directly follows from the definition of $\twoheadrightarrow$ and monotonicity of $\bullet$. □

The relation $\twoheadrightarrow$ defined in the above proof is interesting in its own right. It allows for a syntax-free definition of development relative to any given $\bullet$-function: a reduces to b by a $\bullet$-development if b is between a and $a^\bullet$. Note that the classic (super)development relation in the λ-calculus is a restriction of the syntax-freely defined $\twoheadrightarrow$ via the full-(super)development map as $\bullet$. That $s \twoheadrightarrow t$ whenever s reduces to t by a (super)development is easy to see. However the reverse does not hold.

Example 6.17. Let $I = \lambda x.\, x$ and $s = (\lambda y.\, I)\ ((\lambda x.xx)\ I)$. We have

$$s \to_\beta^* (\lambda y.\, I)\ I \to_\beta^* I = s^\bullet$$

and hence $s \twoheadrightarrow (\lambda y.\, I)\ I$ but there is no (super)development from s to $(\lambda y.\, I)\ I$.

6.4 Assessment

Three well-known methods in the literature for showing confluence of the λ-calculus are:

(a) Complete developments have the diamond property, due to Tait and Martin–Löf [12, Section 3.2].

(b) Complete developments have the triangle property with respect to the full-development function, due to Takahashi [104].

(c) Full-developments have the Z-property, due to Dehornoy and van Oostrom [20].

Our proof varies on this picture along yet another dimension, replacing developments (due to Church and Rosser, cf. [12, Definition 11.2.11]) by superdevelopments (due to Aczel, cf. [87, Section 2.7]). Where full-developments give a "tight"

upper bound on the single-step reducts of a given term, full-superdevelopments do not.

Formalizing confluence of the λ-calculus has a long history, for which we refer to Section 9.1. Here we compare our formalization in Isabelle to two others, Nipkow's formalization in Isabelle/HOL [73], as currently distributed with Isabelle, and Urban and Arnaud's formalization in Nominal Isabelle.[3] There are two major differences of the present proof to Nipkow's formalization. On the one hand Nipkow uses de Brujin indices to represent λ-terms. This considerably increases the size of the formal theories – almost 200 lines of the roughly 550 line development are devoted to setting up terms and the required manipulations on indices. Our development is 300 lines (60 of which are used for our ad hoc Eisbach methods). Of course the complexity of α-equivalence is delegated to Nominal Isabelle in our proof. The second difference is the actual technique used to show confluence: Nipkow proceeds by establishing the diamond property for complete developments. Urban and Arnaud establish the triangle property for multisteps with respect to the full-development function. The use of an auxiliary rewrite relation results in a 100 line increase compared to our formalization, which only needs the $\bullet$ function.

[3] `https://nms.kcl.ac.uk/christian.urban/Nominal/download.html`

Chapter 7

Confluence of Higher-Order Rewriting

Sometimes, the elegant implementation is just a function.
Not a method. Not a class. Not a framework.
Just a function.
John Carmack

We now turn to confluence of higher-order (pattern) rewrite systems (PRSs) as introduced in Section 2.5. Intuitively due to their restricted left-hand sides PRSs behave mostly like TRSs for confluence. However, the presence of bound and functional variables leads to several subtle differences and effects. For instance, the possibility of nesting variables in right-hand sides of rules breaks the desired properties of parallel reduction.

In the remainder of this chapter we describe the confluence criteria implemented in our tool CSI^ho. Starting from critical pairs and a higher-order critical pair lemma due to Nipkow [70] we first show how confluence of terminating PRSs is decidable, like in the first-order setting. Here the main difficulties are dealing with bound variables in the definition of critical pairs and computing them via higher-order unification. For possibly non-terminating systems we discuss two more classical criteria, orthogonality and development closed critical pairs. Finally we investigate modularity of confluence for PRSs and a higher-order version of the redundant rules technique from Chapter 5.

7.1 Higher-Order Critical Pairs

To formulate confluence criteria for PRSs we first need a suitable notion of critical pair. To define critical pairs in the presence of λ we need to deal with bound variables becoming free, when taking a subterm of a rule in the critical pair computation. To rebind those variables and at the same time rename rules apart the following auxiliary definition is used.

Definition 7.1. An $\overline{x_k}$*-lifter* of a term t away from a set of variables V is a substitution $\sigma = \{F \mapsto \rho(F)(\overline{x_k}) \mid F \in \mathsf{fv}(t)\}$ where ρ is a injective mapping from $\mathcal{V}$ to $\mathcal{V}$ with $\rho(F) \neq F$, $\rho(F) \notin V$, and $\rho(F) : \tau_1 \to \cdots \to \tau_k \to \tau$ for $x_1 : \tau_1, \ldots x_k : \tau_k$ and $F : \tau$, for all $F \in \mathsf{fv}(t)$.

For example, consider the function symbol $\mathsf{h} : (\mathsf{o} \to \mathsf{o}) \to \mathsf{o} \to \mathsf{o}$ and the variables $F : \mathsf{o} \to \mathsf{o}$, $G : \mathsf{o}$, $J : \mathsf{o}$, $x : \mathsf{o}$, and $y : \mathsf{o}$. Then the substitution $\sigma = \{F \mapsto H(y), G \mapsto I(y)\}$ is a y-lifter of $\mathsf{h}(\lambda x.\, F(x), G)$ away from $\{F, J\}$ for variables $H : \mathsf{o} \to \mathsf{o} \to \mathsf{o}$ and $I : \mathsf{o} \to \mathsf{o}$, and mapping $\rho(F) = H$, $\rho(G) = I$.

We collect the bound variables along a position p in a term t in $\mathsf{bv}(t, p)$, i.e., $\mathsf{bv}(t, \epsilon) = \varnothing$, $\mathsf{bv}(a(t_1, \ldots, t_n), ip) = \mathsf{bv}(t_i, p)$, and $\mathsf{bv}(\lambda x.\, t, 1p) = \{x\} \cup \mathsf{bv}(t, p)$.

Definition 7.2. Let $\ell_1 \to r_1$ and $\ell_2 \to r_2$ be rewrite rules in a PRS $\mathcal{R}$ with $\mathsf{fv}(\ell_1) \cap \mathsf{bv}(\ell_1) = \varnothing$, p a position in $\mathsf{Pos}(\ell_1)$ such that $\mathsf{tp}(\ell_1|_p) \notin \mathsf{fv}(\ell_1)$. Let $\{\overline{x_k}\} = \mathsf{bv}(\ell_1, p)$ and let σ be an $\overline{x_k}$-lifter of ℓ_2 away from $\mathsf{fv}(\ell_1)$. If $\lambda\overline{x_k}.\, (\ell_1|_p)$ and $\lambda\overline{x_k}.\, (\ell_2\sigma)$ are unifiable with mgu μ, then ℓ_1 and ℓ_2 *overlap* and give rise to the *critical pair* $\ell_1\mu[r_2\sigma\mu]_p \leftarrow\!\!\rtimes\!\!\rightarrow r_1\mu$.

Example 7.3. Consider the signature consisting of the function symbols $\mathsf{f} : \mathsf{o} \to \mathsf{o} \to \mathsf{o}$ and $\mathsf{g} : (\mathsf{o} \to \mathsf{o}) \to \mathsf{o}$, the variables $F : \mathsf{o}$, $G : \mathsf{o} \to \mathsf{o}$, and $x : \mathsf{o}$, and the PRS consisting of the two rewrite rules $\mathsf{g}(\lambda x.\, \mathsf{f}(F, x)) \to F$ and $\mathsf{f}(\mathsf{g}(\lambda x.\, G(x)), F) \to F$. Then for $p = 11$ we have $\mathsf{g}(\lambda x.\, \mathsf{f}(F, x))|_p = \mathsf{f}(F, x)$ and $\mathsf{bv}(\mathsf{g}(\lambda x.\, \mathsf{f}(F, x)), p) = \{x\}$. The substitution $\sigma = \{F \mapsto H(x)\}$ is an x-lifter of $\mathsf{f}(\mathsf{g}(\lambda x.\, G(x)), F)$ away from $\{F\}$. Moreover we have that $\lambda x.\, \mathsf{f}(F, x)$ and $\lambda x.\, (\mathsf{f}(\mathsf{g}(\lambda x.\, G(x)), F))\sigma = \lambda x.\, \mathsf{f}(\mathsf{g}(\lambda x.\, G(x)), H(x))$ are unifiable with unifier $\mu = \{F \mapsto \mathsf{g}(\lambda x.\, G(x)), H \mapsto \lambda y.\, y\}$. Thus we obtain the critical pair $\mathsf{g}(\lambda x.\, x) = \mathsf{g}(\lambda x.\, F\sigma\mu) \leftarrow\!\!\rtimes\!\!\rightarrow \mathsf{g}(\lambda x.\, G(x))$.

The next lemma states that critical pairs capture peaks as desired.

Lemma 7.4. *Let $\mathcal{R}$ be a PRS and let $s \leftarrow\!\!\rtimes\!\!\rightarrow t$ be a critical pair of $\mathcal{R}$. Then there is a term u with $s \,{}_{\mathcal{R}}\!\leftarrow u \to_{\mathcal{R}} t$.*

Proof. From the definition of critical pair we obtain rewrite rules $\ell_1 \to r_1$ and $\ell_2 \to r_2$, a position p and substitutions σ and μ with $t = r_1\mu$ and $s = \ell_1\mu[r_2\sigma\mu]_p$. Define $u = \ell_1\mu$. Then $u \to r_1\mu = t$ is trivial. We also have $\lambda\overline{x_k}.\, (\ell_1|_p)\mu = (\lambda\overline{x_k}.\, (\ell_1|_p))\mu = (\lambda\overline{x_k}.\, (\ell_2\sigma))\mu = \lambda\overline{x_k}.\, (\ell_2\sigma)\mu$ and thus $\ell_1|_p\mu = \ell_2\sigma\mu$. Moreover since $\ell_1|_p$ is a pattern and $\mathsf{tp}(\ell_1|_p) \notin \mathsf{fv}(\ell_1)$ we also have $\ell_1\mu|_p = \ell_2\sigma\mu$. So in total $u = \ell_1\mu = \ell_1\mu[\ell_2\sigma\mu]_p \to \ell_1\mu[r_2\sigma\mu]_p = s$. □

Using this notion of critical pair we can state a higher-order version of the critical pair lemma.

Lemma 7.5 (Nipkow [70])**.** *A PRS $\mathcal{R}$ is locally confluent if and only if $s \downarrow_{\mathcal{R}} t$ for all critical pairs $s \leftarrow\!\rtimes\!\rightarrow t$ of $\mathcal{R}$.*

Proof Sketch. The proof structure is essentially the same as in the first-order setting: in a local peak analyze the positions of the two rewrite steps. If they are overlapping there is a critical pair, which is joinable by assumption. If they are not overlapping then they are either parallel or one step takes place in the substitution of the other. In both cases a common reduct can be constructed, see [64] for a detailed account. □

Example 7.6. Consider the PRS $\mathcal{R}$ for the map function from Example 2.50. Since $\mathcal{R}$ does not give rise to any critical pairs it is locally confluent.

Example 7.7 (Cop #426)**.** The untyped lambda calculus with β- and η-reduction can be encoded as a PRS as follows:

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

There are two critical pairs:

$$\mathsf{abs}(\lambda x.\, M(x)) \leftarrow\!\rtimes\!\rightarrow \mathsf{abs}(\lambda x.\, M(x)) \qquad \mathsf{app}(M, N) \leftarrow\!\rtimes\!\rightarrow \mathsf{app}(M, N)$$

Since they are trivially joinable we conclude local confluence.

Combining the critical pair lemma with Newman's Lemma yields a confluence criterion for PRSs.

Corollary 7.8. *A terminating PRS $\mathcal{R}$ is confluent if and only if $s \downarrow_{\mathcal{R}} t$ for all critical pairs $s \leftarrow\!\rtimes\!\rightarrow t$ of $\mathcal{R}$.* □

To make use of this criterion we need a repertoire of termination criteria. The ones supported by CSI^ho are sketched in the next section.

7.2 Termination

CSI^ho uses a basic higher-order recursive path order [88] and static dependency pairs with dependency graph decomposition and the subterm criterion [59]. Alternatively, one can also use an external termination tool like WANDA [55] as an oracle. This section gives a brief account of these three techniques.

The main ingredient of a recursive path order is a well-founded order on the function symbols, which is then lifted to a well-founded rewrite order $\succ$ on the set of terms. Hence a rewrite system is terminating if $\ell \succ r$ for all $\ell \to r$. Defining such orders for HRSs is problematic since terms are modulo β. The issue is that a relation that is closed under contexts cannot be well-founded. To see this suppose $\succ$ is closed under contexts and assume $b \succ c$. Then we have $(\lambda x.\, a) \cdot b \succ (\lambda x.\, a) \cdot c$ using closure under contexts, but also $((\lambda x.\, a) \cdot b)\updownarrow_\beta^\eta = a = ((\lambda x.\, a) \cdot c)\updownarrow_\beta^\eta$ and hence $a \succ a$. One solution is to define an order on pre-terms in η-long form, and then use $\succ \cdot \to_\beta^*$ to show termination. This approach is taken in [88] and used in CSI^ho. We briefly show the definition and sketch its use.

First note that an η-long pre-term is of the form $\lambda x_1 \ldots x_n.\, s_0(s_1, \ldots, s_n)$ where $s_0(s_1, \ldots, s_n)$ is of base type and s_i is in η-long form for all $1 \leqslant i \leqslant n$. Tailoring an order to the structure of pre-terms yields the following definition.

Definition 7.9. Assume a well-founded order on function symbols $>_\mathcal{F}$ and a partitioning of $\mathcal{F}$ into two disjoint sets $\mathcal{F}_{\mathsf{mul}}$ and $\mathcal{F}_{\mathsf{lex}}$. The *higher-order recursive path order* (HORPO) is defined as follows: we have $s \succ t$ for pre-terms $s : \sigma$ and $t : \tau$ in η-long form if $\sigma = \tau$ after collapsing all base types and one of the following clauses holds:

$$
\begin{array}{ll}
\text{(H1)} & s = f(s_1, \ldots, s_n), \\
 & s_i \succeq t \text{ for some } i \in \{1, \ldots, n\} \\
\text{(H2)} & s = f(s_1, \ldots, s_n),\ t = g(t_1, \ldots, t_m), \\
 & s \succ\!\!\succ \{t_1, \ldots, t_m\} \text{ and } f >_\mathcal{F} g \\
\text{(H3Lex)} & s = f(s_1, \ldots, s_n),\ t = f(t_1, \ldots, t_n),\ f \in \mathcal{F}_{\mathsf{lex}}, \\
 & s \succ\!\!\succ \{t_1, \ldots, t_n\} \text{ and there is an } i \in \{1, \ldots, n\} \text{ with} \\
 & s_i \succ t_i \text{ and } s_j = t_j \text{ for all } 1 \leqslant j < i \\
\text{(H3Mul)} & s = f(s_1, \ldots, s_n),\ t = f(t_1, \ldots, t_n),\ f \in \mathcal{F}_{\mathsf{mul}}, \\
 & \{s_1, \ldots, s_n\} \succ_{\mathsf{mul}} \{t_1, \ldots, t_n\}
\end{array}
$$

(H4) $s = f(s_1, \ldots, s_n),\ t = t_0(t_1, \ldots, t_m)$
$s \succ\!\!\succ \{t_0, \ldots, t_m\}$

(H5) $s = x(s_1, \ldots, s_n),\ t = x(t_1, \ldots, t_n)$
$s_i \succ t_i$ for some $i \in \{1, \ldots, n\}$ and $s_i \succeq t_i$ for all $i \in \{1, \ldots, n\}$

(H6) $s = s_0(s_1, \ldots, s_n),\ t = t_0(t_1, \ldots, t_n)$
$s_i \succ t_i$ for some $i \in \{0, \ldots, n\}$ and $s_i \succeq t_i$ for all $i \in \{0, \ldots, n\}$

(H7) $s = \lambda x.\, s_1,\ t = \lambda x.\, t_1,$
$s_1 \succ t_1$

Here $\succeq$ denotes the reflexive closure of $\succ$ and the relation $\succ\!\!\succ$ is defined as follows: $s = f(s_1, \ldots, s_n) \succ\!\!\succ \{t_1, \ldots, t_m\}$ if for all $i \in \{1, \ldots, m\}$ either $s \succ t_i$ or $s_j \succeq t_i$ for some $j \in \{1, \ldots, n\}$.

Termination can then be shown using the following result.

Theorem 7.10 (van Raamsdonk [88])**.** *A PRS $\mathcal{R}$ is terminating if for all $\ell \to r \in \mathcal{R}$ there is a pre-term r' such that $\ell \succ r' \to_\beta^* r$.* □

Example 7.11. We continue Example 7.6. The PRS for the map function can be shown terminating using HORPO: We have $\mathsf{map}(\lambda x.\, F(x), \mathsf{nil}) \succ \mathsf{nil}$ by (H1) for the first rule. For the second rule we use an intermediate pre-term, showing $\mathsf{map}(\lambda x.\, F(x), \mathsf{cons}(h, t)) \succ \mathsf{cons}((\lambda x.\, F(x))\ h, \mathsf{map}(\lambda x.\, F(x), t))$. This suffices because then we can conclude by $\mathsf{cons}((\lambda x.\, F(x))\ h, \mathsf{map}(\lambda x.\, F(x), t)) \to_\beta^* \mathsf{cons}(F(h), \mathsf{map}(\lambda x.\, F(x), t))$. Using (H2) yields $\mathsf{map} >_\mathcal{F} \mathsf{cons}$ and we are left to prove $\mathsf{map}(\lambda x.\, F(x), \mathsf{cons}(h, t)) \succ\!\!\succ \{(\lambda x.\, F(x))\ h, \mathsf{map}(\lambda x.\, F(x), t)\}$. We get $\mathsf{map}(\lambda x.\, F(x), \mathsf{cons}(h, t)) \succ (\lambda x.\, F(x))\ h$ by (H4) and (H1). To obtain $\mathsf{map}(\lambda x.\, F(x), \mathsf{cons}(h, t)) \succ \mathsf{map}(\lambda x.\, F(x), t)$ we assume $\mathsf{map} \in \mathcal{F}_{\mathsf{mul}}$, apply (H3Mul)[1] and are left to show $\{\lambda x.\, F(x), \mathsf{cons}(h, t)\} \succ_{\mathsf{mul}} \{\lambda x.\, F(x), t\}$, which we get using (H1). In total we obtain confluence by Corollary 7.8.

The intuition behind the dependency pair approach, originally due to Arts and Giesl [10] for first-order TRSs, is to identify those parts of right-hand sides of rules that may give rise to a non-terminating rewrite sequence—one may think of it as analyzing the recursive calls. It is however not obvious how the dependency pair approach can be brought into a higher-order setting. The variants that emerged can be roughly split into the dynamic approach [56, 94], which admits

[1] Choosing $\mathsf{map} \in \mathcal{F}_{\mathsf{lex}}$ and applying (H3Lex) would also work.

dependency pairs headed by functional variables, but avoids bound variables becoming free, while the static approach [59, 92, 102] only considers defined symbols, but bound variables might be freed. CSI^ho implements the static approach. Below we illustrate the technique on an example and refer to the literature for definitions and proofs. Static dependency pairs are defined just like in the first-order case: the signature is extended by marked versions of all functions symbols and left-hand sides of rules together with subterms headed by defined symbols of the corresponding right-hand side are collected as dependency pairs.

Example 7.12. Consider the following PRS $\mathcal{R}$ modeling fold right. The signature consists of

$$\begin{aligned} \mathsf{nil} &: \mathsf{natlist} \\ \mathsf{cons} &: \mathsf{nat} \to \mathsf{natlist} \to \mathsf{natlist} \\ \mathsf{foldr} &: (\mathsf{nat} \to \mathsf{nat} \to \mathsf{nat}) \to \mathsf{nat} \to \mathsf{natlist} \to \mathsf{nat} \end{aligned}$$

We have two rules in $\mathcal{R}$

$$\begin{aligned} \mathsf{foldr}(\lambda x\ y.\ F(x,y), z, \mathsf{nil}) &\to z \\ \mathsf{foldr}(\lambda x\ y.\ F(x,y), z, \mathsf{cons}(h,t)) &\to F(h, \mathsf{foldr}(\lambda x\ y.\ F(x,y), z, t)) \end{aligned}$$

which give rise to one dependency pair:

$$\mathsf{foldr}^{\#}(\lambda x\ y.\ F(x,y), z, \mathsf{cons}(h,t)) \to \mathsf{foldr}^{\#}(\lambda x\ y.\ F(x,y), z, t)$$

Since the static dependency pair method does not give rise to dependency pairs headed by functional variables, it is only applicable to systems where they occur in a harmless fashion. This class of systems is called plain-function passing and was first introduced for Simply Typed Term Rewriting Systems [60] and later extended to HRSs [59]. Intuitively plain function-passing means that free higher-order variables on the left-hand side of a rule are directly passed to the right-hand side. The PRS in Example 7.12 is plain-function passing, we refer to the literature for details.

Dependency pairs are commonly organized in the so-called dependency graph, which captures the idea that, since, for a finite system, there are only finitely many dependency pairs, in an infinite dependency chain at least one of them has to occur infinitely often. Attention can then be restricted to

the strongly connected components of this graph, which can be solved in an iterative fashion. Here one can use an order like HORPO, possibly enhanced by techniques like argument filterings and usable rules [102]. CSI^ho additionally supports the subterm criterion, originally due to Hirokawa and Middeldorp [40]. Roughly speaking, the subterm criterion permits one to delete dependency pairs, where after projecting dependency pair symbols to one of their arguments the right-hand side becomes a subterm of the left-hand side.

Example 7.13. Continuing Example 7.12 we find that by projecting $\mathsf{foldr}^{\#}$ to its third argument we obtain $\mathsf{cons}(h,t) \rhd t$. Hence the only dependency pair can be removed by the subterm criterion and $\mathcal{R}$ is terminating. Since it does not admit any critical pairs, it is confluent by Corollary 7.8.

The termination criteria described thus far are part of CSI^ho mainly so that it can be used as a standalone tool. They are subsumed by the techniques implemented in modern, dedicated higher-order termination tools like WANDA [55]. A transformation from PRSs to algebraic functional systems with meta-variables (AFSMs), the flavor of higher-order rewriting used in WANDA, is implemented in CSI^ho to make WANDA usable as an external oracle. AFSMs are simply typed, use both application as in the λ-calculus and function construction as in first-order rewriting, and have meta-variables for matching, which are variables that take arguments and come with a fixed arity. AFSMs were designed as an encompassing formalism that captures concepts from many other higher-order formalisms, see [55] for details.

Definition 7.14. For any simple type $\sigma = \sigma_1 \to \cdots \to \sigma_n \to \iota$, define a type declaration $\sigma' = [\sigma_1 \times \cdots \times \sigma_n] \to \iota$. Given a PRS $\mathcal{R}$ and a signature $\mathcal{F}$ let $\mathcal{F}' = \{f' : \sigma' \mid f : \sigma \in \mathcal{F}\}$ and choose for all variables $x : \sigma$ that occur as free variables in some rule of $\mathcal{R}$ a corresponding meta-variable $Z^x : \sigma'$. For a set of variables V define the function φ_V, which transforms PRS terms into AFSM meta-terms as follows:

$$
\begin{aligned}
\varphi_V(\lambda x.\, s) &= \lambda x.\, \varphi_V(s) && \\
\varphi_V(f\ s_1 \cdots s_n) &= f'(\varphi_V(s_1), \ldots, \varphi_V(s_n)) && \text{if } f \in \mathcal{F} \\
\varphi_V(x\ s_1 \cdots s_n) &= x\ (\varphi_V(s_1), \ldots, \varphi_V(s_n)) && \text{if } x \in \mathcal{V} \setminus V \\
\varphi_V(x\ y_1{\uparrow^\eta} \cdots y_n{\uparrow^\eta}) &= Z^x(y_1, \ldots, y_n) && \text{if } x \in V,\ y_1, \ldots, y_n \in \mathcal{V} \setminus V \\
\varphi_V(x\ s_1 \cdots s_n) &= Z^x(\varphi_V(s_1), \ldots, \varphi_V(s_n)) && \text{if } x \in V, \text{ otherwise}
\end{aligned}
$$

Define $\mathcal{R}' = \{\varphi_{\mathsf{fv}(\ell)}(\ell) \to \varphi_{\mathsf{fv}(\ell)}(r) \mid \ell \to r \in \mathcal{R}\}$.

This definition can be used to show termination of a PRS via the following theorem.

Theorem 7.15 (Kop [55])**.** *A PRS $\mathcal{R}$ over signature $\mathcal{F}$ is terminating if the AFSM $\mathcal{R}'$ is terminating over signature $\mathcal{F}'$.* □

Note that although AFSMs do allow for an application operator, transforming an application with variable left-hand side into a meta-variable of appropriate arity is essential.

Example 7.16 (Cop #764)**.** Consider the PRS $\mathcal{R}$ consisting of the following four rules:

$$\begin{aligned} \mathsf{f}(\lambda x\, y.\, F\ x) &\to \mathsf{b} & \qquad \mathsf{b} &\to \mathsf{f}(\lambda x\, y.\, \mathsf{a}) \\ \mathsf{f}(\lambda x\, y.\, \mathsf{a}) &\to \mathsf{d} & \qquad \mathsf{b} &\to \mathsf{c} \end{aligned}$$

This system is locally confluent because its four critical pairs

$$\begin{aligned} \mathsf{b} &\leftarrow\!\rtimes\!\rightarrow \mathsf{d} & \qquad \mathsf{f}(\lambda x\, y.\, \mathsf{a}) &\leftarrow\!\rtimes\!\rightarrow \mathsf{c} \\ \mathsf{d} &\leftarrow\!\rtimes\!\rightarrow \mathsf{b} & \qquad \mathsf{c} &\leftarrow\!\rtimes\!\rightarrow \mathsf{f}(\lambda x\, y.\, \mathsf{a}) \end{aligned}$$

are joinable. Note that there is a step $\mathsf{f}(\lambda x\, y.\, \mathsf{a}) \to_{\mathcal{R}} \mathsf{b}$ using the first rule, because $(\mathsf{f}(\lambda x\, y.\, F\ x)\{F \mapsto \lambda z.\, \mathsf{a}\})\!\downarrow_\beta = \mathsf{f}(\lambda x\, y.\, \mathsf{a})$. This also directly yields non-termination, because the first two rules form a loop. Indeed the system is not confluent because of the non-joinable peak

$$\mathsf{d} \leftarrow \mathsf{f}(\lambda x\, y.\, \mathsf{a}) \to \mathsf{b} \to \mathsf{c}$$

Using the translation φ from Definition 7.14 non-termination is reflected: we have $\varphi_{\{F\}}(\mathsf{f}(\lambda x\, y.\, F\ x)) = \mathsf{f}'(\lambda x\, y.\, F'(x))$ for a unary meta-variable F' and $\mathsf{f}'(\lambda x\, y.\, F'(x))$ does match $\mathsf{f}'(\lambda x\, y.\, \mathsf{a}')$ in an AFSM. Keeping the application as e.g. $\mathsf{f}'(\lambda x\, y.\, F''\ x)$ for a nullary F'', it would not match and one might mistakenly conclude termination and confluence.

7.3 Orthogonality

We now turn to non-terminating systems. Again classic criteria from the first-order setting apply, after some complications.

Definition 7.17. A left-linear PRS is *orthogonal* if admits no critical pairs. It is *weakly orthogonal* if $s = t$ for all its critical pairs $s \leftarrow\!\!\rtimes\!\!\rightarrow t$.

The PRS for map from Example 2.50 is orthogonal, the untyped lambda calculus as represented in Example 7.7 is weakly orthogonal. Compared to the first-order setting, the additional complexity is due to the possibility that residuals of a redex might be nested. Hence using parallel rewriting, like we did in Chapter 3 does not work, i.e., $\twoheadrightarrow$ does not have the diamond property for orthogonal HRSs.

Example 7.18. Consider the HRS defined by the two rules

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

Assuming a relation $\twoheadrightarrow$ for HRSs that contracts a set of parallel redexes in one go we have: $\mathsf{f}(\lambda x.\, \mathsf{g}(x)) \twoheadrightarrow \mathsf{g}(\mathsf{g}(\mathsf{a}))$ and $\mathsf{f}(\lambda x.\, \mathsf{g}(x)) \twoheadrightarrow \mathsf{f}(\lambda x.\, \mathsf{h}(x))$ using the first and second rule respectively. Since the only terms reachable from $\mathsf{g}(\mathsf{g}(\mathsf{a}))$ using $\twoheadrightarrow$ are $\mathsf{h}(\mathsf{g}(\mathsf{a}))$ and $\mathsf{g}(\mathsf{h}(\mathsf{a}))$, and the only term reachable from $\mathsf{f}(\lambda x.\, \mathsf{h}(x))$ is $\mathsf{h}(\mathsf{h}(\mathsf{a}))$, the diamond property does not hold.

The obvious solution is to use multisteps, to also allow contracting nested redex in a single step. Standard proofs then proceed by showing the diamond property or the triangle property of $\twoheadrightarrow$.

Corollary 7.19 (Nipkow [72])**.** *Orthogonal PRSs are confluent.* □

As in the first-order setting orthogonality can be relaxed to allow trivial critical pairs. Here we just state the result and refer to the literature for the (standard) proof.

Theorem 7.20 (van Oostrom and van Raamsdonk [81])**.** *Weakly orthogonal PRSs are confluent.* □

This result was further extended by van Oostrom to allow for non-trivial critical pairs that are connected by a multistep, i.e., development closed PRSs.

Theorem 7.21 (van Oostrom [78])**.** *A left-linear PRS $\mathcal{R}$ is confluent if $s \twoheadrightarrow t$ for all critical pairs $s \leftarrow\!\!\rtimes\!\!\rightarrow t$ of $\mathcal{R}$.* □

Example 7.22 (Cop #458)**.** Consider the PRS for the untyped lambda calculus from Example 7.7. If we add an additional bottom symbol of the type $\mathsf{bot} : \mathsf{term}$ and the two rules

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

we obtain two new critical pairs:

$$\mathsf{abs}(\lambda x.\, \mathsf{bot}) \leftarrow\!\rtimes\!\rightarrow \mathsf{bot} \qquad \mathsf{app}(\mathsf{bot}, M) \leftarrow\!\rtimes\!\rightarrow \mathsf{bot}$$

Since both of them are development closed we conclude confluence of the extended system by Theorem 7.21.

7.4 Modularity

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

Example 7.23 (Cop #462)**.** Consider the PRS $\mathcal{R}$ from [9] consisting of the three rules

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

The first two rules and the third rule on their own are confluent, e.g. by Corollary 7.8 and Theorem 7.20 respectively. However, because of the peak

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

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

For left-linear systems modularity does hold. One can obtain this result as a consequence a generalization of Theorem 7.20 to two PRS. We call two PRSs $\mathcal{R}$ and $\mathcal{S}$ weakly orthogonal with respect to each other, if $(\mathcal{R}\leftarrow\!\rtimes\!\rightarrow_{\mathcal{S}}) \cup (\mathcal{R}\leftarrow\!\ltimes\!\rightarrow_{\mathcal{S}}) \subseteq =$, where we use the obvious extension of critical pairs to two PRSs, simply taking one rule from each system in Definition 7.2.

Theorem 7.24 (van Oostrom and van Raamsdonk [81])**.** *Two left-linear PRSs commute if their rules are weakly orthogonal with respect to each other.* □

In combination with Lemma 2.19 this result immediately yields modularity for left-linear PRSs.

Corollary 7.25. *Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be confluent left-linear PRSs over two disjoint signatures. Then $\mathcal{R}_1 \cup \mathcal{R}_2$ is confluent.* □

7.5 Redundant Rules

The technique described in Chapter 5 directly carries over to the higher-order setting. Since rewriting is closed under contexts and substitutions the proofs need no modification.

Example 7.26. Consider the PRS from Example 7.23. After adding the redundant rule $\mathsf{f}(\mu(\lambda x.\, \mathsf{g}(x)), \mu(\lambda x.\, \mathsf{g}(x))) \to \mathsf{b}$ there is a critical pair $\mathsf{a} \leftarrow\!\!\rtimes\!\!\rightarrow \mathsf{b}$ and non-confluence is obvious.

To find new rules like the one above we again use forward closures, applying rules in both directions. In the above example unifying $Z(\mu(\lambda x.\, Z(x)))$ with $\mathsf{g}(x)$ and applying the reversed third rule to the left-hand side of the second rule yields the desired new rule. Note that since right-hand sides need not be patterns we now deal with general higher-order unification. Otherwise implementing transformations based on redundant rules for PRSs is straightforward. One just needs to take care to only add rules that do not violate the pattern restriction.

The following example illustrates removal of redundant rules.

Example 7.27 (Cop #465)**.** Consider the following encoding of lambda calculus with Regnier's σ-reduction [89]:

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

Since the left- and right-hand side of the second and third rule are convertible using the first rule, they can be removed and confluence of the first rule alone can be established by Theorem 7.20.

7.6 Summary

In this chapter we presented generalizations of confluence criteria from term rewriting, where all functions are first-order, to rewrite systems over typed λ-terms. After defining critical pairs, where one needs to take care of bound variables, we gave a critical pair lemma and discussed orthogonality and development closed critical pairs. Here the main difference to term rewriting is that using parallel rewriting does not work, due to nesting of redexes. As a divide-and-conquer technique we considered modularity, which unlike for first-order systems, is restricted to left-linear PRSs. Finally we showed how to adapt the technique of redundant rules to the higher-order setting. The results we discussed are the basis for the higher-order confluence prover CSI^ho, which we describe in the next chapter.

Chapter 8

CSI and CSI^ho

Talk is cheap. Show me the code.
Linus Torvalds

The tool CSI [66,114] is a strong automated confluence prover for first-order term rewrite systems, which has been in development since 2010 and participates in the annual confluence competition [4]. It is built on the same framework as the termination prover TTT2 [58], which is also developed in Innsbruck. Consequently when proving (relative) termination is required for a confluence criterion, a wide range of techniques is immediately available in CSI.

In this chapter we explain how to obtain and use CSI and its sibling CSI^ho, an extension of CSI for proving confluence of higher-order rewrite systems. More precisely, CSI^ho automatically checks confluence of pattern rewrite systems. This restriction is imposed to obtain decidability of unification and thus makes it possible to compute critical pairs. To this end CSI^ho implements a version of Nipkow's functional algorithm for higher-order pattern unification [71].

We also report on some distinguishing features and implementation details of the CSI family and assess its power in an experimental evaluation. Many of the techniques implemented in CSI produce proofs in CPF that can be independently verified by CeTA.

8.1 Usage

CSI is free software, licensed under the GNU LGPL. The source code, precompiled binaries, and the web-interface shown in Figure 16 are available from

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

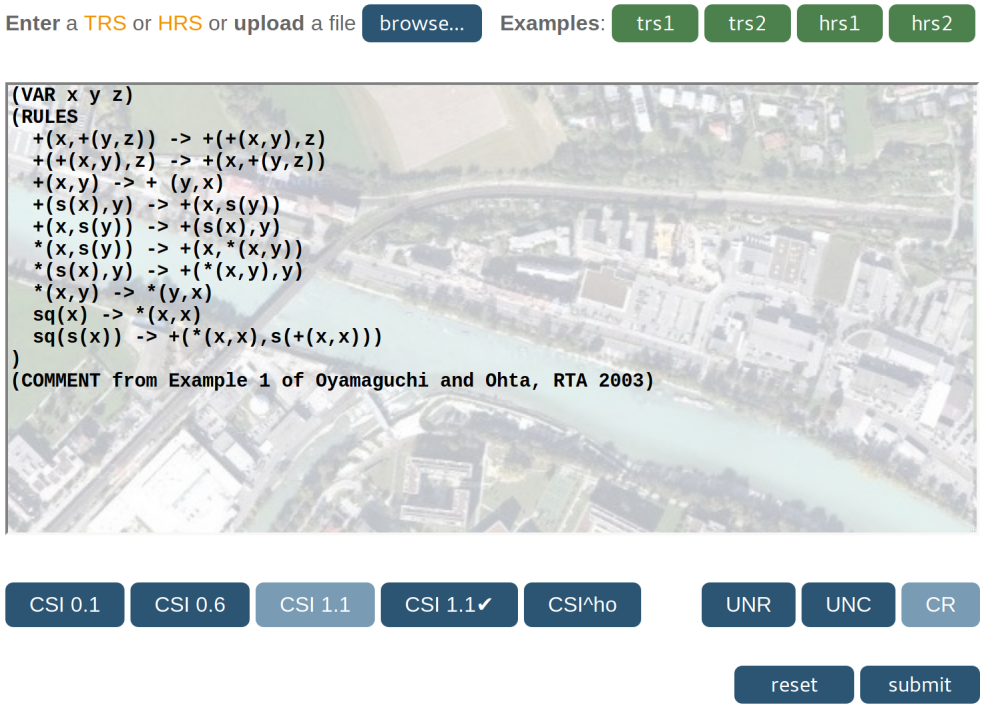


Figure 16: The web-interface of CSI and CSIˆho.

CSIˆho can be obtained from

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

and accessed through the same web-interface.

CSI reads TRSs in the classic TPDB format, the format that is also used in the Cops database. In this format the TRS from Example 3.8 for instance reads as follows:

```
(VAR x y z)
(RULES
  f(f(x, y), z) -> f(x, f(y, z))
  f(x, y) -> f (y, x)
)
```

After specifying which symbols are to be treated as variables—all other symbols

are considered to be function symbols—in a `VAR` declaration, the actual rules of the TRS are given using the keyword `RULES`.

The format for specifying PRSs for consumption by CSI^ho follows the same design, adding type declarations for variables and function symbols. The PRS for the untyped lambda calculus from Example 7.7 is given by

```
(FUN
  abs : (term -> term) -> term
  app : term -> term -> term
)
(VAR
  x : term
  N : term
  M : term -> term
)
(RULES
  app(abs(\x. M x), N) -> M(N),
  abs(\x. app(M, x)) -> M
)
```

Note that terms can be given in both applicative and algebraic notation like in `M x` and `M(N)`. Since application is denoted by juxtaposition a delimiter `,` is now required between the rules to ensure the grammar is non-ambiguous. The format itself does not require systems to be PRSs, i.e., also general HRSs can be specified. A detailed description of the Cops formats is available at

`http://coco.nue.riec.tohoku.ac.jp/problems/#format`

Given a rewrite system in one of these formats using CSI or CSI^ho via their web-interface is easy. After entering the system, select the tool to run, and the property to be shown and press submit. Besides the current versions of CSI (1.1) and CSI^ho, a configuration of CSI using only confluence criteria that can be checked by CeTA (✓CSI) and earlier releases are available.

By default CSI shows (non-)confluence (CR). Since version 1.0 also unique normal form properties are supported, namely UNR (every term has at most one normal form) and UNC (every two convertible normal forms are equal). We refer to the literature [24, 66] for details about these properties and the corresponding implementation in CSI.

For more sophisticated use cases the full power of CSI is available through its command line interface, which offers many settings and configurations. The basic invocation is

```
$ csi [options] <file> [timeout]
```

For a detailed explanation of all options, in particular the proof search strategy, one can ask CSI for help by calling it with the argument `--help`. Below we describe some of the design choices that make CSI powerful, flexible, and easy to extend.

8.2 Implementation Details

Since its first release one of CSI's defining features has been its *strategy language*, which enables the combination of techniques in a flexible manner and facilitates integration of new criteria. The basic building blocks in this language are *processors* that turn an input problem into a list of output problems that need to be solved. From these processors complex strategies are built using *strategy combinators*. Some important combinators are: sequential composition of strategies `;`, alternative composition `|` (which executes its second argument if the first fails), parallel execution `||`, and iteration `*`. A postfix `n*` executes a strategy at most `n` times while `[n]` executes its argument for at most `n` seconds. Finally `?` applies a strategy optionally (i.e., only if it makes progress), and `!` ensures that its argument only succeeds if confluence could be (dis)proved. To illustrate its power we compare the strategy used in CSI 0.1 with the one from CSI 1.1. The original strategy was

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

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

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

which illustrates how to integrate new techniques independently or in combination with others, for instance the REDUNDANT_X strategies, which are the different heuristics for finding redundant rules described in Section 5.3. The other criteria presented in this thesis are implemented in KB for Knuth-Bendix' criterion, CLOSED for strongly and almost parallel closed systems, CPCS(2) for the techniques based on critical pair closing systems, and DD for the rule labeling.

Other additions are a decision procedure for ground systems [21] (GROUND), criteria by Klein and Hirokawa [51] (KH) and by Aoto and Toyama [6] (AT), simple to test syntactic criteria by Sakai, Oyamaguchi, and Ogawa [93], and Toyama and Oyamaguchi [108] (SIMPLE), and a version of the AC critical pair lemma based on extended rules [84] (AC). The full strategy configuration grew from 76 to 233 lines since the initial release.

We remark that the price one pays for flexibility is the possibility for subtle errors. Note for instance the obscure-looking modifier {nono} applied to GROUND in the fourth line. This modifier ensures that the strategy it is applied to only succeeds if it shows confluence (and fails if it would have shown non-confluence). The modifier is necessary here, because this invocation of GROUND appears below REDUNDANT_DEL, which is the strategy that removes rules if their left- and right-hand sides are joinable using other rules, see Section 5.3. Since that transformation only reflects confluence, a potential non-confluence proof by the decision procedure implemented in GROUND needs to be discarded at that point.

Based on the same framework, CSIˆho inherits the strategy language and is configured the same way. Its strategy for proving confluence of PRSs is

```
(if prs then ((if left-linear -ho then homodular else fail)?; (
  (NOTCR | REDUNDANT_FC)3*! ||
  (REDUNDANT_DEL?;(CLOSED || KB_W))3*! ||
  (CLOSED | REDUNDANT_RHS)3*! || (CLOSED | REDUNDANT_JS)3*!
)) else fail)
```

This strategy first checks whether the input is a pattern system, and if not gives up straight away. Otherwise, if the system is left-linear, it is decomposed based on modularity as described in Section 7.4. The critical pair lemma together with the termination criteria described in Section 7.2, including a call to the external tool WANDA, is implemented in KB_W. As in CSI's configuration, NOTCR proves non-confluence, CLOSED tests for orthogonality and development closed

critical pairs, and `REDUNDANT_X` are the heuristics for adding and removing redundant rules.

8.3 Experimental Results

In this section we compare the confluence criteria described in the previous chapters, CSI's certified and full strategies, and the contestants of CoCo, via an experimental evaluation.

The experiments were carried out on a 64 bit GNU/Linux machine with an Intel® Core™ i7-5930K CPU, consisting of six cores clocked at 3.50 GHz with hyper-threading, and 32 GiB of memory.

8.3.1 Term Rewrite Systems

For the first-order setting we considered all 437 TRSs in the Cops database, version 764. For presenting the results we proceed in a top-down manner and first compare all participants of the first-order track of CoCo. Afterwards we will take a closer look at CSI, in particular the certifiable criteria from Chapters 3 and 4, the gain in power due to the redundant rules technique from Chapter 5, and the gap between CSI's certified and full strategies.

The results for the contestants of CoCo 2017 are shown in Table 1. Besides CSI 1.1, ACP 0.51[1] and CoLL-Saigawa 1.1[2] competed. A check-mark ✓ indicates that the tools certified strategy was used, i.e., the tool produces a proof in CPF, which is certified using CeTA. Figure 17 shows the examples solved by the provers in relation to each other. Here we omit the certifiable configurations: the examples solved by ✓CSI are a subset of those solved by CSI, while surprisingly ✓ACP proves non-confluence of two systems (Cops #654 and #680) for ACP answers `MAYBE`. For the 31 systems that CSI cannot handle, its main weakness is lack of special support for non-left-linear rules. Here for instance criteria based on quasi-linearity [7] and implemented in ACP are missing in CSI's repertoire. Of those 31 systems 16 are currently out of reach for all automatic confluence tools, like self-distributivity or extensions of combinatory logic.

While all non-confluence proofs produced by CSI are certifiable there is still a gap in confluence analysis. The main missing techniques are a criterion to deal

[1] `http://www.nue.riec.tohoku.ac.jp/tools/acp/`
[2] `http://www.jaist.ac.jp/project/saigawa/`

	ACP	✓ACP	CoLL-Saigawa	CSI	✓CSI
yes	225	53	142	244	148
no	155	124	99	162	162
maybe	57	260	196	31	127

Table 1: Comparison of CoCo 2017 participants.

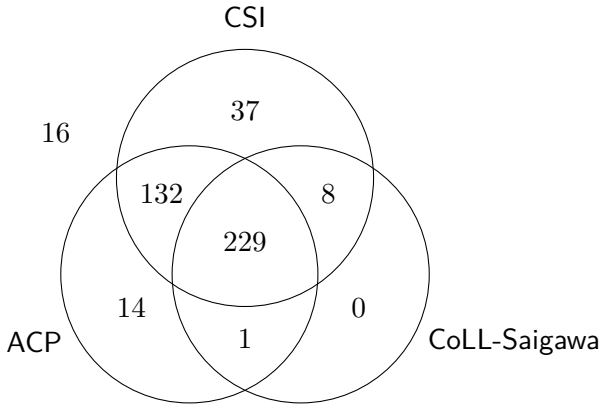


Figure 17: Overlap between solved examples for CoCo 2017 participants.

with AC rules, e.g. one based on the critical pair lemma, or the one by Aoto and Toyama [6], advanced decomposition techniques based on layer systems [25], and techniques for dealing with non-left-linear systems, in particular the criteria by Klein and Hirokawa [51] and by Sakai, Oyamaguchi, and Ogawa [93].

We now take a closer look at CSI's certifiable confluence criteria in comparison to each other. Table 2 shows the results of running CSI with different strategies that apply the direct methods described in Chapters 3 and 4. The first three columns show the results for applying just Knuth-Bendix' criterion, Corollary 3.7 and Corollary 3.23. The number of confluence proofs obtained by using critical-pair-closing systems as in Section 3.5 is shown in the fourth column. Applying the rule labeling, i.e., Lemma 4.17(e) for linear TRSs and Theorem 4.6 for left-linear TRSs, yields the results in the fifth and sixth columns. The last two columns show CSI's non-confluence methods and the combination of all listed criteria. A more detailed story for our main contributions on certification is told in Figure 18. It shows the systems solved

	KB	SC	PC	CPCS	RL-L	RL-LL	NOTCR	Σ
yes	40	57	39	62	78	83	0	106
no	0	0	0	0	0	0	154	154
maybe	397	380	398	375	359	354	283	177

Table 2: Experimental results for certifiable criteria in CSI.

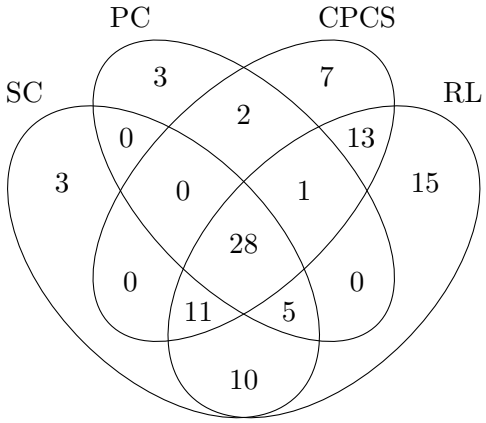


Figure 18: Overlap between solved Cops for four confluence criteria.

by Corollary 3.7 (SC), Corollary 3.23 (PC), critical-pair-closing systems (CPCS) and Theorem 4.6 (RL) in relation to each other. We omit Lemma 4.17(e), since it is subsumed by Theorem 4.6. Overall the decreasing diagrams based rule labeling technique turned out stronger than the other, simpler, mostly syntactic criteria. On our test-bed Theorem 4.6 can establish confluence of as many systems as the other three methods combined (83 systems), and by itself shows confluence of more than 75 % of all systems that are certifiably confluent by CSI's direct methods (106 vs. 83). However, this power comes at a cost: the certificates for confluence proofs by rule labeling are usually larger and much harder to check for a human, making automatic, reliable certification a necessity. The largest certificate (for Cop #60) has 769 KiB and lists 182 candidate joins for showing the 34 critical peaks decreasing. CeTA checks this certificate in 0.3 s. We remark that no confluence tool besides CSI has solved Cop #60 so far, stressing the importance of a certified proof.

	KB	SC	PC	CPCS	RL-L	RL-LL	NOTCR	Σ (✓CSI)
yes	73	91	84	86	99	106	0	148
no	0	0	0	0	0	0	162	162
maybe	364	346	353	351	338	331	275	127

Table 3: Experimental results for certifiable criteria in combination with the redundant rules technique.

	$\mathsf{CSI}_{\mathsf{nrr}}$	$\mathsf{CSI}_{\mathsf{js}}$	$\mathsf{CSI}_{\mathsf{rhs}}$	$\mathsf{CSI}_{\mathsf{fc}}$	$\mathsf{CSI}_{\mathsf{del}}$	CSI
yes	228	227	229	225	238	244
no	154	157	155	162	154	162
maybe	55	53	53	50	45	31
	✓$\mathsf{CSI}_{\mathsf{nrr}}$	✓$\mathsf{CSI}_{\mathsf{js}}$	✓$\mathsf{CSI}_{\mathsf{rhs}}$	✓$\mathsf{CSI}_{\mathsf{fc}}$	✓$\mathsf{CSI}_{\mathsf{del}}$	✓CSI
yes	106	124	116	112	113	148
no	154	157	155	162	154	162
maybe	177	156	166	163	170	127

Table 4: Experimental results for different redundant rules heuristics.

That Knuth-Bendix' criterion is the weakest of the listed techniques is easily explained by the composition of Cops. It simply contains few terminating systems. Since showing confluence is easy after termination has been established, and in order to avoid skewing the database towards termination, they are often deemed of little interest for Cops. The, possibly surprising, power of strong closedness can be similarly explained. Of the 437 TRSs 286 are linear, while only 53 are left-linear but not right-linear.

Finally we evaluate the redundant rules technique from Chapter 5. The impact of redundant rules on each of the direct methods is shown in Table 3. Comparing these numbers to the results from Table 2 reveals that all evaluated criteria benefit heavily, for instance the power of parallel closedness more than doubles. To assess the power of the different heuristics described in Section 5.3, we consider each of them in combination with CSI's full and certified strategies. The results are shown in Table 4.

For the full strategy, adding joining sequences of critical pairs (js) or rewriting

right-hand sides (rhs) show limited effect: (js) gains 3 non-confluence proofs and 2 confluence proofs where it also loses 3; (rhs) gains 5 systems for confluence, while losing 4, and results in 1 additional non-confluence proof. As expected forward closures (fc) turn out strongest for non-confluence, gaining 8 proofs, but weak for confluence, losing 4 proofs, while adding only 1. Removing rules (del) is the most effective technique overall and gains 12 systems while losing 2 other ones. With all heuristics combined, 24 new systems can be shown (non-)confluent.

For the certifiable strategy the results are the same for non-confluence, but the picture for confluence is a bit different. Here, (rhs) gains 10 proofs, (js) gains 18 systems, (fc) gains 6, and (del) gains 21 proofs while losing 14. Remarkably, in 15 of these 21 proofs the TRS becomes orthogonal after removing redundant rules, which emphasizes that our transformations can considerably simplify confluence proofs. In total, 50 new systems are shown (non-)confluent. Note that not only does the redundant rules technique significantly increase the number of certifiable confluence proofs, the resulting certificates can be virtually impossible to check by hand, due to their size. The largest proof produced in our experiments starts from a TRS with five rules, Cop #141, and extends it with 50 redundant rules before showing confluence using a rule labeling—the certificate in CPF has 14 MiB.

8.3.2 Higher-Order Rewrite Systems

For experiments in the higher-order setting we again used Cops version 764, which contains 79 PRSs. We again start by comparing the participants of CoCo 2017. Other than CSIˆho 0.3, the contestants were SOL 1c_2017[3] and ACPH 0.02, a higher-order extension of ACP. The results are shown in Table 5.

Note that ACPH reports some false negatives (Cops # 728, 729, and 730). This is known since CoCo 2017 and the three systems are already removed from ACPH's results and listed separately, in the row labeled "erroneous". On the current test-bed CSIˆho solves all systems where any of the tools produces a proof. Solving the remaining 11 systems will require serious effort—they contain e.g. lambda calculus with surjective pairing and self-distributivity of explicit substitution.

The most powerful of CSIˆho's criteria is development closedness, which on

[3] https://www.cs.gunma-u.ac.jp/~hamana/

	ACPH	CSI^ho	SOL
yes	48	55	54
no	10	13	11
maybe	18	11	14
erroneous	3	0	0

Table 5: Comparison of CoCo 2017 participants for PRSs.

its own shows confluence of 43 systems, 37 of which are weakly orthogonal. The critical pair lemma for terminating systems yields confluence proofs for 29 systems. For 3 of those proofs using WANDA as external termination prover is essential. Redundant rules show limited effect on the PRSs in Cops. Removing redundant rules is used in 3 of the proofs produced by CSI^ho, while adding redundant rules adds one additional proof of non-confluence, namely for the system from Example 7.26.

We conclude by remarking that the possibility for subtle errors, like misconfigurations in the proof search strategy, the complexity of the generated proofs, e.g. for the rule labeling, and `YES-NO` conflicts between tools illustrate that certification is imperative in order to guarantee reliable automatic confluence analysis.

Chapter 9

Conclusion

So beschließen beide denn
nach so manchem Doch und Wenn
sich mit ihren Theorien
vor die Wissenschaft zu knien.
Christian Morgenstern (Alle Galgenlieder)

In the preceding chapters, we have given an overview of our most prominent contributions to IsaFoR/CeTA and CSI/CSI^ho. After discussing preliminaries on first- and higher-order rewriting in Chapter 2, we described our formalization of classic confluence criteria based on restricted joinability of critical pairs and a more modern approach based on critical-pair-closing systems in Chapter 3. We generalized all results in Chapter 3 to the setting of commutation. Together with our formalization of the rule labeling from Chapter 4 this covers all direct confluence criteria that are currently available in IsaFoR. Chapter 5 was devoted to the simple, yet widely applicable and surprisingly powerful technique of redundant rules, which not only further increased the power of CeTA, but also of the full strategy of CSI. We then switched to higher-order rewriting and gave a short, simple, formal proof for confluence of the λ-calculus in Chapter 6, before describing the theory behind our higher-order confluence prover CSI^ho in Chapter 7. Both CSI and CSI^ho were discussed in detail in Chapter 8, where we also gave an extensive experimental evaluation.

9.1 Related Work

While we covered CeTA's full repertoire for confluence in this thesis, it is also a powerful certifier for non-confluence proofs. CeTA can check that, given derivations $s \to^* t_1$ and $s \to^* t_2$, t_1 and t_2 cannot be joined. Here the supported justifications are:

- testing that t_1 and t_2 are distinct normal forms,
- testing that $\mathsf{tcap}(t_1\sigma)$ and $\mathsf{tcap}(t_2\sigma)$ are not unifiable [114],
- usable rules, discrimination pairs, argument filters, and interpretations [3], and
- reachability analysis using tree automata [27].

Formalizing confluence criteria has a long history in the λ-calculus. The first mechanically verified proof of the Church-Rosser property of β-reduction was done using the Boyer-Moore theorem prover [95]. Pfenning's formalization in Elf [85] was later used to formalize conservativity of extensional lambda calculus with surjective pairing [101]. Huet [46] proved a stronger variant of the parallel moves lemma in Coq. Nipkow [73] used Isabelle/HOL to prove the Church-Rosser property of β, η, and $\beta\eta$. For β-reduction the standard Tait/Martin-Löf proof as well as Takahashi's proof [104] were formalized.

Newman's lemma and the critical pair lemma (for first-order rewrite systems) have been formalized using ACL2 [91]. An alternative proof of the latter in PVS, following the structure of Huet's proof, is presented in [30]. PVS is also used in a formalization of the lemmas of Newman and Yokouchi [29]. Knuth and Bendix' criterion has also been formalized in Coq [18] and Isabelle/HOL [98].

Our work on redundant rules draws a lot of inspiration from existing literature. One starting point is [80], where van Oostrom introduces the notion of *feeble* orthogonality. A TRS is feebly orthogonal if the critical peaks arising from its non-redundant* rules are trivial or contain a trivial step (that rewrites a term to itself); a rule is redundant* if it can be simulated by another rule in a single step. Clearly our notion of redundancy generalizes redundancy*.

The most important prior work concerning redundant rules is [6]. In this paper, Aoto and Toyama describe an automated confluence criterion, implemented in ACP, based on decomposing TRSs into a reversible part $\mathcal{P}$ and a terminating part $\mathcal{S}$. In order to help applicability of their criterion, they introduce a procedure based on the inference rules

$$\text{replace}\quad \frac{\langle \mathcal{S} \cup \{\ell \to r\}, \mathcal{P}\rangle}{\langle \mathcal{S} \cup \{\ell \to r'\}, \mathcal{P}\rangle}\quad r \leftrightarrow_{\mathcal{P}}^* r' \qquad \text{add}\quad \frac{\langle \mathcal{S}, \mathcal{P}\rangle}{\langle \mathcal{S} \cup \{\ell \to r\}, \mathcal{P}\rangle}\quad \ell \leftrightarrow_{\mathcal{P}}^* \cdot \to_{\mathcal{S}}^* r$$

The key is that because $\mathcal{P}$ is reversible, $\leftrightarrow_{\mathcal{P}}^*$ and $\to_{\mathcal{P}}^*$ coincide, and therefore confluence of $\mathcal{S} \cup \mathcal{P}$ is not affected by applying these inference rules. This very

same idea underlies Lemma 5.3, which establishes *reduction equivalence*, and thus Corollary 5.4. Note that no rule removal is performed in [6].

There is a second connection between our work and [6] that seems noteworthy. Given a reversible $\mathcal{P}$, every rule from $\mathcal{P}^{-1}$ can be simulated by a sequence of $\mathcal{P}$-steps. Therefore, confluence of $\mathcal{S} \cup \mathcal{P}$ and $\mathcal{S} \cup \mathcal{P} \cup \mathcal{P}^{-1}$ coincide by Corollary 5.4. Using this observation, one could decompose the confluence criteria of [6] into two steps, one that replaces $\mathcal{P}$ by $\mathcal{P} \cup \mathcal{P}^{-1}$, and a respective underlying confluence criterion that does not make use of reversibility, but instead demands that $\mathcal{P}$ is symmetric, i.e., $\mathcal{P}^{-1} \subseteq \mathcal{P}$.

The idea of showing confluence by removing rules whose sides are convertible has already been used in the literature, e.g. [42, Example 11], which is a variation of Example 4.7.

Other works of interest are [33, 116], where Gramlich and Zantema apply a similar idea to Corollary 5.4 to termination: If some additional requirements are met, then termination of $\mathcal{R} \cup \{\ell \to r\}$ is equivalent to termination of $\mathcal{R} \cup \{\ell \to r'\}$ where $r \to_{\mathcal{R}} r'$ by a non-erasing rule. This is true for non-overlapping TRSs [33, Theorem 4], or when the rule used in the $r \to_{\mathcal{R}} r'$ step is locally confluent by itself, left-linear, and furthermore it doesn't overlap with any rules from $\mathcal{R} \cup \{\ell \to r\}$ except itself [116, Theorem 4].

While for first-order term rewriting basic concepts are well agreed-on, higher-order rewriting comes in many different flavors. The first definition of a higher-order rewriting formalism are Aczel's contraction schemes, defined in 1978 [1]. His definition extends first-order rewriting with binders and meta-variables, which allows higher-order functions to be defined. Extending this idea Klop defined combinatory reduction systems in 1980 in his thesis [52]. The first formalisms to use types were introduced in 1991, namely the higher-order rewrite systems by Nipkow [70] described in this thesis and algebraic functional systems by Jouannaud and Okada [48]. Other formalisms introduced in the 1990s include the expression reduction systems of Khasidashvili [50], Wolfram's higher-order term rewriting systems [112], interaction systems by Laneve [61], combinatory reduction systems with eXtensions by Rose [90], abstract data type systems defined by Jouannaud and Okada [49] and inductive data type systems by Blanqui [15].

Due to the many different formalisms, the study of higher-order rewriting is a heterogeneous field. The termination competition for instance uses algebraic functional systems as their format of choice, and consequently most work on termination happened for that formalism. The only formalization effort

for higher-order rewriting we are aware of is a formalization of HORPO for algebraic functional systems in Coq [57].

An interesting recent development was sparked by an extension of the class of higher-order patterns, where unification is still decidable [62]. This result is the basis of the new higher-order tool SOL [37], which can analyze confluence of a larger class of systems than CSIˆho. Moreover this extension allows for a higher-order completion procedure. For PRSs, while technically possible, completion almost always fails immediately, since right-hand sides of rules are usually not patterns, making it impossible to orient newly deduced equations as rewrite rules.

9.2 Future Work

While we could increase the number of certifiable confluence proofs significantly over the last few years, further efforts are needed in order to reach the full power of automatic confluence tools. In particular criteria for dealing with systems containing AC rules are needed to close the gap.

To strengthen confluence tools further, automating the Z-property would benefit both first- and higher-order provers, in order to deal with rules like self-distributivity.

Recently there is interest in automating properties related to confluence as witnessed by the categories on unique normalization in CoCo 2017. More work will be needed to obtain provers for these properties that are as powerful as state-of-the-art confluence tools. An easy first step could be to use the redundant rules technique. For instance, extending a TRS $\mathcal{R}$ with a rule $\ell \to r$ for which we have $\ell \to^+_{\mathcal{R}} r$ affects neither the set of normal forms nor reachability.

Finally, formalization is needed for higher-order rewriting. Not just for certification, but to leverage the synergy between rewriting techniques and proof assistants. Often they use rewriting for equational reasoning, Isabelle/HOL for instance uses conditional higher-order rewriting. Here the choice what theorems to use as rewrite rules is usually left to the user, which means that there are no guarantees about confluence or termination of these systems.

Bibliography

[1] P. Aczel. A general Curch-Rosser theorem. Unpublished Manuscript, University of Manchester, 1978.

[2] T. Aoto. Automated confluence proof by decreasing diagrams based on rule-labelling. In *Proc. 21st International Conference on Rewriting Techniques and Applications*, volume 6 of *Leibniz International Proceedings in Informatics*, pages 7–16, 2010. doi: 10.4230/LIPIcs.RTA.2010.7.

[3] T. Aoto. Disproving confluence of term rewriting systems by interpretation and ordering. In *Proc. 9th International Workshop on Frontiers of Combining Systems*, volume 8152 of *Lecture Notes in Artificial Intelligence*, pages 311–326, 2013. doi: 10.1007/978-3-642-40885-4_22.

[4] T. Aoto, N. Hirokawa, J. Nagele, N. Nishida, and H. Zankl. Confluence Competition 2015. In *Proc. 25th International Conference on Automated Deduction*, volume 9195 of *Lecture Notes in Artificial Intelligence*, pages 101–104, 2015. doi: 10.1007/978-3-319-21401-6_5.

[5] T. Aoto and Y. Toyama. Persistency of confluence. *Journal of Universal Computer Science*, 3(11):1134–1147, 1997. doi: 10.3217/jucs-003-11-1134.

[6] T. Aoto and Y. Toyama. A reduction-preserving completion for proving confluence of non-terminating term rewriting systems. *Logical Methods in Computer Science*, 8(1:31):1–29, 2012. doi: 10.2168/LMCS-8(1:31)2012.

[7] T. Aoto, Y. Toyama, and K. Uchida. Proving confluence of term rewriting systems via persistency and decreasing diagrams. In *Proc. Joint 25th International Conference on Rewriting Techniques and Applications and 12th International Conference on Typed Lambda Calculi and Applications*, volume 8560 of *Lecture Notes in Computer Science (Advanced Research in Computing and Software Science)*, pages 46–60, 2014. doi: 10.1007/978-3-319-08918-8_4.

[8] T. Aoto, J. Yoshida, and Y. Toyama. Proving confluence of term rewriting systems automatically. In *Proc. 20th International Conference on Rewriting Techniques and Applications*, volume 5595 of *Lecture Notes in Computer Science*, pages 93–102, 2009. doi: 10.1007/978-3-642-02348-4_7.

[9] C. Appel, V. van Oostrom, and J. G. Simonsen. Higher-order (non-)modularity. In *Proc. 21st International Conference on Rewriting Techniques and Applications*, volume 6 of *Leibniz International Proceedings in Informatics*, pages 17–32, 2010. doi: 10.4230/LIPIcs.RTA.2010.17.

[10] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000. doi: 10.1016/S0304-3975(99)00207-8.

[11] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998. doi: 10.1017/CBO9781139172752.

[12] H. P. Barendregt. *The Lambda Calculus, Its Syntax and Semantics*. North-Holland, 2nd edition, 1984.

[13] H. P. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Oxford University Press, 1992.

[14] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development; Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer, 2004.

[15] F. Blanqui. Termination and confluence of higher-order rewrite systems. In *Proc. 11th International Conference on Rewriting Techniques and Applications*, volume 1833 of *Lecture Notes in Computer Science*, pages 47–61, 2000. doi: 10.1007/10721975_4.

[16] F. Blanqui and A. Koprowski. CoLoR, a Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates. *Mathematical Structures in Computer Science*, 21(4):827–859, 2011. doi: 10.1017/S0960129511000120.

[17] E. Contejean, P. Courtieu, J. Forest, O. Pons, and X. Urbain. Automated certified proofs with CiME3. In *Proc. 22nd International Conference on Rewriting Techniques and Applications*, volume 10 of *Leibniz International Proceedings in Informatics*, pages 21–30, 2011. doi: 10.4230/LIPIcs.RTA.2011.21.

[18] E. Contejean, P. Courtieu, J. Forest, O. Pons, and X. Urbain. Automated certified proofs with CiME3. In *Proc. 22nd International Conference on Rewriting Techniques and Applications*, volume 10 of *Leibniz International Proceedings in Informatics*, pages 21–30, 2011. doi: 10.4230/LIPIcs.RTA.2011.21.

[19] P. Dehornoy. *Braids and Self-Distributivity*, volume 192 of *Progress in Mathematics*. Springer, 2000. doi: 10.1007/978-3-0348-8442-6.

[20] P. Dehornoy and V. van Oostrom. Z, proving confluence by monotonic single-step upperbound functions. Presentation at International Conference *Logical Models of Reasoning and Computation*, August 2008.

[21] B. Felgenhauer. Deciding confluence of ground term rewrite systems in cubic time. In *Proc. 23rd International Conference on Rewriting Techniques and Applications*, volume 15 of *Leibniz International Proceedings in Informatics*, pages 165–175, 2012. doi: 10.4230/LIPIcs.RTA.2012.165.

[22] B. Felgenhauer. A proof order for decreasing diagrams. In *Proc. 1st International Workshop on Confluence*, pages 7–14, 2012.

[23] B. Felgenhauer. Decreasing diagrams II. *Archive of Formal Proofs*, Aug. 2015. Formal proof development, http://www.isa-afp.org/entries/Decreasing-Diagrams-II.shtml.

[24] B. Felgenhauer. Efficiently deciding uniqueness of normal forms and unique normalization for ground TRSs. In *Proc. 5th International Workshop on Confluence*, pages 16–20, 2016. Available from http://cl-informatik.uibk.ac.at/iwc/2016.php.

[25] B. Felgenhauer, A. Middeldorp, H. Zankl, and V. van Oostrom. Layer systems for proving confluence. *ACM Transactions on Computational Logic*, 16(2:14):1–32, 2015. doi: 10.1145/2710017.

[26] B. Felgenhauer, J. Nagele, V. van Oostrom, and C. Sternagel. The Z property. *Archive of Formal Proofs*, June 2016. Formal proof development, `https://www.isa-afp.org/entries/Rewriting_Z.shtml`.

[27] B. Felgenhauer and R. Thiemann. Reachability, confluence, and termination analysis with state-compatible automata. *Information and Computation*, 253(3):467–483, 2017. doi: `10.1016/j.ic.2016.06.011`.

[28] B. Felgenhauer and V. van Oostrom. Proof orders for decreasing diagrams. In *Proc. 24th International Conference on Rewriting Techniques and Applications*, volume 21 of *Leibniz International Proceedings in Informatics*, pages 174–189, 2013. doi: `10.4230/LIPIcs.RTA.2013.174`.

[29] A. Galdino and M. Ayala-Rincón. A formalization of Newman's and Yokouchi's lemmas in a higher-order language. *Journal of Formalized Reasoning*, 1(1):39–50, 2008. doi: `10.6092/issn.1972-5787/1347`.

[30] A. Galdino and M. Ayala-Rincón. A formalization of the Knuth-Bendix(-Huet) critical pair theorem. *Journal of Automated Reasoning*, 45(3):301–325, 2010. doi: `10.1007/s10817-010-9165-2`.

[31] A. Geser, A. Middeldorp, E. Ohlebusch, and H. Zantema. Relative undecidability in term rewriting: II. The confluence hierarchy. *Information and Computation*, 178(1):132–148, 2002. doi: `10.1006/inco.2002.3150`.

[32] W. D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13(2):225–230, 1981. doi: `10.1016/0304-3975(81)90040-2`.

[33] B. Gramlich. Simplifying termination proofs for rewrite systems by preprocessing. In *Proc. 2nd ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, pages 139–150, 2000. doi: `10.1145/351268.351286`.

[34] B. Gramlich and S. Lucas. Generalizing Newman's lemma for left-linear rewrite systems. In *Proc. 17th International Conference on Rewriting Techniques and Applications*, volume 4098 of *Lecture Notes in Computer Science*, pages 66–80, 2006. doi: `10.1007/11805618_6`.

[35] F. Haftmann. *Code Generation from Specifications in Higher Order Logic*. Dissertation, Technische Universität München, München, 2009. urn:nbn:de:bvb:91-diss-20091208-886023-1-1.

[36] F. Haftmann and T. Nipkow. Code generation via higher-order rewrite systems. In *Proc. 10th International Symposium on Functional and Logic Programming*, volume 6009 of *Lecture Notes in Computer Science*, pages 103–117, 2010. doi: 10.1007/978-3-642-12251-4_9.

[37] M. Hamana. How to prove your calculus is decidable: Practical applications of second-order algebraic theories and computation. *Proc. of the ACM on Programming Languages*, 1(ICFP):22:1–22:28, 2017. doi: 10.1145/3110266.

[38] J. Hindley. *The Church-Rosser Property and a Result in Combinatory Logic*. PhD thesis, University of Newcastle-upon-Tyne, 1964.

[39] N. Hirokawa. Commutation and signature extensions. In *Proc. 4th International Workshop on Confluence*, pages 23–27, 2015.

[40] N. Hirokawa and A. Middeldorp. Tyrolean termination tool: Techniques and features. *Information and Computation*, 205(4):474–511, 2007. doi: 10.1016/j.ic.2006.08.010.

[41] N. Hirokawa and A. Middeldorp. Decreasing diagrams and relative termination. In *Proc. 5th International Joint Conference on Automated Reasoning*, volume 6173 of *Lecture Notes in Artificial Intelligence*, pages 487–501, 2010. doi: 10.1007/978-3-642-14203-1_41.

[42] N. Hirokawa and A. Middeldorp. Decreasing diagrams and relative termination. *Journal of Automated Reasoning*, 47(4):481–501, 2011. doi: 10.1007/s10817-011-9238-x.

[43] N. Hirokawa and A. Middeldorp. Commutation via relative termination. In *Proc. 2nd International Workshop on Confluence*, pages 29–33, 2013.

[44] N. Hirokawa, A. Middeldorp, S. Winkler, and C. Sternagel. Infinite runs in abstract completion. In *Proc. 2nd International Conference on Formal Structures for Computation and Deduction*, volume 84 of *Leibniz International Proceedings in Informatics*, pages 19:1–19:16, 2017. doi: 10.4230/LIPIcs.FSCD.2017.19.

[45] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980. doi: 10.1145/322217.322230.

[46] G. Huet. Residual theory in λ-calculus: A formal development. *Journal of Functional Programming*, 4(3):371–394, 1994. doi: 10.1017/S0956796800001106.

[47] G. Huet and B. Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Informatica*, 11(1):31–55, 1978. doi: 10.1007/BF00264598.

[48] J.-P. Jouannaud and M. Okada. A computation model for executable higher-order algebraic specification languages. In *Proc. 6th IEEE Symposium on Logic in Computer Science*, pages 350–361, 1991. doi: 10.1109/LICS.1991.151659.

[49] J.-P. Jouannaud and M. Okada. Abstract data type systems. *Theoretical Computer Science*, 173(2):349–391, 1997. doi: 10.1016/S0304-3975(96)00161-2.

[50] Z. Khasidashvili. Expression reduction systems. Technical Report 36, Vekua Institute of Applied Mathematics of Tbilisi State University, 1990.

[51] D. Klein and N. Hirokawa. Confluence of non-left-linear TRSs via relative termination. In *Proc. 18th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 7180 of *Lecture Notes in Computer Science (Advanced Research in Computing and Software Science)*, pages 258–273, 2012. doi: 10.1007/978-3-642-28717-6_21.

[52] J. Klop. *Combinatory Reduction Systems*. PhD thesis, Utrecht University, 1980.

[53] J. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: Introduction and survey. *Theoretical Computer Science*, 121(1-2):279–308, 1993. doi: 10.1016/0304-3975(93)90091-7.

[54] D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970. doi: 10.1016/B978-0-08-012975-4.50028-X.

[55] C. Kop. *Higher Order Termination*. PhD thesis, Vrije Universiteit Amsterdam, 2012.

[56] C. Kop and F. van Raamsdonk. Dynamic dependency pairs for algebraic functional systems. *Logical Methods in Computer Science*, 8(2:10):1–51, 2012. doi: 10.2168/LMCS-8(2:10)2012.

[57] A. Koprowski. *Termination of Rewriting and Its Certification*. PhD thesis, Eindhoven University of Technology, 2008.

[58] M. Korp, C. Sternagel, H. Zankl, and A. Middeldorp. Tyrolean Termination Tool 2. In *Proc. 20th International Conference on Rewriting Techniques and Applications*, volume 5595 of *Lecture Notes in Computer Science*, pages 295–304, 2009. doi: 10.1007/978-3-642-02348-4_21.

[59] K. Kusakari, Y. Isogai, M. Sakai, and F. Blanqui. Static dependency pair method based on strong computability for higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E92-D(10):2007–2015, 2009. doi: 10.1587/transinf.E92.D.2007.

[60] K. Kusakari and M. Sakai. Enhancing dependency pair method using strong computability in simply-typed term rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 18(5):407–431, 2007. doi: 10.1007/s00200-007-0046-9.

[61] C. Laneve. *Optimality and Concurrency in Interaction Systems*. PhD thesis, Università di Pisa, 1993.

[62] T. Libal and D. Miller. Functions-as-constructors higher-order unification. In *Proc. 1st International Conference on Formal Structures for Computation and Deduction*, volume 52 of *Leibniz International Proceedings in Informatics*, pages 26:1–26:17, 2016. doi: 10.4230/LIPIcs.FSCD.2016.26.

[63] D. Matichuk, T. Murray, and M. Wenzel. Eisbach: A proof method language for Isabelle. *Journal of Automated Reasoning*, 56(3):261–282, 2016. doi: 10.1007/s10817-015-9360-2.

[64] R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(1):3–29, 1998. doi: 10.1016/S0304-3975(97)00143-6.

[65] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991. doi: 10.1093/logcom/1.4.497.

[66] J. Nagele, B. Felgenhauer, and A. Middeldorp. CSI: New evidence – A progress report. In *Proc. 26th International Conference on Automated Deduction*, volume 10395 of *Lecture Notes in Artificial Intelligence*, pages 385–397, 2017. doi: 10.1007/978-3-319-63046-5_24.

[67] J. Nagele and R. Thiemann. Certification of confluence proofs using CeTA. In *Proc. 3rd International Workshop on Confluence*, pages 19–23, 2014.

[68] J. Nagele and H. Zankl. Certified rule labeling. In *Proc. 26th International Conference on Rewriting Techniques and Applications*, volume 36 of *Leibniz International Proceedings in Informatics*, pages 269–284, 2015. doi: 10.4230/LIPIcs.RTA.2015.269.

[69] M. Newman. On theories with a combinatorial definition of equivalence. *Annals of Mathematics*, 43(2):223–243, 1942.

[70] T. Nipkow. Higher-order critical pairs. In *Proc. 6th IEEE Symposium on Logic in Computer Science*, pages 342–349, 1991. doi: 10.1109/LICS.1991.151658.

[71] T. Nipkow. Functional unification of higher-order patterns. In *Proc. 8th IEEE Symposium on Logic in Computer Science*, pages 64–74, 1993. doi: 10.1109/LICS.1993.287599.

[72] T. Nipkow. Orthogonal higher-order rewrite systems are confluent. In *Proc. 1st International Conference on Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 306–317, 1993. doi: 10.1007/BFb0037114.

[73] T. Nipkow. More Church-Rosser proofs. *Journal of Automated Reasoning*, 26(1):51–66, 2001. doi: 10.1023/A:1006496715975.

[74] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002. doi: 10.1007/3-540-45949-9.

[75] S. Okui. Simultaneous critical pairs and Church-Rosser property. In *Proc. 9th International Conference on Rewriting Techniques and Applications*, volume 1379 of *Lecture Notes in Computer Science*, pages 2–16, 1998. doi: 10.1007/BFb0052357.

[76] V. van Oostrom. Confluence by decreasing diagrams. *Theoretical Computer Science*, 126(2):259–280, 1994. doi: 10.1016/0304-3975(92)00023-K.

[77] V. van Oostrom. *Confluence for Abstract and Higher-Order Rewriting*. PhD thesis, Vrije Universiteit Amsterdam, 1994.

[78] V. van Oostrom. Developing developments. *Theoretical Computer Science*, 175(1):159–181, 1997. doi: 10.1016/S0304-3975(96)00173-9.

[79] V. van Oostrom. Confluence by decreasing diagrams – converted. In *Proc. 19th International Conference on Rewriting Techniques and Applications*, volume 5117 of *Lecture Notes in Computer Science*, pages 306–320, 2008. doi: 10.1007/978-3-540-70590-1_21.

[80] V. van Oostrom. Feebly not weakly. In *Proc. 7th International Workshop on Higher-Order Rewriting*, Vienna Summer of Logic flash drive, 2014.

[81] V. van Oostrom and F. van Raamsdonk. Weak orthogonality implies confluence: The higher order case. In *Proc. 3rd International Symposium on Logical Foundations of Computer Science*, volume 813 of *Lecture Notes in Computer Science*, pages 379–392, 1994. doi: 10.1007/3-540-58140-5_35.

[82] M. Oyamaguchi and N. Hirokawa. Confluence and critical-pair-closing systems. In *Proc. 3rd International Workshop on Confluence*, pages 29–33, 2014.

[83] M. Oyamaguchi and Y. Ohta. On the Church-Rosser property of left-linear term rewriting systems. *IEICE Transactions on Information and Systems*, E86-D(1):131–135, 2003. http://search.ieice.org/bin/summary.php?id=e86-d_1_131.

[84] G. E. Peterson and M. E. Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM*, 28(2):233–264, 1981. doi: 10.1145/322248.322251.

[85] F. Pfenning. A proof of the Church-Rosser theorem and its representation in a logical framework. Technical Report CMU-CS-92-186, School of Computer Science, Carnegie Mellon University, 1992.

[86] Z. Qian. Unification of higher-order patterns in linear time and space. *Journal of Logic and Computation*, 6(3):315–341, 1996. doi: 10.1093/logcom/6.3.315.

[87] F. van Raamsdonk. *Confluence and Normalisation for Higher-Order Rewriting*. PhD thesis, Vrije Universiteit Amsterdam, 1996.

[88] F. van Raamsdonk. On termination of higher-order rewriting. In *Proc. 12th International Conference on Rewriting Techniques and Applications*, volume 2051 of *Lecture Notes in Computer Science*, pages 261–275, 2001. doi: 10.1007/3-540-45127-7_20.

[89] L. Regnier. Une équivalence sur les lambda-termes. *Theoretical Computer Science*, 126(2):281–292, 1994. doi: 10.1016/0304-3975(94)90012-4.

[90] K. H. Rose. *Operational Reduction Models for Functional Programming Languages*. PhD thesis, University of Copenhagen, 1996.

[91] J.-L. Ruiz-Reina, J.-A. Alonso, M.-J. Hidalgo, and F.-J. Martín-Mateos. Formal proofs about rewriting using ACL2. *Annals of Mathematics and Artificial Intelligence*, 36(3):239–262, 2002. doi: 10.1023/A:1016003314081.

[92] M. Sakai and K. Kusakari. On dependency pair method for proving termination of higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E88-D(3):583–593, 2005. doi: 10.1093/ietisy/e88-d.3.583.

[93] M. Sakai, M. Oyamaguchi, and M. Ogawa. Non-E-overlapping, weakly shallow, and non-collapsing TRSs are confluent. In *Proc. 25th International Conference on Automated Deduction*, volume 9195 of *Lecture Notes in Artificial Intelligence*, pages 111–126, 2015. doi: 10.1007/978-3-319-21401-6_7.

[94] M. Sakai, Y. Watanabe, and T. Sakabe. An extension of the dependency pair method for proving termination of higher-order rewrite systems.

IEICE Transactions on Information and Systems, E84-D(8):1025–1032, 2001.

[95] N. Shankar. A mechanical proof of the Church-Rosser theorem. *Journal of the ACM*, 35(3):475–522, 1988. doi: 10.1145/44483.44484.

[96] K. Shintani and N. Hirokawa. CoLL: A confuence tool for left-linear term rewrite systems. In *Proc. 25th International Conference on Automated Deduction*, volume 9195 of *Lecture Notes in Artificial Intelligence*, pages 127–136, 2015. doi: 10.1007/978-3-319-21401-6_8.

[97] R. Statman. The typed lambda-calculus is not elementary recursive. *Theoretical Computer Science*, 9(1):73–81, 1979. doi: 10.1016/0304-3975(79)90007-0.

[98] C. Sternagel and R. Thiemann. Formalizing Knuth-Bendix orders and Knuth-Bendix completion. In *Proc. 24th International Conference on Rewriting Techniques and Applications*, volume 21 of *Leibniz International Proceedings in Informatics*, pages 287–302, 2013. doi: 10.4230/LIPIcs.RTA.2013.287.

[99] C. Sternagel and R. Thiemann. The certification problem format. In *Proc. 11th International Workshop on User Interfaces for Theorem Provers*, volume 167 of *Electronic Proceedings in Theoretical Computer Science*, pages 61–72, 2014. doi: 10.4204/EPTCS.167.8.

[100] C. Stirling. Decidability of higher-order matching. *Logical Methods in Computer Science*, 5(3:2):1–52, 2009. doi: 10.2168/LMCS-5(3:2)2009.

[101] K. Støvring. Extending the extensional lambda calculus with surjective pairing is conservative. *Logical Methods in Computer Science*, 2(2:1):1–14, 2006. doi: 10.2168/LMCS-2(2:1)2006.

[102] S. Suzuki, K. Kusakari, and F. Blanqui. Argument filterings and usable rules in higher-order rewrite systems. *IPSJ Online Transactions*, 4(2):1–12, 2011. doi: 10.2197/ipsjtrans.4.114.

[103] T. Suzuki, T. Aoto, and Y. Toyama. Confluence proofs of term rewriting systems based on persistency. *Computer Software*, 30(3):148–162, 2013. in Japanese. doi: 10.11309/jssst.30.3_148.

[104] M. Takahashi. Parallel reductions in λ-calculus. *Information and Computation*, 118(1):120–127, 1995. doi: 10.1006/inco.1995.1057.

[105] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

[106] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *Proc. 22nd International Conference on Theorem Proving in Higher Order Logics*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468, 2009. doi: 10.1007/978-3-642-03359-9_31.

[107] 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, 1988.

[108] Y. Toyama and M. Oyamaguchi. Church-Rosser property and unique normal form property of non-duplicating term rewriting systems. In *Proc. 4th International Workshop on Conditional and Typed Rewriting Systems*, pages 316–331, 1995. doi: 10.1007/3-540-60381-6_19.

[109] C. Urban and C. Kaliszyk. General bindings and alpha-equivalence in Nominal Isabelle. *Logical Methods in Computer Science*, 8(2:14):1–35, 2012. doi: 10.2168/LMCS-8(2:14)2012.

[110] J. B. Wells, D. Plump, and F. Kamareddine. Diagrams for meaning preservation. In *Proc. 14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *Lecture Notes in Computer Science*, pages 88–106, 2003. doi: 10.1007/3-540-44881-0_8.

[111] T. Wierzbicki. Complexity of the higher order matching. In *Proc. 16th International Conference on Automated Deduction*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 82–96, 1999. doi: 10.1007/3-540-48660-7_6.

[112] D. A. Wolfram. Rewriting, and equational unification: the higher-order cases. In *Proc. 4th International Conference on Rewriting Techniques and Applications*, volume 488 of *Lecture Notes in Computer Science*, pages 25–36, 1991. doi: 10.1007/3-540-53904-2_83.

[113] H. Zankl. Confluence by decreasing diagrams – formalized. In *Proc. 24th International Conference on Rewriting Techniques and Applications*, volume 21 of *Leibniz International Proceedings in Informatics*, pages 352–367, 2013. doi: 10.4230/LIPIcs.RTA.2013.352.

[114] H. Zankl, B. Felgenhauer, and A. Middeldorp. CSI – A confluence tool. In *Proc. 23rd International Conference on Automated Deduction*, volume 6803 of *Lecture Notes in Artificial Intelligence*, pages 499–505, 2011. doi: 10.1007/978-3-642-22438-6_38.

[115] H. Zankl, B. Felgenhauer, and A. Middeldorp. Labelings for decreasing diagrams. *Journal of Automated Reasoning*, 54(2):101–133, 2015. doi: 10.1007/s10817-014-9316-y.

[116] H. Zantema. Reducing right-hand sides for termination. In *Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of his 60th Birthday*, volume 3838 of *Lecture Notes in Computer Science*, pages 173–197, 2005. doi: 10.1007/11601548_12.

Index