

Cumulative Habilitation Thesis

Certified Term Rewriting

Christian Sternagel

September 30, 2019

für Mikro, Emma und Mona

Contents

I. Preface	9
1. Introduction	11
2. Contributions	15
2.1. Code Generation (Chapter 3)	15
2.2. Certification (Chapter 5)	16
2.3. Well-Quasi-Order Theory (Chapter 4)	18
2.4. Completion (Chapters 7 and 8)	18
2.5. Confluence (Chapter 6)	20
2.6. Further Contributions	21
II. Selected Publications	25
3. A Mechanized Proof of GHC's Mergesort	27
3.1. Introduction	27
3.2. GHC's Sorting Algorithm	29
3.3. Preliminaries	31
3.3.1. Correctness	32
3.3.2. Stability	33
3.3.3. Goal	33
3.4. Efficient Mergesort	34
3.4.1. Correctness	34
3.4.2. Stability	35
3.4.3. Complexity	36
3.5. Conclusion and Related Work	38
4. Certified Kruskal's Tree Theorem	43
4.1. Introduction	43
4.2. Preliminaries	46
4.3. Homogeneous Sequences	47
4.4. Dickson's Lemma	48
4.5. Minimal Bad Sequences	49
4.6. Higman's Lemma	52
4.7. The Tree Theorem	53
4.8. Examples	56

4.9. Conclusion and Related Work	58
5. A Framework for Developing Stand-Alone Certifiers	59
5.1. Introduction	59
5.2. Certification	60
5.2.1. Human Inspection	61
5.2.2. Certification via Programs	61
5.2.3. Certification via Proof Assistants	62
5.2.4. Certification via Programs and Proof Assistants	64
5.3. Error Handling	66
5.4. Readable Error Messages	68
5.5. Parsing	70
5.5.1. A Parser from Strings to XML	71
5.5.2. A Library for Parsing XML	72
5.6. Soundness	73
5.7. Conclusion	77
6. Certified Confluence of Conditional Rewriting	79
6.1. Introduction	79
6.2. Preliminaries	83
6.3. Roadmap of Formalized Methods	86
6.4. Orthogonality	88
6.4.1. Certification	93
6.5. A Critical Pair Criterion	94
6.5.1. Certification	98
6.5.2. Certification Challenges	101
6.5.3. Check Functions	103
6.6. Finding Witnesses for Non-Confluence of CTRSs	105
6.6.1. Implementation	107
6.6.2. Certification	108
6.7. Infeasibility of Conditional Critical Pairs	108
6.7.1. Unification	109
6.7.2. Symbol Transition Graph	111
6.7.3. Decomposing Reachability Problems	112
6.7.4. Exact Tree Automata Completion	115
6.7.5. Exploiting Equalities	120
6.7.6. Certification	121
6.8. Supporting Methods	123
6.8.1. Infeasible Rule Removal	124
6.8.2. Inlining of Conditions	125
6.8.3. Certification and Implementation	126
6.9. Experiments	127
6.9.1. Comparing ConCon's Confluence Methods	127
6.9.2. Comparing ConCon's Non-Confluence Methods	129

6.9.3. Comparing ConCon's Infeasibility Methods	130
6.10. Conclusion	131
6.10.1. Formalization and Implementation	132
6.10.2. Future Work	133
6.10.3. Related Work	134
7. Abstract Completion, Formalized	135
7.1. Introduction	135
7.2. Preliminaries	138
7.2.1. Rewrite Systems	138
7.2.2. Abstract Confluence Criteria	140
7.2.3. Critical Peaks	142
7.3. Correctness for Finite Runs	145
7.4. Canonicity and Normalization Equivalence	150
7.5. Ground Completion	154
7.6. Correctness for Infinite Runs	158
7.7. Ordered Completion	162
7.8. Completeness Results for Ordered Completion	171
7.8.1. Ground-Total Orders	171
7.8.2. Linear Systems	176
7.9. Conclusion	180
8. Certified Equational Reasoning via Ordered Completion	183
8.1. Introduction	183
8.2. Preliminaries	185
8.3. Formalized Ordered Completion	186
8.4. Formalized Ground Joinability Criteria	188
8.4.1. A Simple Criterion	188
8.4.2. Ground Joinability via Order Closures	189
8.5. Applications	193
8.6. Certification	194
8.7. Experiments	197
8.8. Conclusion	198
Bibliography	199

Part I.

Preface

1. Introduction

Two of the most important properties in *program verification* are *termination* and *confluence*. The former guarantees that we always obtain a result eventually (e.g., essential for device drivers), while the latter allows for smooth reasoning in the presence of nondeterminism (e.g., parallel execution of code). Thus, automatable formal methods ensuring that a program satisfies either of these properties are of great interest. The most successful approaches are based on general models of computation which capture a variety of programming languages and paradigms (like C, Haskell, Java, Prolog, etc.; ranging from imperative, to functional, to object oriented, and logic programming).

Arguably, the most widely adopted model of computation for proving termination and confluence is *term rewriting* (as demonstrated by the annual international [termination](#) and [confluence competitions](#)).^{1,2} Another cornerstone of rewriting is the related topic of *completion*, where the goal is to “complete” a given set of equations into an equivalent term rewrite system (TRS) that is terminating and confluent, giving rise to a decision procedure for the induced equational theory. While conceptually simple—based on the slogan “replace equals by equals”—term rewriting comes with a formidable collection of results concerning termination, confluence, and completion. These results are implemented in powerful automated tools, so that to date we have a variety of termination provers [1, 26, 44, 72, 73, 123, 169, 181], confluence provers [5, 46, 54, 94, 104, 113, 126, 155, 184], and completion tools [69, 159, 170, 178] for term rewrite systems at our disposal.

On the one hand, the goal of program verification is to prove that a program adheres to its specification. On the other hand, automated program verification tools are programs themselves, and pretty **complex** ones at that. Moreover, they are **tuned for efficiency** using **sophisticated data structures** and often have very **short release cycles** facilitating the quick integration of new techniques. So, why should we trust such tools? The short answer is: we should not! There is a long history of bugs in the field. And these are unavoidable, given the sheer complexity of tools.

What is more, the same reasons that make such tools complex, also make them tremendously hard to verify. And even after managing such an arduous task, the gain would just be a single verified tool. What about later versions of the tool, all the other useful automated tools in the same application area? Verifying every single one of them simply does not scale.

A more viable alternative is *certification via proof assistants*, also called *the certification approach*. *Proof assistants*—like Coq [18], HOL4 [127], and Isabelle [172]—are computer programs that allow a user to interactively compose rigorous machine-checked mathematical proofs. This usually means that a proof has to be completely reduced to

¹http://termination-portal.org/wiki/Termination_Competition

²<http://project-coco.uibk.ac.at>

1. Introduction

primitive inferences using the basic axioms of the employed logic alone. Doing so by hand would lead to an unusable system. Therefore, modern proof assistants provide powerful automated proof tools. But those often only take hold after the user has either given enough hints or broken down the proof into sufficiently small intermediate steps. While the recent advent of *hammers* [19]—i.e., first-order automated theorem provers whose results are rigorously reduced to primitive inferences in a proof reconstruction phase inside a proof assistant—yields a tremendous boost in productivity, proving mathematical theorems in this way can still be laborious. In return, we gain near 100% reliability.

A powerful feature of many proof assistants (supported by at least Coq, HOL4, and Isabelle) is *code generation*. Code generation enables the automatic synthesis of a fully verified program from a suitable mechanization. E.g., Isabelle’s code generator [52] allows us to transform a purely equational subset of higher-order logic (HOL), first into an intermediate functional language whose semantics is given by higher-order rewriting, and in a further step into actual source code for several functional programming languages (at the moment Standard ML, Haskell, OCaml, and Scala). In this way it is possible to first rigorously prove results in Isabelle, then use them to prove correctness of a function also specified in Isabelle, and finally code-generate this function to obtain an actual program whose partial correctness is guaranteed by construction.

For *certification*, instead of requiring a tool to be verifiably correct, we only demand that it justifies each of its claims (e.g., that a given TRS is terminating) by a formal *certificate*. This is then complemented by a *certifier*—an automated tool that is able to rigorously validate a given certificate. As formal judge of assertions that are made by independent tools, a certifier’s impeccable authority is essential. In the following, I present the predominant architecture of such certifiers for term rewriting, which has proven to be powerful (e.g., covering more than 90% of all proofs of this year’s *termination competition*³ and around 90% of the proofs of this year’s *confluence competition*⁴) and useful (by uncovering bugs in tools that have in some cases gone unnoticed for years). So far, the most rigorous way of obtaining such a certifier is by a two-phase approach, employing proof assistants (where “mechanize” is short for “formalize with a proof assistant”):

- (1) Use the proof assistant in order to **mechanize the underlying theory**.
- (2) Mechanize a **verified executable check function** for certificates.

These two phases correspond to the left part of Figure 1.1. On its right, we can see the resulting “certification pipeline,” consisting of an untrusted tool for producing a certificate (a new certificate for each input, hence the dashed arrow), together with a trusted certifier for validation. The certifier is obtained by the code generation facility of the proof assistant, granting partial correctness in the sense that whenever a certificate is accepted, we may conclude that it is correct.

Concrete realizations of this general approach are Coccinelle/CiME [26] (as part of the A3PAT project), CoLoR/Rainbow [20], and IsaFoR/CeTA [165]—which I started for

³http://www.termination-portal.org/wiki/Termination_Competition_2019

⁴<http://project-coco.uibk.ac.at/2019/>

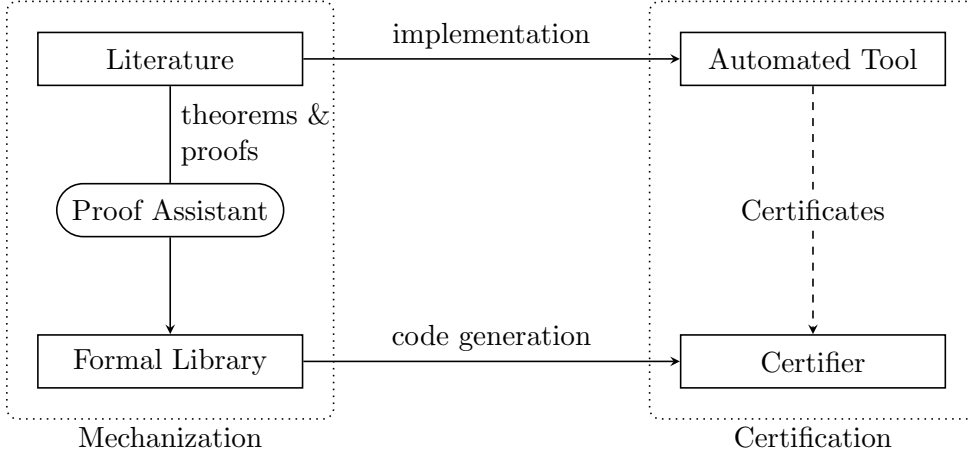


Figure 1.1.: The certification approach

my PhD. The former two use Coq for phase (1), while *CeTA* is code-generated [52] from our *Isabelle Formalization of Rewriting*, which is a formal Isabelle library that contains many results on rewriting (more specifically, most techniques that are employed by modern termination tools as well as most of the first seven chapters of the textbook *Term Rewriting and All That* by Baader and Nipkow [9]). The library and the certifier are available at <http://cl-informatik.uibk.ac.at/isafor>, providing also an overview of the supported techniques and the accompanying literature [6, 59, 76, 88, 95, 97–99, 129, 135, 137–142, 146, 147, 149, 150, 154, 164, 165, 177, 182, 183].

Until recently, certification was restricted to confluence and termination of plain first-order rewriting, where it is adopted by a significant number of tools. With this thesis, my goals are threefold: (1) to make code generation of certifiers easier and improve the performance of generated code in general, (2) to formalize selected topics of the meta-theory and theory of term rewriting and its variants (like the theory of simplification orders via *Kruskal’s Tree Theorem*, the theory of completion for plain and ordered rewriting, and results on conditional term rewriting), and (3) to firmly establish the certification approach also for conditional term rewriting; more specifically, to enable certification of confluence proofs for conditional term rewrite systems.

The remainder is organized as follows: First, in Chapter 2, I summarize my contributions (furthermore, contributions that are not part of this thesis are listed). Then, in Part II, I present my selected publications.

2. Contributions

With this habilitation thesis I contribute to the *certification approach* in general and its application to (conditional) term rewriting in particular. More specifically, our contributions are as follows. First, regarding code generation of check functions for certification:

- I give a formally verified natural mergesort implementation that can be used for code generation without further ado (Chapter 3).
- Moreover, we provide a general framework for developing verified certifiers that takes away a lot of the work that is typically required when applying the *certification approach* in specific cases (Chapter 5).

And then, in the area of term rewriting we focus on the following three areas:

- mechanizing part of the meta-theory of term rewriting in form of Kruskal’s tree theorem (Chapter 4),
- certified confluence of conditional term rewriting (Chapter 6), and
- formalizing the theory and certification of (ordered) completion (Chapters 7 and 8).

In the remainder, publications that are typeset in gray are either auxiliary results like Isabelle/HOL theories in the *Archive of Formal Proofs*¹ (AFP, for short) or conference publications that—while relevant to a topic—are subsumed by later journal articles.

In the following I discuss my contributions in chronological order of the main corresponding publications.

2.1. Code Generation (Chapter 3)

Publications

- [1] Christian Sternagel. Efficient Mergesort. *The Archive of Formal Proofs*, 2011
[afp:Efficient-Mergesort](#)
- [2] Christian Sternagel. Proof Pearl—A Mechanized Proof of GHC’s Mergesort. *Journal of Automated Reasoning*, 51(4):357–370, 2013
[doi:10.1007/s10817-012-9260-7](#)

¹<https://www.isa-afp.org/>

Synopsis and Contributions

In [2] I give an Isabelle/HOL formalization of the natural mergesort algorithm that is for example part of the library of the *Glasgow Haskell Compiler* (the de facto standard for Haskell programs). In addition of its correctness I also prove mergesort to be a stable sorting algorithm. In combination correctness and stability allow me to replace Isabelle’s standard sorting algorithm (a variant of quicksort for immutable lists instead of arrays) by my natural mergesort implementation. Natural mergesort performs far better on immutable lists than quicksort and is especially well suited for (partially) (reverse) sorted lists without sacrificing performance on random lists. As a result every Isabelle user who imports my formalization—which is freely available from the *Archive of Formal Proofs*, see [1]—enjoys an immediate performance boost in generated code.

My contributions were to first narrow a performance issue in CeTA down to sorting, then identify natural mergesort as the most viable alternative sorting algorithm for our intended target language Haskell, and finally formalize its correctness and stability in Isabelle/HOL. Moreover, I took this opportunity to update Chapter 3 in comparison to [2] by a formalized complexity proof and rerun all experiments on more modern hardware using the latest Isabelle version.

Related Work

Variants of mergesort were formalized in other systems before: the first such formalization I am aware of was in Coq,² another one in ACL2.³ However, both of them do not formalize *natural mergesort* with its initial separation into (reverse) sorted sublists which is quite a bit more complex than other mergesort variants that start from singleton lists. Moreover, these earlier formalizations do not consider stability.

Other sorting algorithms were formalized in various systems, like insertion sort, quicksort, and heapsort in Coq [38]; and insertion sort and quicksort in Isabelle/HOL (as part of its standard library).

It seems that my work spawned renewed interest in the formalization of sorting algorithms, as evidenced by: De Gouw, De Boer, and Rot formalize counting sort and radix sort using the KeY system [27]. Then, Leino and Lucio formalize natural mergesort using the verifier Dafny [79]. Later, De Gouw et al., through formalization, identified and fixed a series flaw in OpenJDK’s Timsort algorithm [28, 29]. Another formalization of Timsort, this time for C, is given by Zhang, Zhao, and Sanan [185].

2.2. Certification (Chapter 5)

Publications

- [3] Christian Sternagel and René Thiemann. A Framework for Developing Stand-Alone Certifiers. *Electronic Notes in Theoretical Computer Science* 312:51–67, 2014

²<http://coq.inria.fr/stdlib/Coq.Sorting.Mergesort.html>

³<https://github.com/acl2/acl2/blob/master/books/powerlists/merge-sort.lisp>

[doi:10.1016/j.entcs.2015.04.004](https://doi.org/10.1016/j.entcs.2015.04.004)

- [4] Christian Sternagel and René Thiemann. Certification Monads. *The Archive of Formal Proofs*, 2014
[afp:Certification_Monads](#)
- [5] Christian Sternagel and René Thiemann. Haskell’s Show Class in Isabelle/HOL. *The Archive of Formal Proofs*, 2014
[afp:Show](#)
- [6] Christian Sternagel and René Thiemann. XML. *The Archive of Formal Proofs*, 2014
[afp:XML](#)

Synopsis and Contributions

In [3] we provide three general components that are useful in any formally verified certifier in order to: (1) write check functions that validate proofs from a certificate, (2) produce readable error messages in case of a faulty certificate, and (3) parse XML data from the real world—the actual certificate—into the realm of proof assistants.

To achieve (1) we provide several flavors of error monads together with various primitive check functions and commonly useful combinators that allow us to create more complex check functions from simpler ones [4]. These monads incorporate error messages that are composed in the style of Haskell’s `Show` class [5] and thereby enable (2). This setup is rounded off by a formalization of XML trees as Isabelle/HOL data type together with a parser from strings into that data type and a collection of transformation functions and combinators that facilitate the translation of XML trees into user-defined data types [6], enabling (3).

My specific contributions were as follows: I came up with the idea to employ monads and their well-known properties in the construction of check functions, resulting in a reusable framework; moreover, I formalized most of the port of Haskell’s `Show` class into Isabelle/HOL; and finally, I initiated the use of XML transformers and started with their implementation and formalization.

Related Work

The first instance of what we call the certification approach—using the proof assistant Coq—I am aware of, is a tree automata completion checker due to Boyer, Genet, and Jensen [23]. Two other projects—both based on Coq—on certification of term rewriting related properties that were initiated approximately at the same time as our *IsaFoR/CeTA* project are *CoLoR/Rainbow* [20] and *Coccinelle/CiME* [26] (as part of the *A3PAT* project).

In other areas like certification of proofs generated by SAT solvers, verified certifiers are also gaining momentum: the first such tool is given by Lammich [78].

2.3. Well-Quasi-Order Theory (Chapter 4)

Publications

- [7] Christian Sternagel. Well-Quasi-Orders. *The Archive of Formal Proofs*, 2012
[afp:Well_Quasi_Orders](#)
- [8] Christian Sternagel. Certified Kruskal’s Tree Theorem. In *Proceedings of the 3rd International Conference on Certified Programs and Proofs (CPP)*, volume 8307 of *Lecture Notes in Computer Science*, pages 178–193, Springer, 2013
[doi:10.1007/978-3-319-03545-1_12](#)
- [9] Christian Sternagel. Certified Kruskal’s Tree Theorem. *Journal of Formalized Reasoning*, 7(1):45–62, 2014
[doi:10.6092/issn.1972-5787/4213](#)

Synopsis and Contributions

In [9] I give Isabelle/HOL formalizations of *Dickson’s Lemma*, *Higman’s Lemma*, and *Kruskal’s Tree Theorem*—where the formalized proofs of the latter two are based on Nash-William’s minimal bad sequence argument. These results are all part of well-quasi-order theory and allow us to extend well-quasi-orders on base sets to well-quasi-orders over pairs, lists (or equivalently, finite sequences), and trees (or equivalently, first-order terms), respectively. A well-known application of well-quasi-orders is for example to prove well-foundedness of simplification orders [31, 32].

My contribution was the first formalization of *Kruskal’s Tree Theorem* using a proof assistant and its application to prove well-foundedness of a Knuth-Bendix order that is part of *IsaFoR* [142]. I also formalized some other results that are part of well-quasi-order theory [7] and worked on an alternative proof of *Higman’s Lemma* by *open induction* [110].

Related Work

While *Higman’s Lemma* has been formalized inside a proof assistant before—for example by Berghofer in Isabelle/HOL [16], and by Martín-Mateos et al. in ACL2 [84]—I am not aware of any previous formalizations of *Kruskal’s Tree Theorem*.

My formalization has also been used by others. For example by Wu, Zhang, and Urban to formalize the *Myhill-Nerode Theorem* for regular expressions [180] and by Nagele, Felgenhauer, and Zankl to certify confluence proofs in term rewriting [99].

2.4. Completion (Chapters 7 and 8)

Publications

- [10] Nao Hirokawa, Aart Middeldorp, Christian Sternagel. A New and Formalized Proof of Abstract Completion. In *Proceedings of the 5th International Conference on Interactive Theorem Proving*, volume 8558 of *Lecture Notes in Computer Science*,

pages 292–307, Springer, 2014

[doi:10.1007/978-3-319-08970-6_19](https://doi.org/10.1007/978-3-319-08970-6_19)

- [11] Nao Hirokawa, Aart Middeldorp, Christian Sternagel, and Sarah Winkler. Infinite Runs in Abstract Completion. In *Proceedings of the 2nd International Conference on Formal Structures in Computation and Deduction*, volume 84 of *Leibniz International Proceedings in Informatics*, pages 19:1–19:16, Schloss Dagstuhl, 2017
[doi:10.4230/LIPIcs.FSCD.2017.19](https://doi.org/10.4230/LIPIcs.FSCD.2017.19)
- [12] Nao Hirokawa, Aart Middeldorp, Christian Sternagel, and Sarah Winkler. Abstract Completion, Formalized. *Logical Methods in Computer Science*, 15(3):19:1–19:42, 2019
[doi:10.23638/LMCS-15\(3:19\)2019](https://doi.org/10.23638/LMCS-15(3:19)2019)
- [13] Christian Sternagel and Sarah Winkler. Certified Equational Reasoning via Ordered Completion. In *Proceedings of the 27th International Conference on Automated Deduction*, volume 11716 of *Lecture Notes in Computer Science*, pages 508–528, Springer, 2019
[doi:10.1007/978-3-030-29436-6_30](https://doi.org/10.1007/978-3-030-29436-6_30)

Synopsis and Contributions

In [12] we formalize many results on Knuth-Bendix completion and ordered completion—a variant of completion based on ordered rewriting [83]—that are known from the literature and further extend them. To this end, we do not just formalize existing proofs, but provide new proofs that are not based on the traditional proof orders and thereby allow for more separation into independent results that are reusable in different contexts. After realizing that, at least for ordered rewriting, the abstract results from [12] are not enough for certification, we developed an independent formalization of ordered completion in [13] that is geared towards certification by incorporating variable renamings already in the abstract inference system.

My specific contributions are as follows: I formalized all of the results on abstract completion from [13] that are not concerned with ordered rewriting, like prime critical pair criteria, completeness, ground completion, etc. This was not a formalization of existing proofs but more an interaction between myself and my ingenious co-authors. They had new ideas on how to prove something and I immediately put them to test inside my ongoing formalization. In that way we were able to catch some not so obvious errors early on and achieve what I think is a good compromise between complexity and readability of proofs. Concerning certification of ordered completion I formalized basic results on ordered rewriting and the completion inference system.

Related Work

Knuth and Bendix originally introduced completion [71], a procedure that aims at obtaining a terminating and confluent term rewrite system for a given set of equations. If completion succeeds, we therefore can decide arbitrary equations in the theory induced by the initial set of equations.

2. Contributions

Later, Bachmair, Dershowitz, and Hsiang devised an abstract inference system for completion [10, 12, 13] and established *proof orders* as de facto standard for showing the correctness of completion inference systems.

For certification of a successful completion run we do not require all the restrictions that are needed in order to prove correctness of the original inference system, an observation that was made and formalized by me and Thiemann [142]. In fact, in this setting, we can certify the result of completion without ever mentioning an inference system. However, for ordered completion the situation is more involved and we are not aware of any way to certify a successful ordered completion run without specifying an inference system.

2.5. Confluence (Chapter 6)

Publications

- [14] Christian Sternagel and Thomas Sternagel. Certifying Confluence of Quasi-Decreasing Strongly Deterministic Conditional Term Rewrite Systems. In *Proceedings of the 26th International Conference on Automated Deduction*, volume 10395 of *Lecture Notes in Computer Science*, pages 413–431, Springer, 2017
[doi:10.1007/978-3-319-63046-5_26](https://doi.org/10.1007/978-3-319-63046-5_26)
- [15] Christian Sternagel and Thomas Sternagel. Certifying Confluence of Almost Orthogonal CTRSs via Exact Tree Automata Completion. In *Proceedings of the 1st International Conference on Formal Structures for Computation and Deduction*, volume 52 of *Leibniz International Proceedings in Informatics*, pages 29:1–29:16, Schloss Dagstuhl, 2016
[doi:10.4230/LIPIcs.FSCD.2016.29](https://doi.org/10.4230/LIPIcs.FSCD.2016.29)
- [16] Christian Sternagel and Thomas Sternagel. Certified Confluence of Conditional Rewriting. *Journal of Automated Reasoning*, 2019
submitted

Synopsis and Contributions

In [16] we describe an *IsaFoR*-based formalization of results concerned with confluence of conditional term rewriting, culminating in a certifier that allows us to check most confluence techniques for conditional term rewrite systems (CTRSs) that are employed during the international confluence competition.

With our work we bring certification in the CTRS category of the confluence competition on par with the state of the art for termination and confluence of plain rewriting.

The techniques we formalize can basically be grouped into the following categories: critical pair criteria (that allow us to conclude confluence for a restricted class of CTRSs), transformational techniques (that “reduce” confluence of a given CTRS to confluence of an “easier” CTRS), and methods for proving infeasibility of conditional critical pairs.

My specific contributions are as follows: I invented and formalized the inlining technique described in Chapter 6 (Lemma 6.8.6) and a criterion that facilitates the removal of

infeasible rules from a given CTRSs (Theorem 6.8.4). I formalized the critical pair criterion for almost orthogonality modulo infeasibility and the requisite check functions that allow for its certification. I also developed a locale based Isabelle/HOL formalization of permutations/renamings that allows us to talk about entities like terms, rules, and TRSs modulo renaming of variables in a general way. Concerning exact tree automata completion for infeasibility, I implemented and formalized the check functions that provide the link between certificates and abstract results on tree automata.

Related Work

Recently the notion of infeasibility (basically reachability between to “template” terms that may be specialized by arbitrary substitutions) received interest apart from proving confluence of CTRSs as witnessed by the new INF category of the confluence competition.⁴

While there are new approaches for proving infeasibility, like the work by Lucas and Gutiérrez [80] and by Sternagel and Yamada [153], I am not aware of any other work that is formalized using a proof assistant and thereby would enable certification.

2.6. Further Contributions

I have co-authored the following further publications since I received my PhD (listed in chronological order).

- [17] Christian Sternagel and René Thiemann. Modular and Certified Semantic Labeling and Unlabeling. In *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications*, volume 10 of *Leibniz International Proceedings in Informatics*, pages 329–344, Schloss Dagstuhl, 2011
[doi:10.4230/LIPIcs.RTA.2011.329](https://doi.org/10.4230/LIPIcs.RTA.2011.329)
- [18] Alexander Krauss, Christian Sternagel, René Thiemann, Carsten Fuhs, and Jürgen Giesl. Termination of Isabelle Functions via Termination of Rewriting. In *Proceedings of the 2nd International Conference on Interactive Theorem Proving*, volume 6898 of *Lecture Notes in Computer Science*, pages 152–167, Springer, 2011
[doi:10.1007/978-3-642-22863-6_13](https://doi.org/10.1007/978-3-642-22863-6_13)
- [19] Christian Sternagel and René Thiemann. Generalized and Formalized Uncurrying. In *Proceedings of the 8th International Symposium on Frontiers of Combining Systems*, volume 6989 of *Lecture Notes in Artificial Intelligence*, pages 243–258, Springer, 2011
[doi:10.1007/978-3-642-24364-6_17](https://doi.org/10.1007/978-3-642-24364-6_17)
- [20] Christian Sternagel and René Thiemann. Executable Transitive Closures for Finite Relations. *The Archive of Formal Proofs*, 2011
[afp:Transitive-Closure](https://arxiv.org/abs/1108.0401)
- [21] Christian Sternagel and René Thiemann. Certification of Nontermination Proofs. In *Proceedings of the 3rd International Conference on Interactive Theorem Proving*,

⁴<http://project-coco.uibk.ac.at/2019/categories/infeasibility.php>

2. Contributions

- volume 7406 of *Lecture Notes in Computer Science*, pages 266–282, Springer, 2012
[doi:10.1007/978-3-642-32347-8_18](https://doi.org/10.1007/978-3-642-32347-8_18)
- [22] Mizuhito Ogawa and Christian Sternagel. Open Induction. *The Archive of Formal Proofs*, 2012
[afp:Open_Induction](#)
- [23] Christian Sternagel and René Thiemann. Formalized Knuth-Bendix Orders and Knuth-Bendix Completion. In *Proceedings of the 24th International Conference on Rewriting Techniques and Applications*, volume 21 of *Leibniz International Proceedings in Informatics*, pages 287–302, Schloss Dagstuhl, 2013
[doi:10.4230/LIPIcs.RTA.2013287](https://doi.org/10.4230/LIPIcs.RTA.2013287)
- [24] Christian Sternagel and René Thiemann. Formalizing Monotone Algebras for Certification of Termination and Complexity Proofs. In *Proceedings of the 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*, pages 441–455, Springer, 2014
[doi:10.1007/978-3-319-08918-8_30](https://doi.org/10.1007/978-3-319-08918-8_30)
- [25] Christian Sternagel and René Thiemann. The Certification Problem Format. In *Proceedings of the 11th 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](https://doi.org/10.4204/EPTCS.167.8)
- [26] Christian Sternagel. Imperative Insertion Sort. *The Archive of Formal Proofs*, 2014
[afp:Imperative_Insertion_Sort](#)
- [27] Martin Avanzini, Christian Sternagel, and René Thiemann. Certification of Complexity Proofs using CeTA. In *Proceedings of the 26th International Conference on Rewriting Techniques and Applications*, volume 36 of *Leibniz International Proceedings in Informatics*, pages 23–39, Schloss Dagstuhl, 2015
[doi:10.4230/LIPIcs.RTA.2015.23](https://doi.org/10.4230/LIPIcs.RTA.2015.23)
- [28] Christian Sternagel and René Thiemann. Deriving Comparators and Show Functions in Isabelle/HOL. In *Proceedings of the 6th International Conference on Interactive Theorem Proving*, volume 9236 of *Lecture Notes in Computer Science*, pages 421–437, Springer, 2015
[doi:10.1007/978-3-319-22102-1_28](https://doi.org/10.1007/978-3-319-22102-1_28)
- [29] Christian Sternagel and René Thiemann. Deriving class instances for datatypes. *The Archive of Formal Proofs*, 2015
[afp:Deriving](#)
- [30] Akihisa Yamada, Christian Sternagel, René Thiemann, and Keiichirou Kusakari. AC Dependency Pairs Revisited. In *Proceedings of the 25th EACSL Annual Conference on Computer Science Logic*, volume 62 of *Leibniz International Proceedings in Informatics*, pages 8:1–8:16, Schloss Dagstuhl, 2016
[doi:10.4230/LIPIcs.CSL.2016.8](https://doi.org/10.4230/LIPIcs.CSL.2016.8)

- [31] Bertram Felgenhauer, Julian Nagele, Vincent van Oostrom, and Christian Sternagel. The Z Property. *The Archive of Formal Proofs*, 2016
[afp:Rewriting_Z](#)
- [32] Julian Biendarra, Jasmin Blanchette, Aymeric Bouzy, Martin Desharnais, Mathias Fleury, Johannes Hölzl, Ondřej Kunčar, Andreas Lochbihler, Fabian Meier, Lorenz Panny, Andrei Popescu, Christian Sternagel, René Thiemann, and Dmitriy Traytel. Foundational (Co)datatypes and (Co)recursion for Higher-Order Logic. In *Proceedings of the 14th International Symposium on Frontiers of Combining Systems*, volume 10483 of *Lecture Notes in Computer Science*, pages 3–21, Springer, 2017
[doi:10.1007/978-3-319-66167-4_1](#)
- [33] Florian Meßner, Julian Parsert, Jonas Schöpf, and Christian Sternagel. Homogeneous Linear Diophantine Equations. *The Archive of Formal Proofs*, 2017
[afp:Diophantine_Eqns_Lin_Hom](#)
- [34] Joachim Breitner, Brian Huffman, Neil Mitchell, and Christian Sternagel. HOLCF-Prelude. *The Archive of Formal Proofs*, 2017
[afp:HOLCF-Prelude](#)
- [35] Florian Meßner, Julian Parsert, Jonas Schöpf, and Christian Sternagel. A Formally Verified Solver for Homogeneous Linear Diophantine Equations. In *Proceedings of the 9th International Conference on Interactive Theorem Proving*, volume 10895 of *Lecture Notes in Computer Science*, pages 441–458, Springer, 2018
[doi:10.1007/978-3-319-94821-8_26](#)
- [36] Christian Sternagel and René Thiemann. First-Order Terms. *The Archive of Formal Proofs*, 2018
[afp:First_Order_Terms](#)
- [37] Alexander Lochmann and Christian Sternagel. Certified ACKBO. In *Proceedings of the 8th International Conference on Certified Programs and Proofs*, pages 144–151, ACM, 2019
[doi:10.1145/3293880.3294099](#)
- [38] Florian Meßner and Christian Sternagel. nonreach – A Tool for Nonreachability Analysis. In *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 11427 of *Lecture Notes in Computer Science*, pages 337–343, Springer, 2019
[doi:10.1007/978-3-030-17462-0_19](#)
- [39] Christian Sternagel and Akihisa Yamada. Reachability Analysis for Termination and Confluence of Rewriting. In *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 11427 of *Lecture Notes in Computer Science*, pages 262–278, Springer, 2019
[doi:10.1007/978-3-030-17462-0_15](#)
- [40] Jürgen Giesl, Albert Rubio, Christian Sternagel, Johannes Waldmann, and Akihisa Yamada. The Termination and Complexity Competition. In *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis*

2. Contributions

of Systems, volume 11429 of *Lecture Notes in Computer Science*, pages 156–166, Springer, 2019

[doi:10.1007/978-3-030-17502-3_10](https://doi.org/10.1007/978-3-030-17502-3_10)

Part II.

Selected Publications

3. A Mechanized Proof of GHC’s Mergesort

Publication Details

Christian Sternagel. Proof Pearl—A Mechanized Proof of GHC’s Mergesort. *Journal of Automated Reasoning*, 51(4):357–370, 2013

[doi:10.1007/s10817-012-9260-7](https://doi.org/10.1007/s10817-012-9260-7)

Christian Sternagel. Efficient Mergesort. *The Archive of Formal Proofs*, 2011 (last update: 2019)

`afp:Efficient-Mergesort`

Abstract

In this Isabelle/HOL proof pearl I prove correctness, stability, and linearithmic complexity of the natural mergesort implementation that comes with the *Glasgow Haskell Compiler*. This not only constitutes another example of applying a state-of-the-art poof assistant to real-world code, but also allows users to take advantage of the formalized algorithm during code generation.

3.1. Introduction

In proof assistants like Isabelle/HOL [103], it is common to define algorithms in terms of abstract specifications instead of actual implementations. Where specifications are typically easy to understand and reason about, but potentially inefficient. For implementations in contrast, efficiency is often key, but may come at the price of code that is incomprehensible for the uninitiated.

Specifications facilitate high-level proofs that are mostly concerned with abstract properties and avoid “implementation details” that tend to be tedious to reason about. From the logical viewpoint this is mostly the end of the story: we define an algorithm and prove its desired properties. For actual use in real-world code, however, such specifications are often not efficient enough. This is where *algorithm refinement* comes into play. That is, we implement an alternative, more efficient, variant of our algorithm and formally prove that both versions are equivalent, or in other words, that given equal arguments, both variants yield the equal results.

Additionally, Isabelle/HOL supports code generation [52], that is, to automatically generate actual source code in various target languages (currently, Haskell, OCaml, Scala, and StandardML) from a given formalization of an algorithm. The resulting code is correct by construction.

3. A Mechanized Proof of GHC’s Mergesort

Together with algorithm refinement, code generation opens up the following three-step workflow for efficient verified programs: First formalize “easy” variants of the constituting algorithms and prove all desired properties. Then, formalize efficient variants of the same algorithms and prove them equivalent. And finally, use code generation to obtain an efficient program that is guaranteed to satisfy all properties that have been proven in initially.

Contribution. In the following I present my Isabelle/HOL¹ formalization of GHC’s sorting algorithm for lists² (for brevity, referred to as *sort* in the remainder).

More specifically, my contributions are as follows:

- I describe the implementation of *sort* in GHC’s standard library and give a corresponding implementation in Isabelle/HOL (Section 3.2).
- Then, I explain some preliminaries (Section 3.3) related to sorting that are already provided by Isabelle/HOL.
- In the main part I sketch my formalized proofs of correctness and stability of *sort*.
- Finally, I present my formalization of its linearithmic complexity (regarding the required number of comparisons).

Since *sort* is part of GHC’s standard library, this formalization constitutes a verification of real-world code that is (at least implicitly) used in many Haskell programs. In this work, I just give an overview of the most important ideas and refer to the *Archive of Formal Proofs* [130] for details.

Motivation. My original motivation was to tune *CeTA*,³ a fully verified program whose code is generated from an underlying Isabelle/HOL formalization [165]. *CeTA* is a certifier for termination proofs of first-order term rewrite systems (TRSs for short). Such proofs are highly modular—in the sense that typically, a given TRS is split into several smaller TRSs for which termination is proved separately—and often use transformation techniques (like semantic labeling) that can blow up the number of rewrite rules exponentially. Moreover, for reduction pairs—which are employed to delete rewrite rules from TRSs that cannot be the cause of nontermination—a common task for a certifier is to check that the remaining TRS is a subset of the original one. Since in *CeTA*, TRSs are represented as lists of rewrite rules, this check incorporates sorting those lists and was identified as one of the performance bottle-necks of *CeTA*. My first step was to replace Isabelle/HOL’s default sorting algorithm (an insertion sort variant provided in the *HOL.List* theory) by a supposedly more efficient version from the library (a quick sort variant provided in *HOL-Library.Multiset*). Since this did not give the desired speedup (unfortunately, *CeTA* does not work properly together with *HOL-Library.Code_Binary_Nat*; see also the remark

¹Our development is based on version *Isabelle2019* (June 2019).

²<http://hackage.haskell.org/package/base-4.12.0.0/docs/src/Data.OldList.html#sort>

³More precisely, to make its runtime scale better on large inputs.

```

sequences key (a # b # xs) =
  (if key b < key a then desc key b [a] xs else asc key b ((#) a) xs)
sequences key [x] = [[x]]
sequences key [] = []

asc key a as (b # bs) =
  (if key a ≤ key b then asc key b (λys. as (a # ys)) bs
   else as [a] # sequences key (b # bs))
asc key a as [] = [as [a]]

desc key a as (b # bs) =
  (if key b < key a then desc key b (a # as) bs
   else (a # as) # sequences key (b # bs))
desc key a as [] = [a # as]

```

Figure 3.1.: Formalization of *sequences*.

in Section 3.5) and our target programming language is Haskell, we decided to formalize the sorting algorithm of GHC's standard library.

3.2. GHC's Sorting Algorithm

Consider GHC's sorting algorithm for lists depicted in Listing 3.1. It is a mergesort variant that takes advantage of (reverse) sorted subsequences occurring in the input. The three mutually recursive functions *sequences*, *descending*, and *ascending* take care of transforming an input list into a list of sorted lists. To this end, *ascending* detects sorted subsequences and returns them unchanged, while *descending* detects reverse sorted subsequences and flips them along the way. The resulting sequence of sorted lists is merged into a single list by *mergeAll*. Note that this implementation behaves especially well on typically problematic cases like sorted lists or reverse sorted lists as input. In both cases, *sequences* just needs a single traversal and no merging is required.

Before I treat my Isabelle/HOL formalization of *sort*, some words on its origin. According to the GHC sources, the algorithm is rumored to be based on code by Lennart Augustsson⁴ and possibly bears similarities to an algorithm by Richard O'Keefe [112] (which does not seem to be available any longer). This rumor is supported by the chapter about sorting of [116]. However, I was not able to find a definite answer.

In my Isabelle/HOL formalization I define *sequences* and *merge_all* as shown in Figure 3.1 and Figure 3.2, respectively. When comparing these definitions to those from Listing 3.1 there are some differences that may need explanation. First, partly for brevity and partly to conform to Isabelle/HOL's naming conventions, I changed the names of some functions (*ascending* and *descending* became *asc* and *desc*, and *merge_pairs* as well as *merge_all* use underscores instead of CamelCase). Furthermore, Isabelle/HOL's syntax is slightly different from Haskell's. More specifically, '#' denotes list-cons (':' in

⁴www.mail-archive.com/haskell@haskell.org/msg01822.html

3. A Mechanized Proof of GHC's Mergesort

```
sort = sortBy compare
sortBy cmp = mergeAll . sequences
  where
    sequences (a:b:xs)
      | a `cmp` b == GT = descending b [a] xs
      | otherwise       = ascending  b (a:) xs
    sequences xs = [xs]

    descending a as (b:bs)
      | a `cmp` b == GT = descending b (a:as) bs
    descending a as bs  = (a:as) : sequences bs

    ascending a as (b:bs)
      | a `cmp` b /= GT = ascending b (\ys -> as (a:ys)) bs
    ascending a as bs   = let !x = as [a]
                          in x : sequences bs

    mergeAll [x] = x
    mergeAll xs  = mergeAll (mergePairs xs)

    mergePairs (a:b:xs) = let !x = merge a b
                          in x : mergePairs xs

    mergePairs xs      = xs

    merge as@(a:as') bs@(b:bs')
      | a `cmp` b == GT = b : merge as  bs'
      | otherwise       = a : merge as' bs
    merge [] bs        = bs
    merge as []        = as
```

Listing 3.1: GHC's Sort

```

merge key (a # as) (b # bs) =
  (if key b < key a then b # merge key (a # as) bs
   else a # merge key as (b # bs))
merge key [] bs = bs
merge key (v # va) [] = v # va

merge_pairs key (a # b # xs) = merge key a b # merge_pairs key xs
merge_pairs key [] = []
merge_pairs key [v] = [v]

merge_all key [] = []
merge_all key [x] = x
merge_all key (v # vb # vc) = merge_all key (merge_pairs key (v # vb # vc))

```

Figure 3.2.: Formalization of *merge_all*.

Haskell). Another difference is that instead of Haskell’s `Ord` typeclass, we are using Isabelle/HOL’s built-in typeclass `linorder`, whose instances are all linearly ordered types. As a consequence I do not parametrize my functions over a compare-function, but rather over a key-function that turns list-elements into elements of some linearly ordered type.

Further note that Isabelle/HOL disambiguates the patterns on the left-hand sides of equations such that at most one defining equation is applicable to any term. In Haskell, on the other hand, this is guaranteed by trying patterns from top to bottom.

Despite these rather cosmetic changes, I hope that it is still sufficiently obvious that my formalization is indeed handling the function of Listing 3.1. (By the way, if you want to see the Haskell code that can be generated from the formalization, just use

```
export_code msort_key in Haskell
```

inside Isabelle/HOL.)

Note. The Haskell implementation of `mergeAll` is potentially nonterminating (when called on the empty list), however, by construction the result of `sequences` contains at least one element. Hence there is no problem. In Isabelle/HOL all functions must be terminating and hence the Haskell version is not accepted. That is, why our formalization of *merge_all* contains an extra case for the empty list (which is never encountered during executions of *sort*).

3.3. Preliminaries

Before I describe the default sorting algorithm of Isabelle/HOL, let us have a closer look at the properties that we are interested in. The two properties of sorting algorithms that are of main interest are *correctness* and *stability*. In the following, we investigate each of them in turn and show how they are formalized in Isabelle/HOL’s library.

3. A Mechanized Proof of GHC's Mergesort

3.3.1. Correctness

Probably the first thing that comes to mind, when we think about the correctness of a sorting algorithm, is that its result should be, well, sorted.

Definition 3.3.1 (Sortedness). *A list is sorted when every two consecutive elements are in order. In Isabelle/HOL this is expressed as a recursive function given by the equations:*

```
sorted [] = True
sorted (x # xs) = (( $\forall y \in \text{set } xs. x \leq y$ )  $\wedge$  sorted xs)
```

Note, however, that sortedness on its own is not sufficient to describe the correctness of a sorting algorithm. Consider, for example, the function `wrongsort xs = []`. Even though, its result is sorted, it is clearly not a correct sorting algorithm. We also have to make sure that a prospective sorting algorithm does not add or remove elements. This property is formulated using multisets in Isabelle/HOL. Where a multiset is like a set in that the order of elements is not important, but may contain multiple copies of equal elements.

Definition 3.3.2 (Element Invariance). *A function $f :: 'a \text{ list} \Rightarrow 'a \text{ list}$ is element invariant if it does neither add nor remove elements. More formally, f has to satisfy*

```
mset (f xs) = mset xs
```

where `mset` (defined in theory `HOL-Library.Multiset`) turns a list into a multiset.

Together, the above two properties allow us to define the *correctness* of a sorting algorithm.

Definition 3.3.3 (Correctness). *A function $f :: 'a \text{ list} \Rightarrow 'a \text{ list}$ is a correct sorting algorithm whenever it is element invariant and produces only sorted results.*

In the standard Isabelle/HOL distribution an archetypical sorting algorithm is provided by

$$\text{sort_key } f \text{ } xs = \text{foldr } (\text{insert_key } f) \text{ } xs \text{ } [] \quad (3.1)$$

(in theory `HOL.List`) where `insert_key` is defined by the equations

```
insert_key f x [] = [x]
insert_key f x (y # ys) =
  (if  $f \text{ } x \leq f \text{ } y$  then  $x \# y \# ys$  else  $y \# \text{insert\_key } f \text{ } x \text{ } ys$ )
```

The correctness proof of `sort_key` is distributed over the theories `HOL.List` (for sortedness) and `HOL-Library.Multiset` (for element invariance):

```
sorted_sort_key: sorted (map f (sort_key f xs))
mset_sort:      mset (sort_key f xs) = mset xs
```


3.3.2. Stability

A sorting algorithm is stable when it does not change the relative order of equal elements. Since in Isabelle/HOL equality is built-in (and hence there is no way to distinguish between two equal elements), stability of a sorting algorithm can only be expressed in presence of a key-function, that is, a function that, given an element, produces a key according to which this element should be sorted.

Example 3.3.4. *Consider the list $[2, 3, 2]$. After sorting we obtain $[2, 2, 3]$. It is impossible to say inside Isabelle/HOL whether the first 2 in the result is the same as the first one in the input (or rather, all four occurrences of 2 refer to the same entity). Having a key-function, we can apply a simple trick. First we add the indices of elements to the input $[(2, 0), (3, 1), (2, 2)]$. Then we sort the list using the key-function `fst` (that is, projecting to the first components of the pairs). Finally, we can see for each 2, from which index in the input list it originates. If the result is $[(2, 0), (2, 2), (3, 1)]$ sorting was indeed stable.*

Definition 3.3.5 (Stability). *A sorting function $f :: ('a \Rightarrow 'b) \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list}$ is stable (with respect to the key-function $\text{key} :: 'a \Rightarrow 'b$) whenever the relative order of elements having the same key does not change between xs and $f \text{ key } xs$. In Isabelle/HOL this is expressed as follows*

$$\forall k. [y \leftarrow f \text{ key } xs. \text{key } y = k] = [y \leftarrow xs. \text{key } y = k]$$

where we use the convenience syntax $[x \leftarrow xs. P \ x]$ instead of `filter P xs` (where `filter` keeps just those elements of a list that satisfy the given predicate P).

3.3.3. Goal

Why are we actually interested in the above properties? Correctness should be clear, we want to make sure that `sort` really is a sorting algorithm. But why do we need stability? In principle there are several reasons why stability is interesting: only stable sorting algorithms allow for incremental sorting (for example, sort according to key A and for those elements with equal A, sort according to key B), swapping elements may cause memory updates on physical media, etc. However, my interest in stability has more ad hoc reasons. Those will become clear after showing the following lemma (which is to be found in theory `HOL-Library.Multiset`)

$$\begin{aligned} & \llbracket \text{mset } (f \text{ key } xs) = \text{mset } xs; \\ & \bigwedge k. k \in \text{set } (f \text{ key } xs) \implies \\ & \quad [x \leftarrow f \text{ key } xs. \text{key } k = \text{key } x] = [x \leftarrow xs. \text{key } k = \text{key } x]; \\ & \quad \text{sorted } (\text{map } \text{key } (f \text{ key } xs)) \rrbracket \\ & \implies \text{sort_key } \text{key } xs = f \text{ key } xs \end{aligned} \tag{3.2}$$

which states that it is sufficient for a function f to be a correct (with respect to the key-function `key`) and stable sorting algorithm, in order to be logically equivalent to `sort_key`. Hence, if we succeed in proving the above three assumptions for some function f , we may use it interchangeably with `sort_key`. This, in turn, allows us to install a more

3. A Mechanized Proof of GHC's Mergesort

efficient sorting algorithm than (3.1) for code generation. Thus, every formalization using `sort_key` can take advantage of the more efficient algorithm in generated code for free.

3.4. Efficient Mergesort

The definition of sorting as given in (3.1) is a reasonable implementation and a good compromise between efficiency and ease of specification. In the end, efficiency is irrelevant for the logic and hence definitions should be as natural and easy as possible. For code generation on the other hand, efficiency is a concern. The typical way of handling this situation is starting with a natural, (maybe) inefficient, but easy to use definition and use it throughout the formalization. Then, before generating code, prove so called *code equations* that show the equivalence of this natural definition to some more efficient variant. In the remainder of this section I provide a code equation for `sort_key` that tunes its performance. As we have seen at the end of Section 3.3, we need to show element invariance, stability, and sortedness in order to prove a function equivalent to `sort_key`.

In the following, I describe my corresponding formalization, where most of the proofs are automatic (that is, solved by automatic methods like `auto`, `blast`, `simp`, etc., after indicating the used induction schema).

Obviously most non-trivial proofs about *sequences* require induction. Since we have a mutual dependency between *sequences*, *asc*, and *desc* we have to prove facts about these three functions simultaneously. The corresponding induction schema that is provided by Isabelle/HOL can be seen in Figure 3.3 at the end of this chapter. Applying this schema typically requires us to strengthen the induction hypothesis and introduce additional assumptions for *asc* with its function argument.

Concerning the functional argument of *asc* we need that it behaves “reasonably.” What exactly I mean by reasonable behavior is covered by the predicate *ascP* that is given by

$$\text{ascP } f = (\forall xs. f \text{ xs} = f [] @ xs)$$

and basically says that a function is reasonable as argument to *asc* whenever it only adds some prefix to its (list) argument.

3.4.1. Correctness

Now we have the main ingredients to prove two important facts, *sequences* does not remove or add elements and generates a list of sorted lists

When proven simultaneously with appropriate facts about *asc* and *desc*, both proofs run through automatically in Isabelle/HOL. Hence I just give the corresponding lemmas that are proven by mutual induction. First for element invariance:

$$\begin{aligned} \text{mset } (\text{concat } (\text{sequences key xs})) &= \text{mset } xs & (3.3) \\ \text{ascP } f \implies \text{mset } (\text{concat } (\text{asc key x f ys})) &= \{\#x\} + \text{mset } (f []) + \text{mset } ys \\ \text{mset } (\text{concat } (\text{desc key x xs ys})) &= \{\#x\} + \text{mset } xs + \text{mset } ys \end{aligned}$$

where `concat` concatenates all elements of a list of lists into a single list. Then for sortedness:

$$\begin{aligned}
& \forall x \in \text{set } (\text{sequences key } xs). \text{sorted } (\text{map key } x) \\
& \llbracket \text{ascP } f; \text{sorted } (\text{map key } (f [])); \forall x \in \text{set } (f []). \text{key } x \leq \text{key } a \rrbracket \\
& \implies \forall x \in \text{set } (\text{asc key } a \text{ } f \text{ } ys). \text{sorted } (\text{map key } x) \\
& \llbracket \text{sorted } (\text{map key } xs); \forall x \in \text{set } xs. \text{key } a \leq \text{key } x \rrbracket \\
& \implies \forall x \in \text{set } (\text{desc key } a \text{ } xs \text{ } ys). \text{sorted } (\text{map key } x)
\end{aligned} \tag{3.4}$$

The corresponding facts for `merge_all` are automatically proven by induction:

$$mset (\text{merge_all key } xs) = mset (\text{concat } xs) \tag{3.5}$$

$$\forall x \in \text{set } xs. \text{sorted } (\text{map key } x) \implies \text{sorted } (\text{map key } (\text{merge_all key } xs)) \tag{3.6}$$

Together, (3.3) and (3.5) yield element invariance of `msort_key` which is given by `msort_key key xs = merge_all key (sequences key xs)`, whereas (3.4) and (3.6) yield sortedness:

$$mset (\text{merge_all key } (\text{sequences key } xs)) = mset \text{ } xs \tag{3.7}$$

$$\text{sorted } (\text{map key } (\text{merge_all key } (\text{sequences key } xs))) \tag{3.8}$$

This shows that `msort_key` is a correct sorting algorithm.

3.4.2. Stability

At this point, we turn our attention to stability. Stability (or at least a very similar property) of `sequences` is proven by the lemma

$$[y \leftarrow \text{concat } (\text{sequences key } xs). \text{key } y = k] = [y \leftarrow xs. \text{key } y = k] \tag{3.9}$$

whose proof, when proven simultaneously with the two facts

$$\begin{aligned}
& \text{ascP } f \implies \\
& [y \leftarrow \text{concat } (\text{asc key } a \text{ } f \text{ } ys). \text{key } y = k] = [y \leftarrow f \text{ } [a] @ ys. \text{key } y = k] \\
& \llbracket \text{sorted } (\text{map key } xs); \forall x \in \text{set } xs. \text{key } a \leq \text{key } x \rrbracket \\
& \implies [y \leftarrow \text{concat } (\text{desc key } a \text{ } xs \text{ } ys). \text{key } y = k] = \\
& \quad [y \leftarrow a \text{ } \# \text{ } xs @ ys. \text{key } y = k]
\end{aligned}$$

is fully automatic.

The first step towards stability of `merge_all`, is proving the lemma

$$\begin{aligned}
& \text{sorted } (\text{map key } xs) \implies \\
& [y \leftarrow \text{merge key } xs \text{ } ys. \text{key } y = k] = [y \leftarrow xs. \text{key } y = k] @ [y \leftarrow ys. \text{key } y = k]
\end{aligned} \tag{3.10}$$

which states that for sorted lists `xs`, `merge` behaves like list-append on lists that are filtered corresponding to a specific key.

3. A Mechanized Proof of GHC's Mergesort

Using (3.10), we then prove “stability” of *merge_pairs*

$$\begin{aligned} \forall xs \in \text{set } xss. \text{sorted } (\text{map key } xs) \implies \\ [y \leftarrow \text{concat } (\text{merge_pairs key } xss). \text{key } y = k] = [y \leftarrow \text{concat } xss. \text{key } y = k] \end{aligned} \quad (3.11)$$

which, in turn, finally allows us to prove stability of *merge_all*:

$$\begin{aligned} \forall xs \in \text{set } xss. \text{sorted } (\text{map key } xs) \implies \\ [y \leftarrow \text{merge_all key } xss. \text{key } y = k] = [y \leftarrow \text{concat } xss. \text{key } y = k] \end{aligned} \quad (3.12)$$

An easy consequence of (3.12) and (3.4) is

$$[x \leftarrow \text{merge_all key } (\text{sequences key } xs). \text{key } x = k] = [x \leftarrow xs. \text{key } x = k] \quad (3.13)$$

showing stability of *msort_key key*.

Finally, using (3.2), whose assumptions are discharged by (3.7), (3.13), and (3.8), we can establish the equation:

$$\text{sort_key key} = \text{msort_key key}$$

3.4.3. Complexity

Concerning the complexity of *sort*, I concentrate on the number of comparisons that are required. This number is captured by the following definitions, where *c_f* computes the number of comparisons required by the corresponding function *f*:

$$\begin{aligned} c_sequences \text{ key } (x \# y \# zs) &= \\ &1 + (\text{if key } y < \text{key } x \text{ then } c_desc \text{ key } y \text{ zs else } c_asc \text{ key } y \text{ zs}) \\ c_sequences \text{ key } [] &= 0 \\ c_sequences \text{ key } [x] &= 0 \\ c_asc \text{ key } x (y \# ys) &= \\ &1 + (\text{if } \neg \text{key } y < \text{key } x \text{ then } c_asc \text{ key } y \text{ ys else } c_sequences \text{ key } (y \# ys)) \\ c_asc \text{ key } x [] &= 0 \\ c_desc \text{ key } x (y \# ys) &= \\ &1 + (\text{if key } y < \text{key } x \text{ then } c_desc \text{ key } y \text{ ys else } c_sequences \text{ key } (y \# ys)) \\ c_desc \text{ key } x [] &= 0 \\ c_merge \text{ key } (x \# xs) (y \# ys) &= \\ &1 + \\ &(\text{if key } y < \text{key } x \text{ then } c_merge \text{ key } (x \# xs) \text{ ys else } c_merge \text{ key } xs (y \# ys)) \\ c_merge \text{ key } [] \text{ ys} &= 0 \\ c_merge \text{ key } (v \# va) [] &= 0 \\ c_merge_pairs \text{ key } (xs \# ys \# zss) &= c_merge \text{ key } xs \text{ ys} + c_merge_pairs \text{ key } zss \\ c_merge_pairs \text{ key } [] &= 0 \\ c_merge_pairs \text{ key } [x] &= 0 \\ c_merge_all \text{ key } [] &= 0 \\ c_merge_all \text{ key } [x] &= 0 \\ c_merge_all \text{ key } (v \# vb \# vc) &= \\ &c_merge_pairs \text{ key } (v \# vb \# vc) + \\ &c_merge_all \text{ key } (\text{merge_pairs key } (v \# vb \# vc)) \end{aligned}$$

`c_msort key xs = c_sequences key xs + c_merge_all key (sequences key xs)`

Some easy facts about the `merge` and `merge_pairs` related functions are as follows (where $|x|$ is short for `length x` which computes the number of elements in a given list x):

$$|merge_pairs\ key\ xs| = (1 + |xs|) \div 2 \quad (3.14)$$

$$|concat\ (merge_pairs\ key\ xss)| = |concat\ xss| \quad (3.15)$$

$$c_merge\ key\ xs\ ys \leq |xs| + |ys| \quad (3.16)$$

$$c_merge_pairs\ key\ xss \leq |concat\ xss| \quad (3.17)$$

The key fact about `merge_all` is:

$$c_merge_all\ key\ xss \leq |concat\ xss| * \lceil \log 2\ |xss| \rceil \quad (3.18)$$

Proof. The proof is by induction on the computation of `c_merge_all`. We concentrate on the nontrivial recursive case arising from the third equation. It follows that `xss` is of the form `xs # ys # zss`. Further note that

$$\lceil \log 2\ (\text{real } n + 2) \rceil = \lceil \log 2\ (\text{real } ((n + 1) \div 2 + 1)) \rceil + 1 \quad (\star)$$

for arbitrary n .

Now, let $m = |concat\ xss|$. Then we have

$$\begin{aligned} & c_merge_all\ key\ xss \\ &= c_merge_pairs\ key\ xss + c_merge_all\ key\ (merge_pairs\ key\ xss) \\ &\leq m + c_merge_all\ key\ (merge_pairs\ key\ xss) && \text{using (3.17)} \\ &\leq m + |concat\ (merge_pairs\ key\ xss)| * \lceil \log 2\ |merge_pairs\ key\ xss| \rceil && \text{(IH)} \\ &= m + m * \lceil \log 2\ |merge_pairs\ key\ xss| \rceil && \text{by (3.15)} \\ &= m + m * \lceil \log 2\ ((1 + |xss|) \div 2) \rceil && \text{by (3.14)} \\ &= m + m * \lceil \log 2\ ((1 + |zss|) \div 2 + 1) \rceil \\ &= m * (\lceil \log 2\ ((1 + |zss|) \div 2 + 1) \rceil + 1) \\ &= m * \lceil \log 2\ (|zss| + 2) \rceil && \text{by } (\star) \\ &= m * \lceil \log 2\ |xss| \rceil \end{aligned}$$

□

□

Mutual induction yields the following useful results about `sequences`. The first group of three gives a bound on `c_sequences`:

$$c_sequences\ key\ xs \leq |xs| - 1 \quad (3.19)$$

$$c_asc\ key\ x\ ys \leq |ys|$$

$$c_desc\ key\ x\ ys \leq |ys|$$

3. A Mechanized Proof of GHC's Mergesort

The second group of three is concerned with the total number of elements in the resulting lists of lists:

$$|concat (sequences key xs)| = |xs| \quad (3.20)$$

$$\begin{aligned} ascP f &\implies |concat (asc key a f ys)| = 1 + |f []| + |ys| \\ |concat (desc key a xs ys)| &= 1 + |xs| + |ys| \end{aligned}$$

The third group of three with the fact that for nonempty inputs *sequences* yields nonempty results.

$$\begin{aligned} xs \neq [] &\implies sequences\ key\ xs \neq [] \\ ascP f &\implies asc\ key\ a\ f\ ys \neq [] \\ desc\ key\ a\ xs\ ys &\neq [] \end{aligned} \quad (3.21)$$

Using these facts we obtain the desired linearithmic bound on *c_msort*:

$$c_msort\ key\ xs \leq |xs| + |xs| * \lceil \log 2\ |xs| \rceil \quad (3.22)$$

Proof. Let $n = |xs|$ and note that

$$c_merge_all\ key\ (sequences\ key\ xs) \leq n * \lceil \log 2\ n \rceil \quad (\star)$$

as shown by the derivation:

$$\begin{aligned} &c_merge_all\ key\ (sequences\ key\ xs) \\ &\leq n * \lceil \log 2\ |sequences\ key\ xs| \rceil \text{ by (3.18) with } xss = sequences\ key\ xs \\ &\leq n * \lceil \log 2\ n \rceil \text{ by (3.20)} \end{aligned}$$

We conclude the proof by:

$$\begin{aligned} c_msort\ xs\ xs &= c_sequences\ xs\ xs + c_merge_all\ xs\ (sequences\ xs\ xs) \\ &\leq n + n * \lceil \log 2\ n \rceil \text{ using (3.19) and } (\star) \end{aligned}$$

□

□

3.5. Conclusion and Related Work

I have given an Isabelle/HOL formalization of GHC's mergesort algorithm, showing correctness, stability, and linearithmic complexity. On the one hand, this showcases once more that state-of-the-art proof assistants like Isabelle/HOL, can be used to verify real-world code. On the other hand, our formalization allows existing theories that rely on Isabelle/HOL's default sorting algorithm to take advantage of the more efficient *sort* during code generation. In order to do so, you just have to import *Efficient-Mergesort.Efficient_Sort* (from the Archive of Formal Proofs) in the header of your theory.

	#-elements	Haskell		OCaml		Scala		StandardML	
		is	qs	is	qs	is	qs	is	qs
inc	100,000	1.1	1.9	3.3	15.7	1.2	14.4	0.8	2.5
	500,000	1.1	2.3	4.0	13.2	3.2	19.0	0.9	2.2
	1,000,000	1.1	2.8	4.6	14.0	4.5	27.1	0.6	2.0
dec	100,000	∞	2.0	∞	94.0	∞	20.3	∞	84.0
	500,000	∞	2.9	∞	28.5	∞	23.7	∞	15.6
	1,000,000	∞	3.2	∞	29.8	∞	31.3	∞	13.9
rnd	100,000	∞	1.4	∞	1.4	∞	2.8	∞	2.3
	500,000	∞	1.4	∞	1.4	∞	2.6	∞	1.4
	1,000,000	∞	1.6	∞	1.4	∞	2.7	∞	1.5

Table 3.1.: Relative speedup of *sort*.

The key points to achieve a compact (240 lines for correctness and stability, plus another 130 lines for complexity) formalization are *mutual induction* using the induction schemas that are generated by Isabelle’s function package [74] and *generalizations*. The latter is of course well-known, nevertheless, I think that the generalizations in my formalization constitute another nice example of this concept.

Assessment. In order to compare the generated code for *sort* to Isabelle/HOL’s default insertion sort (is) and the alternative quicksort (qs) implementations from theory *HOL-Library.Multiset* in the standard library, I conducted some experiments whose results can be seen in Table 3.1 on page 39. I tested code generated for different target languages (Haskell, OCaml, Scala, and StandardML) on ascending (inc), descending (dec), and random (rnd) lists of integers of various sizes (100,000 elements, 500,000 elements, and 1,000,000 elements, respectively). In each column of the table, the speedup of *sort* with respect to the given algorithm is listed (that is, a number greater than 1 indicates that *sort* was faster), where I aborted tests after a timeout of 60 seconds (indicated by a speedup of ∞). Each value corresponds to the average results on 100 samples. For every target language, a small wrapper program reads a list of integers and applies the sorting algorithm under consideration. Note that qs performs worse, if it is not used together with the theory *HOL-Library.Code_Binary_Nat*, since the pivot of a list is computed using Isabelle/HOL’s *nat* type which by default uses Peano numbers (also in generated code).

In total *sort* is the algorithm of choice, independent of the used target language. It performs slightly better than qs, even when *HOL-Library.Code_Binary_Nat* is loaded.

A note on *HOL-Library.Code_Binary_Nat*. The default representation of natural numbers in Isabelle/HOL is the data type

```
datatype nat = 0 / Suc nat
```

that is, a unary encoding by so called Peano numbers.

3. A Mechanized Proof of GHC's Mergesort

Compared to the *integer* types which are typically part of any programming language, arithmetic operations on Peano numbers are quite slow. To solve this problem, the theory *HOL-Library.Code_Binary_Nat* (which in turn is based on *HOL-Library.Code_Abstract_Nat*) may be loaded to set up the code generator such that it uses the following more efficient binary encoding of natural numbers:

```
datatype num = One | Bit0 num | Bit1 num
```

In the quicksort variant of Isabelle/HOL, the pivot is computed by division on natural numbers. An advantage of *sort* is that it does not involve any arithmetic operations on natural numbers and thus performs well even without loading *HOL-Library.Code_Binary_Nat*.

Related Work. I am aware of two other formalizations of mergesort. The first is a Coq formalization⁵ which does, however, not consider stability (which I personally found to be the most challenging part). The second is an ACL2 formalization⁶ which, again, does not consider stability and is based on a theory of so called *powerlists*.

There are also formalizations of other sorting algorithms in various systems, like insertion sort, quicksort, and heapsort in Coq [38]; insertion sort (theory *HOL.List*) and quicksort (theory *HOL-Library.Multiset*) in Isabelle/HOL.

Acknowledgments. I thank the anonymous referees for helpful suggestions.

⁵<http://coq.inria.fr/stdlib/Coq.Sorting.Mergesort.html>

⁶<https://github.com/acl2/acl2/blob/master/books/powerlists/merge-sort.lisp>

$$\begin{aligned}
& \llbracket \bigwedge a \ b \ xs. \\
& \quad \llbracket \text{key } b < \text{key } a \implies R \ b \ [a] \ xs; \neg \text{key } b < \text{key } a \implies Q \ b \ ((\#) \ a) \ xs \rrbracket \\
& \quad \implies P \ (a \ \# \ b \ \# \ xs); \\
& \quad \bigwedge x. P \ [x]; P \ []; \\
& \quad \bigwedge a \ as \ b \ bs. \\
& \quad \quad \llbracket \text{key } a \leq \text{key } b \implies Q \ b \ (\lambda ys. as \ (a \ \# \ ys)) \ bs; \\
& \quad \quad \neg \text{key } a \leq \text{key } b \implies P \ (b \ \# \ bs) \rrbracket \\
& \quad \quad \implies Q \ a \ as \ (b \ \# \ bs); \\
& \quad \quad \bigwedge a \ as. Q \ a \ as \ []; \\
& \quad \quad \bigwedge a \ as \ b \ bs. \\
& \quad \quad \quad \llbracket \text{key } b < \text{key } a \implies R \ b \ (a \ \# \ as) \ bs; \neg \text{key } b < \text{key } a \implies P \ (b \ \# \ bs) \rrbracket \\
& \quad \quad \quad \implies R \ a \ as \ (b \ \# \ bs); \\
& \quad \quad \bigwedge a \ as. R \ a \ as \ [] \rrbracket \\
& \implies P \ a0.0 \\
& \llbracket \bigwedge a \ b \ xs. \\
& \quad \llbracket \text{key } b < \text{key } a \implies R \ b \ [a] \ xs; \neg \text{key } b < \text{key } a \implies Q \ b \ ((\#) \ a) \ xs \rrbracket \\
& \quad \implies P \ (a \ \# \ b \ \# \ xs); \\
& \quad \bigwedge x. P \ [x]; P \ []; \\
& \quad \bigwedge a \ as \ b \ bs. \\
& \quad \quad \llbracket \text{key } a \leq \text{key } b \implies Q \ b \ (\lambda ys. as \ (a \ \# \ ys)) \ bs; \\
& \quad \quad \neg \text{key } a \leq \text{key } b \implies P \ (b \ \# \ bs) \rrbracket \\
& \quad \quad \implies Q \ a \ as \ (b \ \# \ bs); \\
& \quad \quad \bigwedge a \ as. Q \ a \ as \ []; \\
& \quad \quad \bigwedge a \ as \ b \ bs. \\
& \quad \quad \quad \llbracket \text{key } b < \text{key } a \implies R \ b \ (a \ \# \ as) \ bs; \neg \text{key } b < \text{key } a \implies P \ (b \ \# \ bs) \rrbracket \\
& \quad \quad \quad \implies R \ a \ as \ (b \ \# \ bs); \\
& \quad \quad \bigwedge a \ as. R \ a \ as \ [] \rrbracket \\
& \implies Q \ a1.0 \ a2.0 \ a3.0 \\
& \llbracket \bigwedge a \ b \ xs. \\
& \quad \llbracket \text{key } b < \text{key } a \implies R \ b \ [a] \ xs; \neg \text{key } b < \text{key } a \implies Q \ b \ ((\#) \ a) \ xs \rrbracket \\
& \quad \implies P \ (a \ \# \ b \ \# \ xs); \\
& \quad \bigwedge x. P \ [x]; P \ []; \\
& \quad \bigwedge a \ as \ b \ bs. \\
& \quad \quad \llbracket \text{key } a \leq \text{key } b \implies Q \ b \ (\lambda ys. as \ (a \ \# \ ys)) \ bs; \\
& \quad \quad \neg \text{key } a \leq \text{key } b \implies P \ (b \ \# \ bs) \rrbracket \\
& \quad \quad \implies Q \ a \ as \ (b \ \# \ bs); \\
& \quad \quad \bigwedge a \ as. Q \ a \ as \ []; \\
& \quad \quad \bigwedge a \ as \ b \ bs. \\
& \quad \quad \quad \llbracket \text{key } b < \text{key } a \implies R \ b \ (a \ \# \ as) \ bs; \neg \text{key } b < \text{key } a \implies P \ (b \ \# \ bs) \rrbracket \\
& \quad \quad \quad \implies R \ a \ as \ (b \ \# \ bs); \\
& \quad \quad \bigwedge a \ as. R \ a \ as \ [] \rrbracket \\
& \implies R \ a4.0 \ a5.0 \ a6.0
\end{aligned}$$
Figure 3.3.: Generated mutual induction schema for *sequences*, *asc*, and *desc*.

4. Certified Kruskal's Tree Theorem

Publication Details

Christian Sternagel. Certified Kruskal's Tree Theorem. *Journal of Formalized Reasoning*, 7(1):45–62, 2014

[doi:10.6092/issn.1972-5787/4213](https://doi.org/10.6092/issn.1972-5787/4213)

Christian Sternagel. Well-Quasi-Orders. *The Archive of Formal Proofs*, 2012 (last update: 2017)

`afp:Well_Quasi_Orders`

Christian Sternagel. Certified Kruskal's Tree Theorem. In *Proceedings of 3rd International Conference on Certified Programs and Proofs (CPP)*, volume 8307 of *Lecture Notes in Computer Science*, pages 178–193, Springer, 2013

[doi:10.1007/978-3-319-03545-1_12](https://doi.org/10.1007/978-3-319-03545-1_12)

Abstract

This article presents the first formalization of Kruskal's tree theorem in a proof assistant. The Isabelle/HOL development is along the lines of Nash-Williams' original minimal bad sequence argument for proving the tree theorem. Along the way, proofs of Dickson's lemma and Higman's lemma, as well as some technical details of the formalization are discussed.

4.1. Introduction

Termination is a key ingredient for total correctness of programs and thus key to program verification. Instead of focusing on a specific programming language, termination is typically considered in a more abstract setting. In this respect, one of the most studied models of computation is term rewriting, as confirmed by the many automatic tools that are available nowadays (for example, AProVE [44], CiME [26], Matchbox [169], MUTERM [1], T_1T_2 [73], and VMTL [123]; to name a few). A central task in this area is to synthesize well-founded relations. Often this is done incrementally, for example, a given well-founded relation is extended to a bigger structure, like sets, multisets, lists, etc. Since this is not always easy, there is interest in stronger conditions than well-foundedness that preserve well-foundedness when extending a given order to bigger structures in more cases. To illustrate the issue, consider the following example.

Example 4.1.1. *Given a quasi-order \leq and two sets A and B , write $A \leq^+ B$ whenever for every $a \in A$ there is some $b \in B$ such that $a \leq b$. In other words, B majorizes A element-wise. One might ask whether the strict part of \leq^+ , that is, $<^+ = \leq^+ \setminus \geq^+$, is*

4. Certified Kruskal’s Tree Theorem

well-founded whenever the strict part of \leq is. The following counterexample shows that this is not the case. Take \geq_d to denote the divisibility order on natural numbers, that is, $m \geq_d n$ whenever there is a natural number k such that $k \cdot n = m$. Note that the strict part of \geq_d is well-founded but admits infinite antichains, for example, the sequence p_1, p_2, p_3, \dots of all prime numbers in increasing order. Now let P_i denote the set of all prime numbers starting from the i -th, that is, $P_i = \{p_k\}_{k \geq i}$. Then we obtain the strictly decreasing sequence

$$P_1 >_d^+ P_2 >_d^+ P_3 >_d^+ \dots$$

showing that $>_d$ is not well-founded.

It turns out that by preventing infinite antichains, one can obtain well-foundedness of (the strict part of) \leq^+ , that is, when the given quasi-order \leq does not admit infinite antichains *and* its strict part is well-founded, then so is the strict part of \leq^+ . An order satisfying these two conditions (or several equivalent ones) is called a *well-quasi-order* (wqo for short).

A famous result of wqo theory is Kruskal’s tree theorem [77] (sometimes called *the tree theorem* or *Kruskal’s theorem* in the following).

Kruskal’s Tree Theorem 1. *Whenever a set A is well-quasi-ordered by a relation \preceq , then the set of finite trees over A is well-quasi-ordered by homeomorphic embedding with respect to \preceq .*

Its usefulness for termination proving was first shown by Dershowitz [31, 32], who employed *simplification orders*—a class of reduction orders for which well-foundedness follows from Kruskal’s theorem.

Nash-Williams gave a short and elegant proof of the tree theorem [100], where he first established what is now known as the *minimal bad sequence argument*: first assume the existence of a minimal “bad” infinite sequence of elements, then construct an even smaller “bad” infinite sequence, thus contradicting minimality and proving well-quasi-orderedness (since the definition of wqo requires all infinite sequences of elements to be “good”).

Besides the minimal bad sequence argument, Nash-Williams’ work [100] contains proofs of Dickson’s lemma [35] (*if A and B are well-quasi-ordered, then so is the Cartesian product $A \times B$*) and a variant of Higman’s lemma [57] (*if A is well-quasi-ordered, then so is the set of finite subsets of A*), where the latter also incorporates an instance of the minimal bad sequence argument.

The work at hand constitutes a formalization along the lines of Nash-Williams’ original proofs in the proof assistant Isabelle[103].¹ His argumentation is short (in fact, Nash-Williams’ paper consists of only two and a half pages in total) and elegant (which was also the main reason for basing the formalization on his work). However, formalizations using proof assistants typically require us to be more rigorous than with pen and paper. Thus, the formalization is more detailed in places, which results in somewhat longer (about two thousand lines of Isabelle/HOLtheories) proofs. In this article, a high-level

¹Available from <http://isabelle.in.tum.de> (try Isabelle/jEdit for browsing).

overview of the formalization is given. The full development is part of the *Archive of Formal Proofs* [131].

Contributions This article is a reworked version of an earlier account by the author [133]. To the best of the author’s knowledge, the presented work constitutes the first unrestricted formalization of Higman’s lemma in Isabelle/HOL as well as the first formalization of Kruskal’s tree theorem ever. Both are important combinatorial results with applications in rewriting theory. For example, the theory of simplification orders [32, 91] was formalized—on top of the presented work—as part of *IsaFoR*,² where it is applied to show well-foundedness of the Knuth-Bendix order [142].

Moreover, the author believes that besides their high trustworthiness, formalizations of existing mathematical results are also of archival and educational value. Especially since a formalization contains *all* non-trivial steps of a proof. No doubt, more often than not, those steps were already conducted in the minds of the original proof authors. However, when the original author writes down a proof in condensed form for publishing, some of the steps may get lost. If, much later, another person tries to understand the proof, there may be some mental gaps (or in the worst case even errors).

Finally, formalizations are often hard to read for non-experts (but note that the Isar language for Isabelle [171] is a huge improvement in that respect). Thus, the author hopes that this high-level overview makes the presented formalization more accessible.

Comparison to Previous Work In my previous work [133] the focus was on following Nash-Williams’ original argumentation as closely as possible. In hindsight this turned out to pose unnecessary complications in some proofs. However, it is always easier to say which of two variants of a proof is better suited for mechanization after formalizing both. By slightly deviating from the original proofs and starting from a crucial fact about *homogeneous subsequences* (for example, presented by Marc Bezem [163, Appendix 5]) I was able to significantly simplify three parts of the development compared to my previous work: the construction of minimal bad sequences, the proof of Higman’s lemma, and the proof of the tree theorem.

A more detailed comparison to my previous work can be found at the end of every section whose corresponding formalization changed significantly.

Overview The remainder is structured as follows. In Section 4.2, necessary preliminaries are covered. Then, in Section 4.3, a crucial fact about almost-full relations is discussed, which will be useful for many of the later proofs. The next four sections present a formalization of Dickson’s lemma, in Section 4.4; a general construction of minimal bad sequences, in Section 4.5; a formalization of Higman’s lemma, in Section 4.6; and ultimately, a formalization of Kruskal’s tree theorem, in Section 4.7. Some example instances of finite tree data types are discussed in Section 4.8. Finally, the paper concludes in Section 4.9, where also applications are sketched, and future as well as related work is discussed.

²<http://cl-informatik.uibk.ac.at/software/ceta/>

4.2. Preliminaries

Throughout this article, standard mathematical notation is used as far as possible. However, additionally some Isabelle-specific notation is employed, since Isabelle's document preparation facilities were used for typesetting all lemmas and theorems (in the words of Haftmann et al. [53]: *no typos, no omissions, no sweat*; alas, this does not extend to the regular text). Thus, some explanation might be in order.

Isabelle/HOL is a higher-order logic based on the simply-typed lambda calculus. Thus, every term has a type, with *type variables* $'a$, $'b$, $'c$, \dots ; and *type constructors* like `nat` for natural numbers, $'a \Rightarrow 'b$ for the function space, $'a \times 'b$ for ordered pairs, $'a$ *set* for sets, and $'a$ *list* for finite lists. *Type constraints* are written $t :: 't$ and denote that term t is of type $'t$. As usual for lambda calculi, function application is denoted by juxtaposition, that is, $f\ x$ denotes the application of function f to the argument x . The type $'a \Rightarrow 'a \Rightarrow \text{bool}$ is used to encode binary relations.

The following constants from Isabelle/HOL's library are freely used in the remainder: $(\circ) :: ('a \Rightarrow 'b) \Rightarrow ('c \Rightarrow 'a) \Rightarrow 'c \Rightarrow 'b$, where $f \circ g$ denotes the functional composition of the two functions f and g , that is, $f \circ g \stackrel{\text{def}}{=} \lambda x. f\ (g\ x)$, and sometimes f_φ is used instead of $f \circ \varphi$ for brevity (especially when f denotes an infinite sequence and φ is an *index-mapping*, that is, a function from the natural numbers to the natural numbers); $\text{fst} :: 'a \times 'b \Rightarrow 'a$ and $\text{snd} :: 'a \times 'b \Rightarrow 'b$ extract the first and second component of a pair, respectively; $\text{set} :: 'a \text{ list} \Rightarrow 'a \text{ set}$, where $\text{set}\ xs$ is the set of elements occurring in the list xs ; $[] :: 'a \text{ list}$, the empty list; $(\cdot) :: 'a \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list}$, where $x \cdot xs$ denotes adding the element x in front of the list xs ; and $(@) :: 'a \text{ list} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list}$, where $xs @ ys$ denotes the concatenation of the two lists xs and ys . Note that since (\cdot) and $(@)$ are both right-associative and have the same priority, $xs @ y \cdot ys$ is the same as $xs @ (y \cdot ys)$ and denotes a list that is constructed by inserting the element y between those of xs and ys .

When stating formulas, sometimes Isabelle-specific notation is used. Then, \bigwedge denotes universal quantification and \implies (right-associative) implication. Moreover, nested implication, like $A \implies B \implies C$, is abbreviated to $\llbracket A; B \rrbracket \implies C$.

Let \preceq be a binary relation and A a set. The relation \preceq is *reflexive on* A , written $\text{refl}_A(\preceq)$, if and only if $\forall x \in A. x \preceq x$; and *transitive on* A , written $\text{trans}_A(\preceq)$, if and only if $\forall x \in A. \forall y \in A. \forall z \in A. x \preceq y \wedge y \preceq z \longrightarrow x \preceq z$.

An infinite sequence over elements of type $'a$ is represented by a function f of type $\text{nat} \Rightarrow 'a$. The set of all infinite sequences over elements from a set A is denoted by A^ω . A binary relation \preceq is *transitive on a sequence* f , written $\text{trans}_f(\preceq)$, if and only if $\forall i\ j. i < j \longrightarrow f\ i \preceq f\ j$. Note that $< :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$ is transitive on an index-mapping φ if and only if φ is a strictly monotone mapping from natural numbers to natural numbers. Thus, for every f and strictly monotone φ , f_φ is a subsequence of f whose elements are in the same relative order.

A sequence f is *good* with respect to a relation \preceq , written $\text{good}_\preceq(f)$, if and only if there are indices $i < j$ such that $f\ i \preceq f\ j$. A sequence that is not good, is called *bad*.

The author follows Veldman [168] and Vytiniotis et al. in basing wqos on *almost-full* relations (which are basically wqos without transitivity). The main reason for doing

so, is that all the properties of interest also hold for almost-full relations and are easily extended to wqos.

The relation \preceq is *almost-full on A*, written $af_A(\preceq)$, if and only if all infinite sequences over elements of A are good, that is, $\forall f \in A^\omega. \text{good}_{\preceq}(f)$. Note that every almost-full relation is necessarily reflexive: just take an infinite sequence f that repeats an arbitrary element $a \in A$ ad infinitum, then reflexivity trivially follows from the definitions of almost-full and good, that is, there are $i < j$ such that $f \ i \preceq f \ j$ and thus $a \preceq a$.

Let \preceq be almost-full on A . If in addition \preceq is transitive on A , then \preceq is a *wqo on A* (or A is well-quasi-ordered by \preceq), written $wqo_A(\preceq)$. In the literature, several equivalent definitions for wqos are used. One of them, also mentioned in the introduction, is that a wqo is a quasi-order that does not admit infinite antichains and whose strict part is well-founded (see theory *Well_Quasi_Orders.Well_Quasi_Orders* for other definitions and equivalence proofs). Here, an *infinite antichain* f is an infinite sequence such that every two elements at disjoint positions are incomparable, that is, $\forall i \ j. i < j \longrightarrow f \ i \not\preceq f \ j \wedge f \ j \not\preceq f \ i$.

4.3. Homogeneous Sequences

While the definition of almost-full relations just requires that in every infinite sequence there are two elements such that the former is smaller than or equal to the latter, it can be shown that every infinite sequence contains a subsequence on which \preceq is transitive. (In the literature, such sequences are called *homogeneous* [163].) In some cases, this result allows us to obtain transitivity for free (and hence prove several results already for almost-full relations rather than wqos).

Before formally stating the above result, let us have a look at the following variant of Ramsey's theorem (see Isabelle/HOL's library, file `~/src/HOL/Library/Ramsey.thy`) which is used in its proof.

$$\llbracket \text{infinite } Z; \forall i \in Z. \forall j \in Z. i \neq j \longrightarrow h \{i, j\} < n \rrbracket \\ \implies \exists I \ c. I \subseteq Z \wedge \text{infinite } I \wedge c < n \wedge (\forall i \in I. \forall j \in I. i \neq j \longrightarrow h \{i, j\} = c)$$

In words: Let Z be an infinite set and let h be a function that, given a two-element subset of Z , returns a natural number smaller than n . Then there is an infinite subset I of Z and a natural number c smaller than n such that h encodes all two-element subsets of I by c . More abstractly, assume there is an infinite graph with nodes from Z such that every edge has exactly one of n colors. Then there is an infinite subgraph with nodes from I and all edges of color c .

Lemma 4.3.1. *Every infinite sequence f over elements of a set A that is almost-full with respect to \preceq contains a homogeneous subsequence. That is, there is a strictly monotone index-mapping $\varphi :: \text{nat} \Rightarrow \text{nat}$ such that f_φ is transitive with respect to \preceq . In Isabelle: $\llbracket af_A(\preceq); f \in A^\omega \rrbracket \implies \exists \varphi. \forall i \ j. i < j \longrightarrow \varphi \ i < \varphi \ j \wedge f_\varphi \ i \preceq f_\varphi \ j$.*

Proof. Let \preceq be almost-full on A and $f \in A^\omega$. Then partition the set of two-element subsets of the natural numbers into the set $X = \{\{i, j\} \mid i < j \wedge f \ i \preceq f \ j\}$ and

4. Certified Kruskal's Tree Theorem

its complement $Y = - X$ and colorize two-element sets $\{i, j\}$ of natural numbers by 0 (*white*) and 1 (*black*) according to the following function:

$$h \{i, j\} = \begin{cases} 0 & \text{if } \{i, j\} \in X, \\ 1 & \text{otherwise.} \end{cases}$$

Now Ramsey's theorem can be applied (since the set of natural numbers is infinite and there are exactly two colors). Thus, an infinite set I and a color c , such that for all $i \neq j$ in I the corresponding color $h \{i, j\}$ is c , are obtained. Since I is well-ordered, there is a function $\varphi : \text{nat} \Rightarrow \text{nat}$ that enumerates its elements in increasing order, that is, $\varphi i < \varphi j$ for all $i < j$. Moreover, $h \{\varphi i, \varphi j\} = c$ for all $i < j$. Consider the following two cases (for arbitrary but fixed $i < j$):

- **case** (c is *white*). Then, $h \{\varphi i, \varphi j\} = 0$ and thus $\{\varphi i, \varphi j\} \in X$ which implies $f_\varphi i \preceq f_\varphi j$.
- **case** (c is *black*). Then, $h \{\varphi i, \varphi j\} = 1$, and thus $\{\varphi i, \varphi j\} \in Y$ which implies $f_\varphi i \not\preceq f_\varphi j$ and thus yields the bad sequence f_φ , contradicting the fact that \preceq is almost-full on A . \square

Comparison to Previous Work Also in my previous work Ramsey's theorem was employed. However, only to obtain a proof of Dickson's lemma without transitivity and not for the more general result about homogeneous subsequences of this section.

4.4. Dickson's Lemma

In essence, the presented formalization is about preservation of well-quasi-orderedness by certain type constructors (Dickson's lemma for pairs, Higman's lemma for lists, and the tree theorem for trees). For each of these constructors, a way to extend the orders on the base types to an order on the newly constructed type is required. For Dickson's lemma the following is used:

Definition 4.4.1. *Given two orders \preceq_1 and \preceq_2 , the pointwise order on pairs is defined by $(a_1, a_2) \preceq_1 \times \preceq_2 (b_1, b_2) \stackrel{\text{def}}{=} a_1 \preceq_1 b_1 \wedge a_2 \preceq_2 b_2$.*

Using Lemma 4.3.1, Dickson's lemma for almost-full relations is shown.

Lemma 4.4.2. *The pointwise combination $\preceq_1 \times \preceq_2$ of two almost-full relations \preceq_1 and \preceq_2 on sets A_1 and A_2 , is almost-full on the Cartesian product $A_1 \times A_2$. In Isabelle: $\llbracket af_{A_1}(\preceq_1); af_{A_2}(\preceq_2) \rrbracket \implies af_{A_1 \times A_2}(\preceq_1 \times \preceq_2)$.*

Proof. Assume $af_{A_1}(\preceq_1)$ and $af_{A_2}(\preceq_2)$. Moreover, to derive a contradiction, assume $\neg af_{A_1 \times A_2}(\preceq_1 \times \preceq_2)$. Then there is a sequence f on $A_1 \times A_2$ which is bad. Note that $f \text{st} \circ f \in A_1^\omega$ and $f \text{nd} \circ f \in A_2^\omega$. With Lemma 4.3.1 we obtain a strictly monotone index-mapping φ such that $f \text{st} (f_\varphi i) \preceq_1 f \text{st} (f_\varphi j)$ for all $i < j$. Then $f \text{nd} \circ f \circ \varphi \in A_2^\omega$ and thus $f \text{nd} \circ f \circ \varphi$ is good since A_2 is almost-full by assumption. Thus, we obtain indices $i < j$ such that $f \text{nd} (f_\varphi i) \preceq_2 f \text{nd} (f_\varphi j)$. In total, we have $f_\varphi i \preceq_1 \times \preceq_2 f_\varphi j$ which together with $\varphi i < \varphi j$ contradicts the badness of f . \square

Lemma 4.4.2 trivially extends to wqos.

Dickson’s Lemma 1. *The pointwise combination of two wqos is again a wqo. In Isabelle: $\llbracket wqo_{A_1}(\preceq_1); wqo_{A_2}(\preceq_2) \rrbracket \implies wqo_{A_1 \times A_2}(\preceq_1 \times \preceq_2)$.*

Proof. Assuming transitivity of \preceq_1 on A_1 and \preceq_2 on A_2 , it is trivial to show transitivity of $\preceq_1 \times \preceq_2$ on $A_1 \times A_2$. With Lemma 4.4.2, this concludes the proof. \square

Comparison to Previous Work The new proof of Lemma 4.4.2 for almost-full relations is based on homogeneous subsequences. As before, the effect is that transitivity on some infinite sequence is obtained without requiring transitivity of the whole relation.

4.5. Minimal Bad Sequences

Since the minimal bad sequence argument is needed for Higman’s lemma as well as Kruskal’s theorem, a general construction that is applicable to both cases is provided (see theory *Well_Quasi_Orders.Minimal_Bad_Sequences* for the formal proof development). To this end, Isabelle/HOL’s locale mechanism is employed which allows us to define new constants and prove facts using an “interface” of hypothetical constants and assumptions. As long as the assumptions can be discharged, the new constants and proven facts can be instantiated to arbitrary special cases.

Below, the locale *mbs* which captures the construction of a minimal bad sequence over elements from a given set is described (early versions, that could be simplified drastically since, were presented at the *Isabelle Users Workshop* in 2012 [132] and at the *3rd International Conference on Certified Programs and Proofs* [133]). The locale fixes the following constant:

- A set A whose elements are equipped with a size-function $| \cdot | :: 'a \Rightarrow \text{nat}$.

(For Isabelle initiates: here $| \cdot |$ refers to the library type class *size*, which is automatically instantiated for all data types). Since $| \cdot |$ is a well-founded measure, it makes sense to talk about *minimal* elements. It turns out that these ingredients are enough to construct—under the assumption that there is a bad sequence—a minimal bad sequence. Informally, a bad infinite sequence is a *minimal* bad sequence, when replacing any element by a smaller one, turns it into a good sequence. To make this more formal, a partial order on bad infinite sequences is introduced.

Definition 4.5.1. *Infinite sequences over A are partially ordered by the following relation. An infinite sequence f is considered less than another infinite sequence g , written $f \triangleleft^\omega g$, whenever there is a position i such that the two sequences are equal for all earlier elements and $|f\ i| < |g\ i|$, that is, $\exists i. |f\ i| < |g\ i| \wedge (\forall j < i. g\ j = f\ j)$. The reflexive closure of \triangleleft^ω on A is denoted by \preceq^ω .*

In other words, infinite sequences are compared lexicographically with respect to the size of their elements. First note that this order is not well-founded in general.

4. Certified Kruskal's Tree Theorem

Example 4.5.2. Take the set of strings over the alphabet $\{a_1, a_2, a_3, \dots\}$, ordered by $w \preceq v$ if and only if the set of letters in w is a subset of the set of letters in v . Moreover, let size of w be its length. Now, consider the infinite descending sequence of sequences

$$\begin{array}{rcllcl} A_1 = & a_1 a_1 & a_2 & a_3 & a_4 & \cdots \\ A_2 = & a_1 & a_2 a_2 & a_3 & a_4 & \cdots \\ A_3 = & a_1 & a_2 & a_3 a_3 & a_4 & \cdots \\ A_4 = & a_1 & a_2 & a_3 & a_4 a_4 & \cdots \\ & \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

that is, $A_i = a_1, a_2, a_3, \dots, a_i a_i, \dots$. Obviously all the A_i are bad infinite sequences. Furthermore, A_1, A_2, A_3, \dots is an infinite decreasing sequence that shows that \triangleleft^ω is not well-founded.

The example shows that we cannot directly obtain a minimal bad sequence by means of \triangleleft^ω . Thus, in the following we will construct a minimal bad sequence by choosing smallest possible elements from left to right, which is possible since $|\cdot|$ is well-founded. To this end, we need some auxiliary constructions, for example, to filter the set of bad sequences such that only those remain that are equal to a given sequence up to a certain point.

Definition 4.5.3. Two infinite sequences are equal up to position i , when they are equal for all previous positions. For a set of infinite sequences S over elements of A , let S_i^f denote all those elements of S that are equal to f up to position i . Moreover, let $S[i]$ denote the “ i -th column” of the sequences in S , that is, the set $\{f[i] \mid f \in S\}$. Finally, for a subset B of A , let \min_B denote a minimal element of B with respect to its size (which exists, whenever B is not empty; note however that in general it is not uniquely determined, thus the use of Hilbert's choice operator below).

In the formalization \min_B is defined by

$$\min_B = (\text{SOME } x. x \in B \wedge (\forall y \in A. |y| < |x| \longrightarrow y \notin B))$$

where $\text{SOME } x. Q x$ is Hilbert's epsilon operator, that is, it yields a witness x such that $Q x$, whenever $\exists x. Q x$, and some arbitrary value of the appropriate type, otherwise. The above definitions are employed to construct an infinite sequence from a given set of infinite sequences as follows:

Definition 4.5.4. Given the set B of all bad infinite sequences over elements of A , define a new infinite sequence ℓ (intended to be a lower bound of B with respect to \triangleleft^ω) as follows:

$$\ell[i] = \min_{B[i]^f} \ell[i]$$

That is, $\ell[0]$ is a minimal element among the first elements of sequences in B (since $S_0^f = S$ for all sets S and sequences f); and to obtain the $i+1$ -th element, first restrict

\mathcal{B} to those sequences that are equal to ℓ up to position $i+1$, and of the resulting set of sequences take a minimal element among their $i+1$ -th elements. The well-definedness of the above definition is guaranteed by the fact that to obtain the $i+1$ -th element, we only have to consult all the previous elements of ℓ .

The elements of ℓ satisfy the following properties:

$$\mathbf{h} \in \mathcal{B} \implies \ell \restriction i \in \mathcal{B}_i^\ell[i] \quad (4.1)$$

$$[\mathbf{h} \in \mathcal{B}; y \in A; |y| < |\ell \restriction i|] \implies y \notin \mathcal{B}_i^\ell[i] \quad (4.2)$$

That is, under the assumption that there is a bad sequence \mathbf{h} , the i -th element of ℓ is in the i -th column of the sequences of \mathcal{B} that are equal to ℓ up to position i , and is minimal amongst its elements.

Of course it has to be shown that ℓ is indeed a bad infinite sequence.

Lemma 4.5.5. *If there is at least one bad infinite sequence, then ℓ is bad. In Isabelle: $\mathbf{h} \in \mathcal{B} \implies \ell \in \mathcal{B}$.*

Proof. To derive a contradiction, assume that ℓ is good. Then there are indices $i < j$ such that $\ell \restriction i \preceq \ell \restriction j$. Moreover, from (4.1) we obtain $\ell \restriction j \in \mathcal{B}_j^\ell[j]$, which means that there is some bad infinite sequence \mathbf{g} in \mathcal{B}_j^ℓ such that $\mathbf{g} \restriction k = \ell \restriction k$ for all $k \leq j$, and thus $\mathbf{g} \restriction i \preceq \mathbf{g} \restriction j$. This, in turn, means that \mathbf{g} is good and therefore contradicts the previously derived $\mathbf{g} \in \mathcal{B}_j^\ell$. \square

The second crucial property of ℓ is that it is a lower bound of the set \mathcal{B} . That is, every infinite sequence that is strictly smaller than ℓ is not bad.

Lemma 4.5.6. *If there is at least one bad infinite sequence, then every infinite sequence that is strictly smaller than ℓ with respect to \triangleleft^ω is good. In Isabelle: $\mathbf{h} \in \mathcal{B} \implies \forall \mathbf{g}. \mathbf{g} \triangleleft^\omega \ell \longrightarrow \mathbf{g} \notin \mathcal{B}$.*

At this point it can be shown that if a relation is not almost-full, then there is a minimal bad sequence.

Theorem 4.5.7. *Let \preceq be a relation that is not almost-full on A . Then there is a minimal bad sequence, that is, a bad sequence such that all sequences that are strictly smaller with respect to \triangleleft^ω are good. In Isabelle: $\neg \text{af}_A(\preceq) \implies \exists \mathbf{m} \in \mathcal{B}. \forall \mathbf{g}. \mathbf{g} \triangleleft^\omega \mathbf{m} \longrightarrow \text{good}_{\preceq}(\mathbf{g})$.*

Proof. Assume that \preceq is not almost-full. Then there is some sequence $\mathbf{h} \in \mathcal{B}$. Together with Lemma 4.5.5 and Lemma 4.5.6, we obtain that ℓ is a minimal bad sequence. \square

Comparison to Previous Work Instead of basing the *mbs* locale on some arbitrary well-founded and transitive relation (as in [133]), minimality is now fixed to refer to the size of elements. While this is only a specific instance of the previous construction, it suffices for all the later proofs.

Moreover, the construction of a minimal bad sequence could be significantly simplified by step-wise narrowing down the set of all bad sequences using the notions of *equal up to*, *filtering* sets of infinite sequences with respect to a given infinite sequence, *column* of a set of infinite sequences, and *minimal element* of a set (only the first of which was present in my previous work).

4.6. Higman's Lemma

Before Higman's lemma for almost-full relations is stated formally, a construction that extends a given order on elements to an order on lists is required: *homeomorphic embedding*. The set of lists over elements from a set A , written A^* , is defined inductively:

$$\frac{}{[] \in A^*} \quad \frac{x \in A \quad xs \in A^*}{x \cdot xs \in A^*}$$

The size of a list is measured by its length (that is, number of elements). Homeomorphic embedding on lists, for a given base order \preceq , is defined inductively by the rules

$$\frac{}{[] \preceq^* ys} \quad \frac{xs \preceq^* ys}{xs \preceq^* y \cdot ys} \quad \frac{x \preceq y \quad xs \preceq^* ys}{x \cdot xs \preceq^* y \cdot ys}$$

(In this article the notation \preceq^* is used consistently to denote list-embedding with respect to the base order \preceq and is not to be confused with the reflexive and transitive closure of a relation.) Intuitively, it might be easier to think about homeomorphic embedding on lists as follows: a list xs is embedded in a list ys if and only if xs can be obtained from ys by dropping elements and replacing elements with arbitrary smaller ones (with respect to the base order). An important special case of embedding is $=^*$, which is called the *sublist relation*. Then, $xs =^* ys$ if and only if the list xs can be obtained from the list ys by dropping elements.

The *mbs* locale can be instantiated by taking A^* for its parameter A and the length of lists as their size. Thus,

$$\neg af_{A^*}(\preceq^*) \implies \exists m \in \mathcal{B}. \forall g. g \preceq^\omega m \longrightarrow good_{\preceq^*}(g)$$

which allows us to prove Higman's lemma for almost-full relations.

Lemma 4.6.1. *Homeomorphic embedding with respect to an almost-full relation \preceq on a set A , is almost-full on the set of finite lists over A . In Isabelle: $af_A(\preceq) \implies af_{A^*}(\preceq^*)$.*

Proof. Assume $af_A(\preceq)$ but $\neg af_{A^*}(\preceq^*)$, for the sake of a contradiction. Then there is a minimal bad sequence m . All lists in m are non-empty (since otherwise m would be good). Hence, there are sequences h and t of heads and tails of m (that is, $m \ i = h \ i \cdot t \ i$).

Clearly, $h \in A^\omega$ and thus, by Lemma 4.3.1, there is a strictly monotone index-mapping such that h_φ is a \preceq -homogeneous sequence. Moreover, t_φ is bad, since otherwise m would be good.

Let n abbreviate $\varphi \ 0$ and c be the combination of the infinite sequences m and t , defined by $c \ i \stackrel{\text{def}}{=} \text{if } i < n \text{ then } m \ i \text{ else } t \ (\varphi \ (i - n))$ (that is, c is the same as t_φ , but prepended by the first n elements of m). Then c is bad, since otherwise a contradiction is obtained as follows: Assume c is good. Then there are $i < j$ such that $c \ i \preceq^* c \ j$. Now, analyze the following cases:

- **case** ($j < n$). Then $m \ i \preceq^* m \ j$, contradicting the badness of m .
- **case** ($n \leq i$). Let $i' = i - n$ and $j' = j - n$. Then $i' < j'$ and $t_\varphi \ i' \preceq^* t_\varphi \ j'$, contradicting badness of t_φ .

- **case** ($i < n$ and $n \leq j$). Let $j' = j - n$. Then $m\ i \preceq^* t\ (\varphi\ j')$ (from $c\ i \preceq^* c\ j$). Moreover, $m\ i \preceq^* m\ (\varphi\ j')$ (by the second clause of the inductive definition of embedding). Together with $i < \varphi\ j'$, this contradicts the badness of m .

Thus, c is bad. Furthermore, $\forall i < n. c\ i = m\ i$ and $|c\ n| < |m\ n|$, and thus c is good (since m is minimal): A contradiction, concluding the proof. \square

This result can be easily extended to wqos.

Higman’s Lemma 1. *Whenever a set A is well-quasi-ordered by a relation \preceq , then the set of finite lists over A is well-quasi-ordered by homeomorphic embedding with respect to \preceq^* . In Isabelle: $wqo_A(\preceq) \implies wqo_A^*(\preceq^*)$.*

Proof. For transitivity of \preceq^* (under the assumption that \preceq is transitive), refer to lemma `list_emb_trans` in theory `HOL-Library.Sublist`. Together with Lemma 4.6.1, this yields Higman’s lemma. \square

Comparison to Previous Work By employing Lemma 4.3.1, the slightly tedious reasoning about the non-existence of an infinite bad sequence “of special shape” (which is also to be found in Nash-Williams’ original proof) could be completely avoided. This change made it possible to shorten the previous 166-line proof to more reasonable 66 lines.

4.7. The Tree Theorem

The tree theorem is for finite trees, what Higman’s lemma is for finite lists. However, whereas for finite lists, their representation inside Isabelle/HOL is quite unambiguous and the existing data type is generally applicable; this is not so much the case for finite trees. Consider the following two data types

```
datatype 'a t = Tree 'a ('a t list)
datatype 'a t' = E | N 'a ('a t' list)
```

or the type of first-order terms

```
datatype ('f, 'v) term = Var 'v | Fun 'f (('f, 'v) term list)
```

also a kind of finite tree (and more importantly, one of the types to which the tree theorem is applied, in order to formalize the fact that the Knuth-Bendix order is a simplification order [142]). Restricting the tree theorem to a specific data type would strongly restrict its applicability. Therefore, again Isabelle/HOL’s locale mechanism is employed. This time, for a locale `kruskal_tree` that fixes the following constants (see theory `Well_Quasi_Orders.Kruskal` for details):

- A set $\mathcal{F} :: ('b \times \text{nat}) \text{ set}$ representing the signature over which trees are built.
- A function $mk :: 'b \Rightarrow 'a \text{ list} \Rightarrow 'a$ that is used to construct a finite tree from a given node and a given list of finite trees.

4. Certified Kruskal's Tree Theorem

- A function $\text{root}::'a \Rightarrow 'b \times \text{nat}$ that extracts the root node together with its arity from a given tree.
- A function $\text{args}::'a \Rightarrow 'a \text{ list}$ that extracts the list of arguments (direct subtrees) from a given tree.
- As well as the set $\mathcal{T}(\mathcal{F})::'a \text{ set}$ of well-formed trees with respect to the signature \mathcal{F} .

These constants are required to satisfy the following assumptions (thereby turning mk into kind of a data type constructor with extractors root and args):

$$\llbracket t \in \mathcal{T}(\mathcal{F}); s \in \text{set } (\text{args } t) \rrbracket \implies |s| < |t| \quad (\text{F1})$$

$$(f, |ts|) \in \mathcal{F} \implies \text{root } (\text{mk } f \ ts) = (f, |ts|) \quad (\text{F2})$$

$$(f, |ts|) \in \mathcal{F} \implies \text{args } (\text{mk } f \ ts) = ts \quad (\text{F3})$$

$$t \in \mathcal{T}(\mathcal{F}) \implies \text{mk } (\text{fst } (\text{root } t)) (\text{args } t) = t \quad (\text{F4})$$

$$t \in \mathcal{T}(\mathcal{F}) \implies \text{root } t \in \mathcal{F} \quad (\text{F5})$$

$$t \in \mathcal{T}(\mathcal{F}) \implies |\text{args } t| = \text{snd } (\text{root } t) \quad (\text{F6})$$

$$\llbracket t \in \mathcal{T}(\mathcal{F}); s \in \text{set } (\text{args } t) \rrbracket \implies s \in \mathcal{T}(\mathcal{F}) \quad (\text{F7})$$

That is, the size of a direct subtree of a well-formed tree is strictly smaller than the size of the tree itself (F1); mk is injective when applied to a number of arguments corresponding to the arity of a node (that is, $\llbracket (f, |ss|) \in \mathcal{F}; (g, |ts|) \in \mathcal{F} \rrbracket \implies (\text{mk } f \ ss = \text{mk } g \ ts) = (f = g \wedge ss = ts)$; (F2) and (F3)); and mk , root , and args interact “as expected” on well-formed trees ((F4), (F5), (F6), and (F7))

Homeomorphic embedding on (well-formed) finite trees is defined inductively by the two rules:

$$\frac{(f, m) \in \mathcal{F} \quad |ts| = m \quad \text{set } ts \subseteq \mathcal{T}(\mathcal{F}) \quad t \in \text{set } ts \quad s \preceq_{\text{emb}} t}{s \preceq_{\text{emb}} \text{mk } f \ ts}$$

$$\frac{\begin{array}{cccc} (f, m) \in \mathcal{F} & (g, n) \in \mathcal{F} & |ss| = m & |ts| = n \\ \text{set } ss \subseteq \mathcal{T}(\mathcal{F}) & \text{set } ts \subseteq \mathcal{T}(\mathcal{F}) & (f, m) \preceq (g, n) & ss \preceq_{\text{emb}}^* ts \end{array}}{\text{mk } f \ ss \preceq_{\text{emb}} \text{mk } g \ ts}$$

The first rule subsumes what is often called the *subterm property* (that is, a proper subtree of a well-formed tree is also in the embedding relation). The second rule states that the nodes of a tree may be replaced by smaller ones with respect to \preceq and their arguments by smaller ones with respect to list-embedding where the underlying order is \preceq_{emb} .

To instantiate the mbs locale, take $\mathcal{T}(\mathcal{F})$ for its parameter \mathbf{A} . Thus,

$$\neg \text{af}_{\mathcal{T}(\mathcal{F})}(\preceq_{\text{emb}}) \implies \exists m \in \mathcal{B}. \forall g. g \triangleleft^\omega m \longrightarrow \text{good}_{\preceq_{\text{emb}}}(g)$$

Finally, the tree theorem for almost-full relations can be stated and proved (see theory `Well_Quasi_Orders.Kruskal` for details).

Theorem 4.7.1. *Homeomorphic embedding with respect to an almost-full relation \preceq on a set \mathcal{F} , is almost-full on the set of finite trees over \mathcal{F} . In Isabelle: $af_{\mathcal{F}}(\preceq) \implies af_{\mathcal{T}(\mathcal{F})}(\preceq_{\text{emb}})$*

Proof. Assume that \preceq is almost-full on \mathcal{F} but, for the sake of a contradiction, \preceq_{emb} is not almost-full on $\mathcal{T}(\mathcal{F})$. Then, by Theorem 4.5.7, there is a minimal bad sequence m such that any smaller sequence with respect to \triangleleft^ω is good. Moreover, there are sequences r and a of roots and arguments of m (that is, $m\ i = mk\ (\text{fst}\ (r\ i))\ (a\ i)$). Let A denote the set of all trees occurring in a (that is, the set of arguments of all $m\ i$).

Then it is shown that \preceq_{emb} is almost-full on A . To this end, suppose the contrary. Thus, there is a sequence $s \in A^\omega$ which is bad. Let n be the least index such that there is some element $s\ k$ that is an argument of $m\ n$ (that is, $s\ k \in \text{set}\ (a\ n)$ for some k). Let c be the combination of m and s , defined by

$$c\ i \stackrel{\text{def}}{=} \text{if } i < n \text{ then } m\ i \text{ else } s\ (k + (i - n))$$

Clearly, $c\ i = m\ i$ for all $i < n$ and $c\ i = s\ (k + (i - n))$, otherwise. Then c is bad, since otherwise a contradiction is obtained as follows: Assume c is good. Then, there are $i < j$ such that $c\ i \preceq_{\text{emb}} c\ j$. Now analyze the following cases:

- **case** ($j < n$). Then $m\ i \preceq_{\text{emb}} m\ j$, contradicting the badness of m .
- **case** ($n \leq i$). Let $i' = k + (i - n)$ and $j' = k + (j - n)$. Then $i' < j'$ and $s\ i' \preceq_{\text{emb}} s\ j'$, contradicting the badness of s .
- **case** ($i < n$ and $n \leq j$). Let $j' = k + (j - n)$. Then $m\ i \preceq_{\text{emb}} s\ j'$. Thus, there is some index $l \geq n$ such that $s\ j' \in \text{set}\ (a\ l)$, which in turn implies $m\ i \preceq_{\text{emb}} m\ l$. Together with $i < l$, this contradicts the badness of m .

Thus term c is bad. Since also $c \triangleleft^\omega m$ (since $c\ n$ is an argument of $m\ n$), we obtain the desired $af_A(\preceq_{\text{emb}})$.

Now, by Lemma 4.6.1 and Lemma 4.3.1 we obtain a strictly monotone index-mapping φ such that $\varphi\ i < \varphi\ j$ and $a_\varphi\ i \preceq_{\text{emb}}^* a_\varphi\ j$ for all $i < j$. Moreover, $r_\varphi\ i \in \mathcal{F}$ for all i and thus there are indices $i < j$ such that $r_\varphi\ i \preceq r_\varphi\ j$. Together, this implies $m_\varphi\ i \preceq_{\text{emb}} m_\varphi\ j$, contradicting the badness of m . \square

Kruskal's Tree Theorem 2. *Whenever a set \mathcal{F} is well-quasi-ordered by a relation \preceq , then the set of finite trees over \mathcal{F} is well-quasi-ordered by homeomorphic embedding with respect to \preceq_{emb} . In Isabelle: $wqo_{\mathcal{F}}(\preceq) \implies wqo_{\mathcal{T}(\mathcal{F})}(\preceq_{\text{emb}})$.*

Proof. By induction on the definition of embedding, it can be shown that \preceq_{emb} is transitive whenever the base order \preceq is. Together with Theorem 4.7.1 this yields the tree theorem. \square

Notes As in my previous work [133], the definition of homeomorphic embedding on trees could have ignored arities of nodes and in turn well-formedness of trees. This would constitute a slightly simpler definition and still allow us to obtain the tree theorem.

4. Certified Kruskal's Tree Theorem

Moreover, as in my previous work, closure under context and transitivity could have been built-in. However, note that every extension of an almost-full relation is again an almost-full relation (an easy consequence of the definition of almost-full). Thus it seems desirable to have an embedding relation that is as small as possible. Since the proof of the tree theorem goes through with the current version, I went with it. But considering arities does not only make embedding potentially smaller, it is also necessary for some applications as shown in the next section.

Comparison to Previous Work Again, by employing Lemma 4.3.1, the very tedious reasoning about the non-existence of an infinite bad sequence “of special shape” (which is also to be found in Nash-Williams’ original proof) could be avoided completely. Thereby shortening the original 188-line proof to 90 lines and, more importantly, making the argument much simpler.

4.8. Examples

In this section we consider concrete instances of the *kruskal_tree* locale for the following data types:

- Rose trees: **datatype** 'a tree = Node 'a ('a tree list)
- First-order terms: **datatype** ('f, 'v) term = Var 'v | Fun 'f (('f, 'v) term list)
- “Arithmetic” expressions involving addition of variables and constants:
datatype 'a exp = V 'a | C nat | Plus ('a exp) ('a exp)

For rose trees consider the selector functions *node* (Node *f ts*) = (*f*, |*ts*|) and *succs* (Node *f ts*) = *ts*, as well as the inductive set of trees over a given set of nodes *A*:

$$\frac{f \in A \quad \forall t \in \text{set } ts. t \in \text{trees } A}{\text{Node } f \ ts \in \text{trees } A}$$

The *kruskal_tree* locale is easily instantiated by

interpretation *kruskal_tree* "(A × UNIV)" Node node succs "(trees A)"

and we obtain the following variant of the tree theorem

$$wqo_{A \times \text{UNIV}}(\preceq) \implies wqo_{\text{trees } A}(\preceq_{\text{emb}}).$$

However, arities are actually not interesting (since nodes in a rose tree may have arbitrarily many successors) thus it might be desirable to start from a base order \preceq on *A* (instead of $A \times \text{UNIV}$). This is easily possible by noting that the full relation ($x \preceq y$ for all *x* and *y*) is a wqo on any set and invoking Dickson’s lemma.

For first-order terms consider the selector functions *root* (Fun *f ts*) = (*f*, |*ts*|) and *args* (Fun *f ts*) = *ts*, as well as the inductively defined set of ground terms over a signature *F*:

$$\frac{(f, n) \in F \quad |ts| = n \quad \forall s \in \text{set } ts. s \in \mathcal{T}(F)}{\text{Fun } f \ ts \in \mathcal{T}(F)}$$

Again, the `kruskal_tree` locale is easily instantiated by

interpretation `kruskal_tree` \mathcal{F} *Fun root args* " $\mathcal{T}(\mathcal{F})$ "

and we obtain the following variant of the tree theorem

$$wqo_{\mathcal{F}}(\preceq) \implies wqo_{\mathcal{T}(\mathcal{F})}(\preceq_{\text{emb}}).$$

For arithmetic expressions consider the constructor function

`mk (v x) [] = V x`
`mk (c n) [] = C n`
`mk p [a, b] = Plus a b`

the root selector function

`rt (V x) = (v x, 0)`
`rt (C n) = (c n, 0)`
`rt (Plus a b) = (p, 2)`

and the argument selector function

`ags (V x) = []`
`ags (C n) = []`
`ags (Plus a b) = [a, b]`

where

datatype `'a symb` = `v 'a` / `c nat` / `p`.

Moreover, consider the inductively defined set of arithmetic expressions:

$$\frac{}{V\ x \in \text{exps}} \quad \frac{}{C\ n \in \text{exps}} \quad \frac{a \in \text{exps} \quad b \in \text{exps}}{Plus\ a\ b \in \text{exps}}$$

For the signature $\Sigma \stackrel{\text{def}}{=} \{(v\ x, 0) \mid x \geq 0\} \cup \{(c\ n, 0) \mid n \geq 0\} \cup \{(p, 2)\}$ (which ensures that constructors are applied to the correct number of arguments), the `kruskal_tree` locale can be instantiated by

interpretation `kruskal_tree` Σ *mk rt ags exps*

and we obtain the following variant of the tree theorem

$$wqo_{\Sigma}(\preceq) \implies wqo_{\text{exps}}(\preceq_{\text{emb}}).$$

4.9. Conclusion and Related Work

An Isabelle/HOL formalization of three important results from combinatorics was presented: Dickson’s lemma, Higman’s lemma, and Kruskal’s tree theorem. The formalized proofs are reasonably simple and the tree theorem is presented in a general version that is applicable to several instances.

Parts of the presented formalization were used by Wu et al. [179] to formalize a proof of: *For every language A , the languages of sub- and superstrings of A are regular.* (Details are given in the corresponding journal article [180].)

Moreover, the presented formalization of the tree theorem is employed for a proof that the Knuth-Bendix order is a simplification order [142].

There are formalizations of Higman’s lemma in Isabelle/HOL by Berghofer [16] and using other proof assistants by Murthy [93], Fridlender [39], Herbelin [56], Seisenberger [125], and Martín-Mateos et al. [84].

Since Berghofer’s work was also conducted using Isabelle/HOL, some comments on the relation to the presented work are in order. First note that Berghofer’s formalization is constructive (based on an earlier proof by Coquand and Fridlender in an unpublished manuscript entitled *A Proof of Higman’s Lemma by Structural Induction*). Furthermore, it is restricted to a two letter alphabet (and Berghofer notes that “*the extension of the proof to an arbitrary finite alphabet is not at all trivial*”). Also noteworthy is that the focus of Berghofer’s work is on program extraction and the computational behavior of the resulting program. In contrast, the presented work constitutes a formalization of Higman’s lemma without restricting the alphabet, that is, the alphabet may be infinite as long as it is equipped with a wqo (which is always the case for finite alphabets).

An intuitionistic proof of Kruskal’s tree theorem is presented in [168]. However, to the best of the author’s knowledge the presented work constitutes the first formalization of the tree theorem in a proof assistant ever.

The tree theorem is a special case of the graph minor theorem, which was proved by Robertson and Seymour in a series of twenty papers [120, 121]. The size of this (pen and paper) proof alone makes a formalization interesting. However, an extension of the current proof would constitute significant extra effort and it is unclear whether the minimal bad sequence argument could be applied at all. Thus, we leave it as future work.

5. A Framework for Developing Stand-Alone Certifiers

Publication Details

Christian Sternagel and René Thiemann. A Framework for Developing Stand-Alone Certifiers. *Electronic Notes in Theoretical Computer Science* 312:51–67, 2014
[doi:10.1016/j.entcs.2015.04.004](https://doi.org/10.1016/j.entcs.2015.04.004)

Christian Sternagel and René Thiemann. Certification Monads. *The Archive of Formal Proofs*, 2014
`afp:Certification_Monads`

Christian Sternagel and René Thiemann. Haskell’s Show Class in Isabelle/HOL. *The Archive of Formal Proofs*, 2014 (last update: 2015)
`afp:Show`

Christian Sternagel and René Thiemann. XML. *The Archive of Formal Proofs*, 2014
`afp:XML`

Abstract

Current tools for automated deduction are often powerful and complex. Due to their complexity there is a risk that they contain bugs and thus deliver wrong results. To ensure reliability of these tools, one possibility is to develop certifiers which check the results of tools with the help of a trusted proof assistant. We present a framework which illustrates the essential steps to develop stand-alone certifiers which efficiently check generated proofs outside the employed proof assistant. Our framework has already been used to develop certifiers for various properties, including termination, confluence, completion, and tree automata related properties.

5.1. Introduction

Due to their increased power, automated provers like SAT-solvers, SMT-solvers, automated first-order theorem provers, model checkers, termination provers, etc., are becoming increasingly popular for software verification. However, the complexity of these provers comes with the risk of bugs that cause wrong answers (e.g., a termination claim for a nonterminating program). Hence, the reliability of the generated answer is usually reduced whenever the complexity of the prover is increased.

5. A Framework for Developing Stand-Alone Certifiers

For reliability it is therefore of major importance to validate answers. To this end, provers not only have to deliver a binary answer like SAT or UNSAT, but must additionally provide justification in form of a certificate, which usually depends on the domain of the prover. It might be a satisfying assignment or a natural deduction proof for a SAT-solver, a well-founded measure or looping sequence for a termination prover, etc. *Certification*—i.e., validation of the certificate—can be applied to recover the desired degree of reliability for powerful but complex automated provers.

In this paper we present a concrete framework for conveniently developing highly reliable, efficient, and easy-to-use certifiers. To this end, in Section 5.2, we first discuss various alternatives on how to perform certification. Then, our framework is introduced step-by-step. We discuss error handling in Section 5.3, error generation in Section 5.4, parsing in Section 5.5, and proving soundness of the final certifier in Section 5.6. We conclude in Section 5.7.

We illustrate our framework by means of a running example. Since this example poses only a quite simple certification task, we shortly want to mention that the framework has already successfully been applied for much more complex certification tasks where the certifier itself consists of over 35,000 lines of Haskell code.

In the following, everything is illustrated for the proof assistant Isabelle/HOL [103], but most parts should easily be adaptable to similar proof assistants like Coq [18] or PVS [114], provided they support code generation mechanisms. By *code generation* we mean an automatic and trusted translation from functions defined in the logic of the used proof assistant into actual program code. For example, Isabelle’s code generator supports Standard ML and Haskell (amongst others) as target languages. We refer to the work of Haftmann and Nipkow [52] for more details.

All components of the framework have been made available in the archive of formal proofs [143, 145, 148], and the sources of the running example are freely available under <http://cl-informatik.uibk.ac.at/software/ceta/framework>. Some parts of this work have already been presented earlier [165], but in a much less complete and detailed form.

Our approach is aimed to ease the construction of verified checkers for certifying algorithms [21]. In the running example this is demonstrated for Post’s correspondence problem, while in earlier work [165] we employed the same methodology to build the checker *CeTA* for termination provers (in fact, the framework we present here was distilled from those parts of *CeTA* we deemed generally useful).

5.2. Certification

Certification of an automatically generated proof (asserting that some input has some property) can be performed in several ways, shortly discussed in the following.

As a running example, we consider Post’s Correspondence Problem (PCP) [119]. Given an alphabet Σ , a PCP instance p is a set of pairs of words over Σ . It is solvable iff there is a nonempty list $[(x_1, y_1), \dots, (x_n, y_n)]$ of pairs of words such that each $(x_i, y_i) \in p$ and $x_1 \dots x_n = y_1 \dots y_n$.

It is well-known that solvability of PCP instances is undecidable in general. We want to validate certificates for solvable PCP instances. This is a trivial certification task, but can be used to illustrate various design choices and challenges in the process of developing a certifier. We assume that the certificate numbers each pair of words in p and provides the solution as a list of numbers.

5.2.1. Human Inspection

Clearly, humans can check certificates, provided that certificates are rendered in a human readable form. For example, the PCP instance $p = \{0 : (A, ABA), 1 : (AB, BB), 2 : (BAA, AA)\}$ and the certificate in form of the solution 0, 2, 1, 2 is rendered in the following table.

0	2	1	2
A_1	$B_2A_3A_4$	A_5B_6	$B_7A_8A_9$
$A_1B_2A_3$	A_4A_5	B_6B_7	A_8A_9

It is easy to see from this table that p is solvable: just check whether the columns correspond to word pairs in p . Moreover, the subscripts 1, \dots , 9 for the position within the word help when checking that both rows contain the same word: just check that both rows contain the subscripts 1 to 9 in ascending order and each number is attached to the same letter in both rows.

However, human inspection is clearly error-prone and therefore not the best method for certification. For example, consider the PCP instance

$$p' = \{0 : (AAB, A), 1 : (AB, ABB), 2 : (AB, BAB), 3 : (BA, AAB)\}$$

for which the shortest solution is 1, 3, 2, 3, 3, 1, 0, 1, 3, 2, 3, 2, 3, 3, 2, 3, 3, 1, 0, 3, 3, 1, 0, 2, 3, 0, 0, 2, 3, 3, 3, 1, 0, 1, 0, 0, 0, 2, 3, 2, 3, 0, 1, 0, 3, 3, 1, 0, 3, 0, 0, 2, 3, 0, 0, 2, 0, 0, 2, 0, 1, 0, 3, 0, 0, 2. Checking this solution by hand is at least tedious. When we move from PCP to more complex certificates—whose validation involves elaborate computations—human inspection is not feasible any more.

5.2.2. Certification via Programs

Instead of human inspection, we can write a program that checks all proof steps mentioned in the certificate.

This is often not too complex—in comparison to writing the program which has to produce the proof—and also possibly a good option for getting a certifier in case of simple certificates like the ones for solvable PCP instances. Nevertheless, this approach also has some severe drawbacks: e.g., if checking certificates requires some complicated decision procedure, then the program which implements this decision procedure is itself complex and may be buggy. Hence, the reliability of the certifier decreases with its complexity.

Another problem is the dependence on potentially flawed paper proofs and inconsistent assumptions: for example, theorems as they are stated in papers (and implemented in

5. A Framework for Developing Stand-Alone Certifiers

```

datatype letter = A | B

type_synonym word = "letter list"
type_synonym pcp_problem = "(word × word) set"

definition solvable :: "pcp_problem ⇒ bool"
where
  "solvable pcp ⟷ (∃ pair_list.
    set pair_list ⊆ pcp ∧
    pair_list ≠ [] ∧
    concat (map fst pair_list) = concat (map snd pair_list))"

definition p' :: pcp_problem
where
  "p' =
    {([A, A, B], [A]),
     ([A, B], [A, B, B]),
     ([A, B], [B, A, B]),
     ([B, A], [A, A, B])}"

```

Figure 5.1.: Specifying Input and Solvability

tools) might be wrong; and when combining methods from different papers, it might happen, that the methods make slightly different but incompatible assumptions where this incompatibility might remain undetected. For example, [25] contains some inconsistent assumptions that have only been spotted in [166, §5] during the development of a certifier—in this case all problems could be repaired, but this is not always the case.

An example of this approach is the algorithmic library LEDA (which was extended to use verified checkers by Alkassar et al.[2]).

5.2.3. Certification via Proof Assistants

To increase reliability, we can make use of LCF-style [49, 50, 115] proof assistants, i.e., proof assistants whose soundness relies on a small trusted kernel and where definitional packages allow us to write more high-level proofs which are then broken down into kernel-primitives without adding new axioms.

When using proof assistants, one first has to model the property of interest. Whether the model corresponds to the real property that one is interested in, has to be carefully checked by humans.

However, afterwards one can turn the certificate into a proof script which can then be checked by the proof assistant, yielding the desired high degree of reliability.

As an example, consider the following Isabelle/HOL [103] formalization of PCP. It starts with the specification of PCP instances and their solvability, and defines one instance p' (corresponding to example p' mentioned in Section 5.2.1), cf. Figure 5.1.

In the definition of *solvable*, the condition $\text{set pair_list} \subseteq \text{pcp}$ asserts that all pairs in

```

fun pair_of_index :: "nat ⇒ word × word"
where
  "pair_of_index i = nth
    [([A, A, B], [A]),
     ([A, B], [A, B, B]),
     ([A, B], [B, A, B]),
     ([B, A], [A, A, B])] i"

lemma pcg_solvable: "solvable p'"
  apply (unfold solvable_def p'_def)
  apply (rule exI [of _ "(map pair_of_index
    [1,3,2,3,3,1,0,1,3,2,3,2,3,3,2,3,3,1,0,3,3,1,0,2,3,0,0,2,3,3,3,1,0,
     1,0,0,0,2,3,2,3,0,1,0,3,3,1,0,3,0,0,2,3,0,0,2,0,0,2,0,1,0,3,0,0,2])")])
  apply simp
done

```

Figure 5.2.: Proving Solvability

the list are contained in the PCP instance, and in the equality test `concat ... = concat ...`, `map fst pair-list` and `map snd pair-list` projects the list of pairs of words into the list of words for the left- and right-hand sides of the pairs, respectively.

After the specification, solvability (of p') can be proven by the script in Figure 5.2. First, the function `pair-of-index` is defined, which maps indices to corresponding word-pairs of p' . Then, the proof of solvability is performed: first, the solution from the certificate is used as witness for the existential quantifier, and then Isabelle's simplifier is invoked to check that all conditions of a valid solution are met.

This approach has several advantages, but also some disadvantages:

- + The validation is highly reliable.
- + One can perform a shallow embedding, i.e., features of the proof assistant may be used for modeling the given input problem and for establishing the proof. As a consequence it is often possible to specify the model succinctly and readable, and it also eases the generation of proofs.

In the case of PCP, as example for shallow embedding we created a datatype for letters which is specific to the PCP instance p' . Moreover, we used Isabelle's simplifier to conclude validity of a solution. Similarly, one might use built-in operators or quantifiers like λ , \forall , etc., to model the input problem; or one might invoke some powerful routines from the proof assistant to discharge proof obligations, like an arithmetic solver, etc.

- + If the property of interest is related to proof obligations in the proof assistant itself, then certification allows safe integration of untrusted automated tools into the proof assistant in order to increase the degree of automation.

For example, the Sledgehammer tool of Isabelle [22] can solve open proof goals by invoking external automated theorem provers, where the generated proofs are

5. A Framework for Developing Stand-Alone Certifiers

then replayed within the proof assistant with the help of *metis*, an Isabelle internal prover acting as a certifier.

- For certification, one needs to have the proof assistant installed and started. Moreover, checking proofs within the proof assistant is usually slower than just executing a program as in Section 5.2.2.
- If a certificate is not accepted, then the proof assistant gets stuck on some intermediate proof obligation, potentially with some error message. Some knowledge of the proof assistant may be required in order to understand why the certificate was rejected. For example, for understanding rejected PCP certificates, it might be required to understand Coq-, or Isabelle-, or PVS-scripts.
- Changes in the proof assistant are only detected at run-time. E.g., if Isabelle would change the configuration of the simplifier, then it might be the case that the simplifier invocation in Figure 5.2 no longer succeeds.

Successful examples of this approach are the two termination proof certifiers *Coccinelle/CiME* [26], and *CoLoR/Rainbow* [20]. Here, *Coccinelle* and *CoLoR* are Coq-libraries on termination of rewrite systems, i.e., they define the notion of termination, and contain soundness theorems of some termination criteria. And *CiME* and *Rainbow* are tools which turn the certificates from the automated termination tools into proof scripts, which then apply suitable tactics based on the theorems that are available in the libraries.

5.2.4. Certification via Programs and Proof Assistants

Finally, we also present an approach which combines the best of Sections 5.2.2 and 5.2.3. The basic idea is to write a program *check-prop* :: *input* \Rightarrow *certificate* \Rightarrow *bool* which efficiently checks certificates as in Section 5.2.2, but is completely written within a proof assistant. As a result, we can develop a model of the desired property *P* within the proof assistant, in combination with a static soundness proof of *check-prop*:

$$\text{check-prop input certificate} \Longrightarrow P \text{ input} \tag{5.1}$$

Hence, we get the high reliability of Section 5.2.3.

Once this is established one just needs to execute *check-prop*. This can be done within the proof assistant via reflection. Alternatively, one can invoke the code generator of the proof assistant to get *check-prop* as stand-alone program, which can then be conveniently and efficiently executed by everyone, without even having to install the proof assistant. As an example, consider Figure 5.3 which contains a checker for solvable PCP instances, where in the last line the full checker is made available as Haskell code via Isabelle’s code generator [52].

With the described approach, one can overcome all disadvantages which are mentioned at the end of Section 5.2.3, at the cost of not being able to perform shallow embedding. Therefore, we cannot create suitable datatypes like *letters* on the fly as in Section 5.2.3, but instead use a polymorphic type for the alphabet with type variable *'a* (we could


```

type_synonym 'a word = "'a list"
type_synonym 'a pcp_problem = "('a word × 'a word) set"

definition solvable :: "'a pcp_problem ⇒ bool"
where
  "solvable pcp ⟷ (∃ pair_list.
    set pair_list ⊆ pcp ∧
    pair_list ≠ [] ∧
    concat (map fst pair_list) = concat (map snd pair_list))"

type_synonym 'a pcp_problemI = "('a word × 'a word) list"

fun pair_of_index :: "'a pcp_problemI ⇒ nat ⇒ 'a word × 'a word"
where
  "pair_of_index pcp i = nth pcp i"

type_synonym pcp_certificate = "nat list"

fun check_solvable :: "'a pcp_problemI ⇒ pcp_certificate ⇒ bool"
where
  "check_solvable pcp solution =
    (let pair_list = map (pair_of_index pcp) solution in
     list_all (λ i. i < length pcp) solution ∧
     solution ≠ [] ∧
     concat (map fst pair_list) = concat (map snd pair_list))"

lemma check_solvable:
  assumes check: "check_solvable pcp solution"
  shows "solvable (set pcp)"
proof -
  let ?pair_list = "map (pair_of_index pcp) solution"
  have "concat (map fst ?pair_list) = concat (map snd ?pair_list)" using check
  by simp
  moreover have "?pair_list ≠ []" using check by simp
  moreover have "set ?pair_list ⊆ set pcp" using check by (auto simp add:
list_all_iff)
  ultimately show ?thesis
  unfolding solvable_def by (intro exI [of _ ?pair_list]) auto
qed

export_code check_solvable in Haskell

```

Figure 5.3.: A First Certified Checker for PCP

5. A Framework for Developing Stand-Alone Certifiers

also have chosen strings or numbers, etc.). As a further consequence, all routines within *check-prop* have to be programmed as such, i.e., if we need an arithmetic solver, we need to program it and prove it correct, and there is no possibility to just invoke the arithmetic solver that may be available via some tactic in the proof assistant.

In the running example, let us shortly describe the differences between Figure 5.2 and Figure 5.3. The latter solution cannot encode the concrete PCP instance into *pair-of-index* but has to pass it as parameter. It further uses a new type for representing PCP instances in an executable form, namely lists of word-pairs instead of sets of word-pairs: *pcp-problemI*. Moreover, conditions that have previously been discharged by the simplifier are now explicit in the *check-solvable* function, e.g., the check via *list-all* that all indices within the solution point to valid word-pairs.

In the remainder of this paper, we will illustrate how to improve this basic version of a *check-prop*-program.

5.3. Error Handling

At the moment, the type of *check-prop* is *input* \Rightarrow *certificate* \Rightarrow *bool*. That is, the return value just provides one bit of information. Whereas for accepted certificates this is sufficient, for rejected ones we are often interested in the reason for rejection.

With the current approach (*check-solvable* from Figure 5.3), we are even worse off in case of rejection than in Section 5.2.3 (where we were required to interpret error messages from the proof assistant), since now we only obtain the resulting value: *False*.

Hence, our next goal is to extend *check-prop* in a way that it returns error messages in case of rejection. Moreover, this should be done without much overhead and especially it should not clutter the soundness proof of *check-prop*.

We propose to use the error monad represented by Isabelle’s sum type

```
datatype 'a + 'b = Inl 'a | Inr 'b
```

where errors are indicated by *Inl* and proper results by *Inr*. Booleans are now replaced by type *'e check* which is an abbreviation for *'e + unit*. Then *Inr ()* corresponds to *True* and *Inl e* to *False* enriched by the error message *e*.

More general check functions may also return new results *Inr x* instead of plain *()* in case of success. For example, a function for checking some inference rule might fail if the preconditions of the inference rule are not met, and return the new proof obligations arising from applying the rule, otherwise.

In the following, we focus on *'e check* which replaces the Boolean return type of *check-prop*. We provide the following functionality to ease the transition from Booleans to the error monad.

- *inspection*: a function *isOk* which tests whether a given monadic value is an error or not. Consequently, soundness proofs like (5.1) are now reformulated as

$$isOk (check-prop\ input\ certificate) \implies P\ input \quad (5.2)$$

```

fun check_solvable :: "'a pcp_problemI  $\Rightarrow$  pcp_certificate  $\Rightarrow$  string check"
where
  "check_solvable pcp solution = do {
    check_all ( $\lambda$  i. i < length pcp) solution
    <+? ( $\lambda$  i. ''index i invalid'');
    let pair_list = map (pair_of_index pcp) solution;
    check (solution  $\neq$  []) ''solution must not be empty'';
    check (concat (map fst pair_list) =
      concat (map snd pair_list)) ''resulting words are not equal''
  } <+? ( $\lambda$  s. ''problem in ensuring satisfiability of PCP: '' @ s)"

lemma check_solvable:
  assumes check: "isOK (check_solvable pcp solution)"
  shows "solvable (set pcp)"

```

Figure 5.4.: A Certified Checker with Error Messages

- *assertions*: for asserting basic properties, we provide the function `check::bool \Rightarrow 'e \Rightarrow 'e + unit`, where `check b e = (if b then Inr () else Inl e)`, i.e., the asserted property is coupled with an error message.
- *combinators*: we provide several combinators like monadic bind (`>>::'e + 'a \Rightarrow ('a \Rightarrow 'e + 'b) \Rightarrow 'e + 'b`) (acting as short-circuited conjunction) and `check_all :: ('a \Rightarrow bool) \Rightarrow 'a list \Rightarrow 'a check` (which behaves like \forall on lists in case of success, and returns the first element for which the given predicate fails, otherwise). Moreover, specifically for monadic bind, we extended Isabelle's parser in a way that it supports Haskell's `do`-notation, facilitating writing of readable check functions.
- *error messages*: there are operators for changing error messages like `<+?::'e + 'a \Rightarrow ('e \Rightarrow 'f) \Rightarrow 'f + 'a` which takes a function that is used to modify the error message of the given monadic value. Since modification takes only place in case of error, this operation has no impact below `isOK`.
- *proving*: we configured Isabelle in a way that most of the time the simplifier can easily eliminate monadic overhead and error message processing.

At this point, it is quite easy to integrate error messages into our PCP checker. The result is depicted in Figure 5.4, where `@` is Isabelle's append operator for lists.

Note that the soundness proof remains almost unchanged w.r.t. Figure 5.3. We only change the assumption `check_solvable pcp solution` into `isOK (check_solvable pcp solution)`. This works since after our setup, Isabelle's simplifier immediately translates the new assumption into

$$\begin{aligned}
 &(\forall x \in \text{set } \text{solution}. x < \text{length } \text{pcp}) \wedge \\
 &\text{solution} \neq [] \wedge \\
 &\text{concat } (\text{map } \text{fst } (\text{map } (\text{pair_of_index } \text{pcp}) \text{solution})) = \\
 &\quad \text{concat } (\text{map } \text{snd } (\text{map } (\text{pair_of_index } \text{pcp}) \text{solution}))
 \end{aligned}$$

5. A Framework for Developing Stand-Alone Certifiers

```

type_synonym "shows" = "string  $\Rightarrow$  string"

class "show" =
  fixes shows_prec :: "nat  $\Rightarrow$  'a  $\Rightarrow$  shows"
  and shows_list :: "'a list  $\Rightarrow$  shows"
  assumes "shows_prec p x (y @ z) = shows_prec p x y @ z"
  and "shows_list xs (y @ z) = shows_list xs y @ z"
begin
abbreviation "shows  $\equiv$  shows_prec 0"
abbreviation "show x  $\equiv$  shows x []"
end

```

Figure 5.5.: A *show*-class in Isabelle/HOL

which speaks again about Boolean connectives and does not contain any monadic values or error messages at all.

5.4. Readable Error Messages

In the previous section we made use of some rudimentary error messages. However, these were just static strings. For example, invoking *check-solvable* on p' with certificate $[1, 3, 2, 3, 4, 1, 2]$ yields the output.

```
In1 ''problem in ensuring satisfiability of PCP: index i invalid''
```

The “i” in *index i invalid* is just an uninformative character and does not reflect the more informative number i that would be available inside *check-all* via the binding λi . Similarly, the resulting words are not displayed if they do not match, and it is also not shown which PCP instance instance is actually analyzed.

However, to generate all these error messages, we need some functionality to display arbitrary values. To this end, we introduced a type class *show* similar to Haskell’s *Show* class [60]. The class interface is shown in Figure 5.5.

Here, *shows* is the type of functions from strings to strings, which allows for constant time concatenation. For each instance $'a$ of the *show*-class, there is a function *shows-prec* that takes a precedence (which may influence parenthesization) and a value of type $'a$. The given value is turned into a string, wrapped inside the *shows* type. To display lists in a special form, *shows-list* can be used, e.g., to allow special treatment of strings, which in Haskell and Isabelle are just lists of characters. The show-law which should be satisfied according to the Haskell documentation (and more or less states that a show-function is not allowed to modify an incoming string) is enforced in the Isabelle class definition.

In addition to *shows-prec* and *shows-list* which have to be defined for each instance, there are the functions *shows* and *show* which do not require any precedence and deliver a string, potentially wrapped into the type *shows*.

Note that in comparison to Haskell where it suffices to define *shows-prec* during instantiation (in which case *shows-list* gets a default implementation), in Isabelle’s

```

instantiation unit :: "show"
begin

definition "shows_prec p (x::unit) = shows_string '()''"
definition "shows_list (xss::unit list) = showsp_list shows_prec 0 xss"

instance
  by (standard) (auto simp: shows_prec_unit_def shows_list_unit_def
show_law_simps)

end

```

Figure 5.6.: Instantiating the *show*-class for the *unit*-type

type-class system, there is no direct possibility to define default implementations.

To this end, we designed a dedicated command **standard-shows-list** which automatically generates a definition for *shows-list*, based on *shows-prec*, and also proves the show-law for *shows-list*, using the one for *shows-prec*.

For example, the instantiation for the *unit* type is provided in Figure 5.6.

In a similar way, we defined show functions for lists, \mathbb{N} , \mathbb{Z} , \mathbb{Q} , and products. Only for characters, we defined a dedicated *shows-list* function.

For some other standard types of Isabelle, namely *bool*, *sum*, and *option*, we have used a more automatic method, similar to Haskell’s **deriving Show**. To be more precise, we have written a tactic that automatically defines show functions for datatypes—printing the constructors of the datatypes with added parentheses—and proving the required show-law. It is then possible to instantiate the *show*-class with the simple command: **derive show datatype**.

Although we could have used this facility to define the instances for \mathbb{N} and products, we did not choose this solution in order to get a nicer presentation. Currently, *show* (3, *True*) results in the string (3, *True*), whereas if we would have used **derive**, the result would have been *Pair* (*Suc* (*Suc* (*Suc* (*zero*)))) (*True*).

Using *show* it is now possible to add proper error messages into the PCP checker, cf. Figure 5.7. Here, *++* and *+@* are constant time concatenation operators of type *string* \Rightarrow *shows* \Rightarrow *shows* and *shows* \Rightarrow *shows* \Rightarrow *shows*, respectively.

When comparing the new definition with the previous one in Figure 5.4, one first notices a difference in the type of *check-solvable*: the type of letters *'a* now is equipped with the type class constraint *show*. Moreover, the resulting error message is of type *shows* instead of *string*.

Within the definition, clearly the error messages changed from static to dynamic ones, e.g., the index *i* is printed, the resulting words are displayed, and even the whole PCP instance is returned in the error message.

Note that the performed modifications (w.r.t. Figure 5.4) did not require a single change in the soundness proof.

5. A Framework for Developing Stand-Alone Certifiers

```
fun check_solvable :: "('a :: show) pcp_problemI  $\Rightarrow$  pcp_certificate  $\Rightarrow$  shows
check"
where
  "check_solvable pcp solution = do {
    check_all ( $\lambda$  i. i < length pcp) solution
    <+? ( $\lambda$  i. ''index '' +#+ shows i +@+ shows '' invalid'');
    let pair_list = map (pair_of_index pcp) solution;
    check (solution  $\neq$  []) (shows ''solution must not be empty'');
    let left = concat (map fst pair_list);
    let right = concat (map snd pair_list);
    check (left = right)
    (''resulting words are not equal: '' +#+ shows left +@+ '' != '' +#+
shows right)
  } <+? ( $\lambda$  s. ''problem in ensuring satisfiability of PCP '' +#+ shows pcp +@+
shows_nl +@+ s)"
```

Figure 5.7.: A Certified Checker with Proper Error Messages

5.5. Parsing

Let us shortly recapitulate what we have achieved so far: we can conveniently define *check-prop* programs of type *input* \Rightarrow *certificate* \Rightarrow *shows check*, which guarantee semantic properties, deliver readable error messages in case of rejection, and can be exported into various target languages via code generation.

Hence, for validating some concrete input and certificate, one just needs to transform the input and certificate provided by the automated prover into the types *input* and *certificate* that are expected by *check-prop*. However, these transformations usually depend on the code generator and the target language: how are the Isabelle types *input* and *certificate* reflected in the generated code, e.g., what are the exact names of the constructors, etc. Therefore, instead of having to build several parsers—one for each target language—and also maintain them by reflecting for example changes in the naming scheme of the code generator, we propose to build only one parser which does not need any maintenance.

The idea is to define the parser directly within the proof assistant. Then this parser can also be exported to all target languages, and the only interface to the target language that must be maintained are strings.

Since we are not aware of any automatic parser generators for proof assistants, i.e., generators which automatically produce parsers within the logic of the proof assistant, we developed some machinery to ease the manual definition of parsers.

Here, we restrict to inputs and certificates in the structured XML format. Our support is divided into two steps: we provide functionality to parse strings into XML-documents (with an accompanying Isabelle datatype to represent XML-documents), and a set of combinators to ease parsing XML-documents.

```

parse_nodes ts =
(if ts = [] ∨ take 2 ts = '</' then return [] ts
 else if hd ts ≠ CHR '<'
   then (do {
     t ← parse_text;
     ns ← parse_nodes;
     return (XML_text (the t) # ns)
   })
   ts
 else (do {
   exactly '<';
   n ← parse_name;
   atts ← parse_attributes;
   e ← oneof ['>', '>'];
   λts'. if e = '>'
     then (do {
       cs ← parse_nodes;
       return (XML n atts [] # cs)
     })
     ts'
   else (do {
     cs ← parse_nodes;
     exactly '</' ;
     exactly n;
     exactly '>';
     ns ← parse_nodes;
     return (XML n atts cs # ns)
   })
   ts'
 })
 ts)

```

Figure 5.8.: A Parser for Lists of XML-Nodes.

5.5.1. A Parser from Strings to XML

For the first phase, where strings should be converted into XML, we specified a handwritten parser as a monadic function where the monad is a state-monad with error, i.e., it captures a state (the remaining list of characters) and either returns a normal result or ends with an error message. Using the `do`-notation for monads, this parser was quite easy to define in a readable way. For example, the most complicated parser is the one for lists of XML-nodes which is depicted in Figure 5.8, where the current state is mostly hidden within the monad and where `xml list parser` is just an abbreviation for `string ⇒ string + (xml list × string)`.

However, since for the definition we used Isabelle’s function package [74], we needed to prove termination of the parser. This required tedious reasoning about the internal state of the state monad, where we had to prove that some of the auxiliary parsers actually

5. A Framework for Developing Stand-Alone Certifiers

consume tokens before each recursive invocation of *parse-nodes*, and that none of the parsers which are invoked before a recursive call, increases the length of the token list, which includes *parse-nodes* itself. Therefore, a simple structural termination argument is not applicable, and instead we wrote a proof of 160 lines that simultaneously shows termination and a decrease of the length of the resulting token list.

As the final result of the first phase, we provide a function *doc-of-string* of type $\text{string} \Rightarrow \text{string} + \text{xmldoc}$ which takes a string and either returns an error message or an XML-document.

5.5.2. A Library for Parsing XML

In the second phase, where XML-parsers for input and certificates have to be defined, we support the developer of the certifier by a collection of combinators which can be used to easily define parsers. In contrast to Section 5.5.1, here we do not use the function package, but use Isabelle’s **partial-function** command [75]. The advantage is that this command allows us to define functions without any termination proof. And as indicated in the previous paragraph, these termination proofs can become quite tedious even for simple parsers; in fact, before using **partial-function** we often just postulated termination of various parsers as axioms. However, there is one prerequisite for using **partial-function**: the functions have to be monadic, and monotone w.r.t. some pointed complete partial order with a least element \perp , which is required to specify the behavior in case of nontermination.

In principle the error monad $\text{'a} + \text{'b}$ would be an appropriate return type for the XML parsers. However, this type does not satisfy the preconditions, since it does not possess a unique least element \perp , as it admits different error messages.

To this end, we defined a dedicated monadic type $\text{'a} +_{\perp} \text{'b}$ with constructors *Left* 'a (for errors), *Right* 'b (for results), and \perp (for nontermination). Moreover, changing results or error messages are monotone operations on this type.

To conveniently specify monadic XML parsers on this type we provide several basic parsers (for strings, numbers, etc.) as well as combinators like *pair* or *many* which combine two parsers or lift a parser for single XML nodes to one over lists of XML nodes. Although the definitions of these combinators are straightforward, we would like to mention that setting up the combinators was not a completely trivial task: we had to configure Isabelle in a way that the required monotonicity proofs of parsers defined by the combinators are automatic.

For PCP, an XML-schema and parser is easily setup using the combinators, cf. Figures 5.9 and 5.10. The former provides the certificate for the PCP instance *p* in XML format, and the latter shows the parser as well as the function *certifier* which is the final certifier that invokes all required components.

First, the parsers for solutions and PCP instances are defined. Whereas the former, *certificate-of-xml* is a standard (non-recursive) definition, the latter *pcp-of-xml* is defined via **partial-function** and could use recursion without requiring termination; however, the format for PCP is so simple that no recursion is required.

Afterwards, *parse-input-and-certificate* combines the string-to-XML parser with


```

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="pcp.xsl"?>
<certificate>
  <pcp>
    <pair>
      <lhs><sym>A</sym></lhs>
      <rhs><sym>A</sym><sym>B</sym><sym>A</sym></rhs>
    </pair>
    <pair>
      <lhs><sym>A</sym><sym>B</sym></lhs>
      <rhs><sym>B</sym><sym>B</sym></rhs>
    </pair>
    <pair>
      <lhs><sym>B</sym><sym>A</sym><sym>A</sym></lhs>
      <rhs><sym>A</sym><sym>A</sym></rhs>
    </pair>
  </pcp>
  <solution>
    <idx>0</idx><idx>2</idx><idx>1</idx><idx>2</idx>
  </solution>
</certificate>

```

Figure 5.9.: The PCP Instance p and its Solution in XML

the XML-parsers to yield the full parser from strings to pairs of PCP instance and solution. This is also the place, where a conversion from the error monad $'a + 'b$ to its variant $'a +_{\perp} 'b$ with bottom element takes place.

Finally, the full *certifier* is defined which just parses the input string s , invokes the *check-solvable* function and converts again between the two kinds of error monads. Moreover, the error message e of type *shows* is converted into a *string*, by starting the evaluation via invocation with the empty string $''''$ as argument.

It is now quite easy to wrap the certifier function inside some glue-code in the target language in order to get a stand-alone program.

For example, Figure 5.11 shows the full Haskell program that is used as wrapper to invoke the certifier for PCP, where the certifier was exported via:

```
export_code certifier sumbot Inl Inr in Haskell module_name Certifier
```

This command exports the main *certifier* as Haskell program, in combination with the constructors *sumbot*, *Inl*, and *Inr* which are required for pattern matching the result of type $string +_{\perp} string$.

5.6. Soundness

Now that we have the fully executable *certifier*, we also want to have some soundness guarantees about it. Recall that the return type of *certifier* is $string +_{\perp} string$ with

5. A Framework for Developing Stand-Alone Certifiers

```

definition certificate_of_xml :: "xml  $\Rightarrow$  string +⊥ pcp_certificate"
where
  "certificate_of_xml = Xmlt.many ''solution'' (Xmlt.nat ''idx'') id"

partial_function (sum_bot) pcp_of_xml :: "xml  $\Rightarrow$  string +⊥ string pcp_problemI"
where
  [code]: "pcp_of_xml xml =
    Xmlt.many ''pcp''
      (Xmlt.pair ''pair''
        (Xmlt.many ''lhs'' (Xmlt.text ''sym'') id)
        (Xmlt.many ''rhs'' (Xmlt.text ''sym'') id)
        Pair) id xml"

definition
  parse_input_and_certificate :: "string  $\Rightarrow$  string +⊥ (string pcp_problemI ×
  pcp_certificate)"
where
  "parse_input_and_certificate s =
    (case Xml.doc_of_string s of
      Inl e  $\Rightarrow$  error e
    | Inr doc  $\Rightarrow$  Xmlt.pair ''certificate'' pcp_of_xml certificate_of_xml Pair
      (root_node doc))"

definition certifier :: "string  $\Rightarrow$  string +⊥ string"
where
  "certifier s = do {
    (pcp, c)  $\leftarrow$  parse_input_and_certificate s;
    (case (check_solvable pcp c) of
      Inl e  $\Rightarrow$  error (e ''')
    | Inr _  $\Rightarrow$  return ''certified that pcp is solvable'')
  }"

```

Figure 5.10.: A Parser and Certifier for Solvability of PCP

```

module Main (main) where

import Certifier -- the certifier
import System.Environment -- for getArgs
import System.IO -- for file reading
import System.Exit -- for error codes

main = do args <- getArgs
        case Prelude.length args of
          1 -> do input <- readFile (args !! 0)
                  start input
          _ -> error "usage: pcp certificate.xml"

start input =
  case certifier input of
    Sumbot (Inr message) ->
      do putStrLn "ACCEPT"
         putStrLn message
         exitSuccess
    Sumbot (Inl message) ->
      do putStrLn "REJECT"
         hPutStrLn stderr message
         exitWith (ExitFailure 1)

```

Figure 5.11.: A Haskell Wrapper to Invoke the Certifier

5. A Framework for Developing Stand-Alone Certifiers

constructors \perp , *Left*, and *Right*. For the success-case we can easily prove the following lemma inside the proof assistant (here specialized to our PCP certifier).

$$\begin{aligned} \text{certifier } s = \text{Right } m &\implies \\ \exists \text{ pcp } c. \text{ solvable } (\text{set pcp}) \wedge \text{parse_input_and_certificate } s = \text{Right } (\text{pcp}, c) & \end{aligned} \quad (5.3)$$

The problem in (5.3) is the brittle connection between the input string s and the semantic object pcp : the only connection between s and pcp is the parser. Hence, if one does not trust the parser and has nothing proven about it, then (5.3) is reduced to the following theorem.

$$\text{certifier } s = \text{Right } m \implies \exists \text{ pcp}. \text{ solvable } (\text{set pcp}) \quad (5.4)$$

This implication clearly lacks any connection between s and pcp , i.e., if the certifier accepts s , one only knows that some pcp is solvable, which is not necessarily the PCP instance that is encoded in s . And indeed, if the parser would be written in a way that it always returns the trivial PCP instance $\{(A, A)\}$ with solution $[0]$, then the certifier will never reject any proof.

Whereas a full correctness proof of the parser might be possible, there definitely is a simpler way to ensure soundness, namely via *show* functions. One can for example replace the last return-statement in Figure 5.10 by *return (show pcp)*. Then the soundness theorem is the following one

$$\text{certifier } s = \text{Right } m \implies \exists \text{ pcp}. \text{ solvable } (\text{set pcp}) \wedge m = \text{show pcp} \quad (5.5)$$

where at least the returned message m is related to the semantic object, pcp , via the *show* function *show*. Then the user of the certifier can inspect whether the string obtained from pcp corresponds to the intended input that is given in s . Clearly, here one has to trust the *show* function, but usually this is less complex than the parser and hence, also more reliable.

Instead of a human inspection we also integrated a way for an automatic comparison that the parsed input corresponds to the given input string. To this end, we make use of an XML *show* function *to-xml* which outputs the semantic object pcp as an XML-string. Then one can also easily check whether the string obtained from the parsed input is contained in the original input s , i.e., in (5.4) and (5.5) one gets the additional guarantee:

$$\exists \text{ before input after}. s = \text{before @ input @ after} \wedge \text{input} =_w \text{to-xml pcp} \quad (5.6)$$

Here, the input string s is decomposed into three parts where usually *before* is some XML preamble, *after* contains the certificate, and where $=_w$ is pure string-comparison modulo whitespace.

Of course, if one enforces such a strict comparison via strings, then the input XML string has to be normalized in some way, e.g., it must not contain comments, since the *show* function *to-xml* will not be able to invent the right comments. Moreover, there must be consensus about the input XML string and the *show* function, whether to print `<foo></foo>` or `<foo/>`, etc.

5.7. Conclusion

We presented a framework to develop stand-alone certifiers, with a simple certifier for PCP as an example. To adapt it to other certification problems, of course one has to adapt the major soundness proofs, but the method of integrating error messages, and the theories on parsing, show functions, etc. should all be easily reusable.

Based on this schema we developed **CeTA**, a certifier which supports (non)termination proofs [165], (non)confluence proofs [96], and complexity proofs [144]. Each of the soundness results is in the form of (5.4) in combination with (5.6).

We also considered safety properties like $\rightarrow^*(initial-states) \cap bad-states = \emptyset$, stating that no bad state is reachable via evaluation with \rightarrow . Here, a prototype certifier is available which accepts certificates in the form of tree automata which over-approximate the set of reachable states, cf. [36, 41].

Acknowledgments.

We thank the anonymous reviewers for their helpful comments. The authors are listed in alphabetical order regardless of individual contributions or seniority.

6. Certified Confluence of Conditional Rewriting

Publication Details

Christian Sternagel and Thomas Sternagel. Certified Confluence of Conditional Rewriting. *Journal of Automated Reasoning*, 2019
submitted

Christian Sternagel and Thomas Sternagel. Ceritfying Confluence of Almost Orthogonal CTRSs via Exact Tree Automata Completion. In *Proceedings of the 1st International Conference on Formal Structures for Computation and Deduction*, volume 52 of *Leibniz International Proceedings in Informatics*, pages 29:1–29:16, Dagstuhl, 2016
doi:10.4230/LIPIcs.FSCD.2016.29

Christian Sternagel and Thomas Sternagel. Certifying Confluence of Quasi-Decreasing Strongly Deterministic Conditional Term Rewrite Systems. In *Proceedings of the 26th International Conference on Automated Deduction*, volume 10395 of *Lecture Notes in Computer Science*, pages 413–431, Springer, 2017
doi:10.1007/978-3-319-63046-5_26

Abstract

Conditional term rewriting is a Turing-complete model of computation that allows for more direct translations between programs and rewrite systems than is typically the case for plain term rewriting without conditions. On the one hand, this alleviates program verification. On the other hand, properties of the corresponding conditional term rewrite systems, like confluence, are typically harder to establish than for plain term rewrite systems.

However, there are several automated tools that try to establish confluence of conditional term rewrite systems. In order to make the results of such tools more reliable, we formalize most of the state-of-the-art techniques they use for proving confluence of conditional term rewrite systems using the proof assistant Isabelle/HOL. Using these results, we provide a fully verified certifier that allows us to validate proofs that are generated by automated tools. Moreover, we evaluate our approach on standard benchmarks.

6.1. Introduction

Term rewriting is a well-studied and Turing-complete model of computation used in many different areas of computer science. One such area is program verification, where

6. Certified Confluence of Conditional Rewriting

```

qsort []          = []
qsort (x:xs)      = qsort us ++ [x] ++ qsort vs
  where
    (us, vs) = split x xs

    split x []          = ([], [])
    split x (y:ys) | y <= x = (y:us, vs)
                  | otherwise = (us, y:vs)
  where
    (us, vs) = split x ys

```

Listing 6.1: Haskell implementation of quicksort

we typically first translate a given computer program into a corresponding term rewrite system (TRS for short) and then apply term rewriting techniques in order to establish properties like confluence and termination. Of course this approach only allows us to transfer an inferred property from the TRS to the original program, provided the employed translation is sound with regard to the property. The more direct the translation, the easier it is to guarantee its soundness. Typically, conditional term rewrite systems (CTRSs for short) allow for more direct translations than plain TRSs.

Consider, for example, the Haskell program from Listing 6.1 that implements a version of the quicksort algorithm for lists.

It can be translated into the following CTRS, where **where**-clauses as well as pattern guards (like “| $y \leq x$ ”) of the Haskell program are directly captured by conditions:

$$\begin{aligned}
 & \text{qs}(\text{nil}) \rightarrow \text{nil} \\
 & \text{qs}(x : xs) \rightarrow \text{qs}(us) ++ (x : \text{nil}) ++ \text{qs}(vs) \\
 & \quad \Leftarrow \text{split}(x, xs) \rightarrow \langle us, vs \rangle \\
 \\
 & \text{split}(x, \text{nil}) \rightarrow \langle \text{nil}, \text{nil} \rangle \\
 & \text{split}(x, y : ys) \rightarrow \langle y : us, vs \rangle \\
 & \quad \Leftarrow \text{split}(x, ys) \rightarrow \langle us, vs \rangle, \text{leq}(y, x) \rightarrow \text{true} \\
 & \text{split}(x, y : ys) \rightarrow \langle ys, y : vs \rangle \\
 & \quad \Leftarrow \text{split}(x, ys) \rightarrow \langle us, vs \rangle, \text{leq}(y, x) \rightarrow \text{false}
 \end{aligned}$$

In recent years automatic tools that try to show various properties of (C)TRSs have been developed. Since these tools are complex pieces of software, wrong answers are not unheard of. To increase their trustworthiness a common approach is to formalize the techniques used in these tools on the computer using a proof assistant.¹ From such a

¹In the following, whenever we use the words “formalize” or “formalization,” we mean a computer aided formalization using a proof assistant.

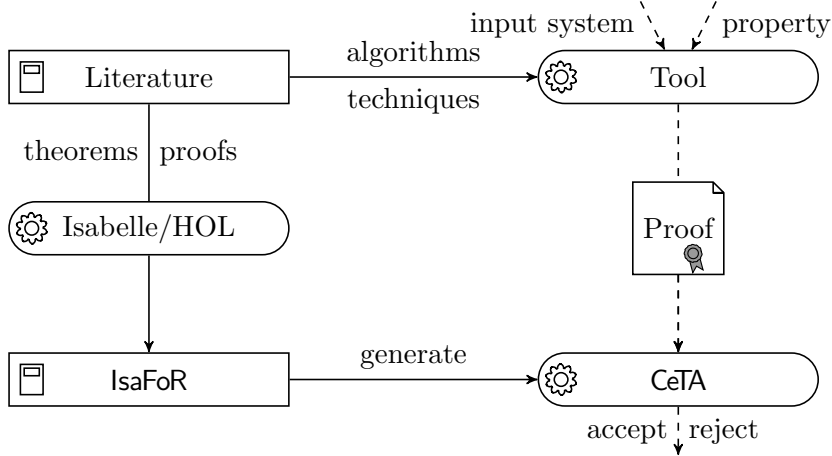


Figure 6.1.: The certification approach for automated tools.

formalization we then obtain a formally verified program that can certify the output of automated tools.

One of the largest efforts in this respect is the `IsaFoR/CeTA`² project. The *Isabelle Formalization of Rewriting* (or `IsaFoR` for short) is developed in the proof assistant Isabelle/HOL employing *Higher-Order Logic*. Before we can formally certify the output of automated tools, we have to formalize all of the underlying theory. In Isabelle/HOL it is possible to generate code [52] for a certifier from a formalization provided that check functions for all techniques and corresponding soundness lemmas have first been formalized. The certifier which is code generated from `IsaFoR` is called `CeTA` (short for *Certified Tool Assertions*) and is a fully verified, stand-alone Haskell program. Figure 6.1 gives a schematic overview of the general approach: We first search the literature for results on the properties we want to prove. Then we formalize the corresponding theorems and proofs in a proof assistant like Isabelle/HOL. Most of the time, this is not straightforward: Sometimes proofs in the literature contain real errors and we have to fix them before we can proceed. Some other time the authors built their proofs on some implicit assumptions that we have to state explicitly in order for the proof to succeed. Almost always there are gaps in the proofs, missing details or tedious technicalities that one may skip on paper and still convince an informed reader about the correctness of the proof but that are needed to convince the computer, that is, Isabelle/HOL.

Then the certifier itself is specified and proved correct inside the proof assistant to allow us to automatically code generate it from `IsaFoR`—the formal library that is the result of our formalization work.

In parallel we implement the algorithms and techniques we found in the literature in an automatic tool and adapt its output in order for the certifier to be able to parse it. A tool has to provide detailed information about every technique it employed in a proof and state this in form of a *certificate* that is readable by `CeTA`.

²<http://cl-informatik.uibk.ac.at/isafor>

6. Certified Confluence of Conditional Rewriting

Finally, we arrive at an automatic and reliable two-stage approach to check properties of the input system:

1. The tool tries to decide the desired property of a given input system automatically.
2. If a certificate is generated it is given to **CeTA** which in turn checks all the involved steps and accepts the certificate if it is correct or rejects it otherwise (giving a hopefully helpful error message).

Of course in general there are various different tools for a certain property, all using similar techniques. The main advantage of the certification approach is that instead of having to prove each of these tools correct independently, we just provide one certifier that is able to rigorously assure correctness of the tools' output with respect to a given input and we only have to prove correctness of this certifier once and for all.

For termination and confluence of plain TRSs, **CeTA** can handle more than 80% of all generated certificates in the respective tool competitions [45, 92]. However, for many applications plain TRSs are either inconvenient or not expressible enough, leading to several extensions of the base formalism. The one we are interested in here is *conditional term rewriting*. Two prominent areas where conditional rewriting is employed are the rewriting engines of modern proof assistants (like Isabelle's simplifier [102]) and functional(-logic) programming with where-clauses (like Haskell [82] and Curry [3]). An important property of conditional term rewrite systems (CTRSs) is confluence and recently interest in automatic tools to show this property has been growing (see for example the conditional category of the confluence competition [92]).

Contribution. Our goal is to get the same coverage as for termination and confluence of plain term rewriting for confluence of conditional term rewriting. So we need a formalization against which we can verify the output of the automatic confluence tools for conditional term rewriting. To this end we have extended **IsaFoR** by several results from the conditional term rewriting and tree automata literature as well as related topics. More concretely, our contributions are as follows:

- We start with a formalization of orthogonality results for CTRSs (Section 6.4).
- Then, we present the formalization of a critical pair criterion (Section 6.5).
- We also give methods for finding non-confluence witnesses (Section 6.6).
- Moreover, we discuss three methods that help us to ignore certain cases of the confluence analysis (Section 6.7) by identifying so called *infeasible* conditional critical pairs: the symbol transition graph, decomposition of reachability problems, and tree automata techniques.
- We introduce two supporting methods that allow us to ignore or simplify rules of CTRSs (Section 6.8).
- Finally (Section 6.9), we evaluate our approach with extensive experiments.

To facilitate our experiments we have implemented all the formalized methods in the automatic tool **ConCon** that is able to check confluence (and some other properties) of programs represented as CTRSs. **ConCon** is taking part in the annual confluence competition—CoCo.³ There automatic confluence tools test their mettle in several different categories. The problems that these tools try to solve come from the confluence problems database—Cops⁴ (version 1137 at the time of writing; where the version of Cops is the number of problems that are contained in that version). Beyond the results we get from this competition we provide extensive experiments, comparing the different methods of **ConCon** to each other. Certification is really a joint effort by the tool that outputs a proof and the certifier that reads it. For that reason we extended both **ConCon** and **IsaFoR** in such a way that the certifier **CeTA** is now able to check all of the above mentioned techniques and at the same time **ConCon** can provide detailed enough output for all methods it implements in a format readable by **CeTA**. Concerning check functions in our formalization it is worth mentioning that their return type is only “morally” `bool`. In order to have nice error messages we actually employ a monad. So whenever we need to handle the result of a check function as `bool` we encapsulate it in a call to `isOK` which results in `False` if there was an error and `True`, otherwise.

This work extends and unifies our previous work [135, 136, 156, 158]. Additionally, we added many examples to clarify the presented results.

In the remainder we give hyperlinks (marked by ) to an HTML rendering of our formalizations.

6.2. Preliminaries

While we try to be self-contained, some familiarity with the basics of (conditional) term rewriting [9, 111] will be helpful. We recapitulate terminology and notation that we use in the remainder in a condensed form.

An *abstract rewrite system* (ARS for short) \mathcal{A} is a carrier A together with a binary relation $\rightarrow_{\mathcal{A}}$ on A . If \mathcal{A} is clear from context we just write \rightarrow . Instead of the more common $(a, b) \in \rightarrow$ we write $a \rightarrow b$ and we call this a *rewrite step*. Given an arbitrary binary relation \rightarrow_{α} , we write $\alpha \leftarrow$, \rightarrow_{α}^+ , and \rightarrow_{α}^* for its *inverse*, its *transitive closure*, and its *reflexive transitive closure*, respectively. Moreover, given another arbitrary binary relation \rightarrow_{β} , the relations $\alpha \leftarrow \cdot \rightarrow_{\beta}^*$ and $\rightarrow_{\beta}^* \cdot \alpha \leftarrow$ are called *meetability* and *joinability*. We may abbreviate the relation $\rightarrow_{\beta}^* \cdot \alpha \leftarrow$ by \downarrow . Sometimes we call a situation $b \xrightarrow{\alpha \leftarrow} a \xrightarrow{\rightarrow_{\beta}^*} c$ a *diverging situation* or a *peak* if it consists of two single steps $b \xrightarrow{\alpha \leftarrow} a \xrightarrow{\rightarrow_{\beta}} c$. We say that \rightarrow_{α} has the *diamond property* whenever $\alpha \leftarrow \cdot \rightarrow_{\alpha} \subseteq \rightarrow_{\alpha} \cdot \alpha \leftarrow$ holds. We say that \rightarrow_{α} and \rightarrow_{β} *commute* whenever $\alpha \leftarrow \cdot \rightarrow_{\beta}^* \subseteq \rightarrow_{\beta}^* \cdot \alpha \leftarrow$ holds. The same property is called *confluence*, in case α and β coincide. If we have $b \downarrow c$ whenever $b \xrightarrow{\alpha \leftarrow} a \xrightarrow{\rightarrow_{\alpha}} c$ we call \rightarrow_{α} *locally confluent*. If \rightarrow_{α} has the diamond property and the inclusion $\rightarrow_{\beta} \subseteq \rightarrow_{\alpha} \subseteq \rightarrow_{\beta}^*$ holds then \rightarrow_{β} is confluent. An ARS \mathcal{A} is *terminating* if there is no $a \in A$ that admits an infinite rewrite sequence starting from a . The following famous result by Newman

³<http://project-coco.uibk.ac.at>

⁴<http://cops.uibk.ac.at>

6. Certified Confluence of Conditional Rewriting

[101] establishes a connection between termination and confluence: Every terminating and locally confluent ARS is confluent.

We use $\mathcal{V}(\cdot)$ to denote the set of variables occurring in a given list of syntactic objects, like terms, rules, etc. The set of terms $\mathcal{T}(\mathcal{F}, \mathcal{Var})$ over a given signature of function symbols \mathcal{F} and set of variables \mathcal{V} is defined inductively: $x \in \mathcal{T}(\mathcal{F}, \mathcal{Var})$ for all variables $x \in \mathcal{V}$, and for every n -ary function symbol $f \in \mathcal{F}$ and terms $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{Var})$ also $f(t_1, \dots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{Var})$. A term is called *linear* if it does not contain multiple occurrences of the same variable. Furthermore, we say that a term t is *ground* if $\mathcal{Var}(t) = \emptyset$. The root position of a term is denoted by ϵ . Given a term t , we write $\mathcal{Pos}(t)$ for the *set of positions* in t and $t|_p$ with $p \in \mathcal{Pos}(t)$ for the subterm of t at position p . We write $s[t]_p$ for the result of replacing $s|_p$ by t in s . Given a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{Var})$ and a fresh constant $\perp \notin \mathcal{F}$ the function *root* that returns the *root symbol* of t is defined as $\text{root}(t) = f$ if $t = f(t_1, \dots, t_n)$ and \perp otherwise. Although the use of \perp in *root* is unusual, it will be helpful in Section 6.7.2.

Given a fresh constant \square the terms in the subset of $\mathcal{T}(\mathcal{F} \cup \{\square\}, \mathcal{Var})$ where there is exactly one occurrence of \square are called *contexts*. We denote the set of all contexts by $\mathcal{C}(\mathcal{F}, \mathcal{V})$ and we also call \square the *empty context*. If C is a context, p the position of \square in the context, and t some term then $C[t]$ denotes the term $C[t]_p$. A binary relation $>$ on terms is *closed under contexts* if $C[s] > C[t]$ for all contexts C and terms s, t where $s > t$. A *rewrite rule* is a pair of terms written $\ell \rightarrow r$ where the left-hand side ℓ is not a variable and there are no variables in r that do not already occur in ℓ . A rule with variable right-hand side is called *collapsing*. We sometimes label rewrite rules like $\rho: \ell \rightarrow r$. A set of rewrite rules is called a *term rewrite system (TRS)*. A TRS is called *(left-, right-)linear* if all (left-hand, right-hand side) terms are linear. Sometimes we use *extended TRSs* where we only impose the first variable restriction. As usual \mathcal{R}^{-1} denotes the *inverse* of a TRS \mathcal{R} . Note that the inverse of a TRS could very well have variable left-hand sides. Given a TRS \mathcal{R} over signature \mathcal{F} the set $\mathcal{D} = \{\text{root}(\ell) \mid \ell \rightarrow r \in \mathcal{R}\}$ is called the *defined symbols* of \mathcal{R} . In contrast the set $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$ is called the *constructor symbols* (or just the constructors) of \mathcal{R} . A term over $\mathcal{T}(\mathcal{C}, \mathcal{V})$ is called a *constructor term*. A *substitution* is a mapping from variables to terms where only finitely many variables are not mapped to themselves. The *empty substitution* is the identity on \mathcal{V} and written ϵ . We write $t\sigma$ to denote the application of the substitution σ to the term t that is defined recursively as $t\sigma = \sigma(t)$ if t is a variable and $t\sigma = f(t_1\sigma, \dots, t_n\sigma)$ if $t = f(t_1, \dots, t_n)$. Sometimes it is useful to represent a substitution by its set of variable bindings $\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$. We call a bijective variable substitution $\pi: \mathcal{V} \rightarrow \mathcal{V}$ a *variable renaming* or *(variable) permutation*, and denote its *inverse* by π^{-1} . For two substitutions σ, τ and a set of variables V we write $\sigma = \tau[V]$ if $\sigma(x) = \tau(x)$ for all $x \in V$. We write $\sigma\tau$ for the *composition* of the two substitutions σ and τ which is defined to be $(\sigma\tau)(x) = \sigma(x)\tau$, that is, a composition lists substitutions in their order of application. Finally, a substitution σ is called a *ground substitution* if it maps variables to ground terms. A binary relation $>$ on terms is *closed under substitutions* if $s\sigma > t\sigma$ for all substitutions σ and terms s, t where $s > t$. A term s *matches* a term t if $t = s\sigma$ for some substitution σ . We say that t is an *instance* of s and conversely that s is a *generalization* of t . We say that terms s and t *unify*, written $s \sim t$, if $s\sigma = t\sigma$ for some substitution σ .

A binary relation on terms that is closed under contexts and substitutions is called a *rewrite relation*. Given a TRS \mathcal{R} we write $s \rightarrow t$ if there exists a rewrite rule $\ell \rightarrow r$ in \mathcal{R} , a substitution σ and a context C such that $s = C[\ell\sigma]$ and $t = C[r\sigma]$. We call the subterm $\ell\sigma$ the *redex*. Given a TRS \mathcal{R} a term t such that there is no step $t \rightarrow_{\mathcal{R}} s$ for any term s is called an \mathcal{R} -*normal form*. A substitution σ is \mathcal{R} -*normalized* if $\sigma(x)$ is an \mathcal{R} -normal form for all variables x . An *oriented conditional term rewrite system (CTRS)* \mathcal{R} is a set of conditional rewrite rules of the shape $\ell \rightarrow r \Leftarrow c$ where ℓ and r are terms, ℓ is not a variable, and c is a possibly empty sequence of pairs of terms (called *conditions*) $s_1 \twoheadrightarrow t_1, \dots, s_k \twoheadrightarrow t_k$. The extended TRS obtained from a CTRS \mathcal{R} by dropping the conditional parts of the rewrite rules is denoted by \mathcal{R}_u and called the *underlying TRS* of \mathcal{R} . Note that $\rightarrow_{\mathcal{R}} \subseteq \rightarrow_{\mathcal{R}_u}$. For a rule $\rho : \ell \rightarrow r \Leftarrow c$ of a CTRS \mathcal{R} the set of *extra variables* is defined as $\mathcal{EV}(\rho) = \mathcal{V}(\rho) - \mathcal{V}(\ell)$. The rewrite relation induced by a CTRS \mathcal{R} is structured into *levels*. For each level i , a TRS \mathcal{R}_i is defined recursively by

$$\begin{aligned} \mathcal{R}_0 &= \emptyset \\ \mathcal{R}_{i+1} &= \{\ell\sigma \rightarrow r\sigma \mid \ell \rightarrow r \Leftarrow c \in \mathcal{R} \text{ and } s\sigma \xrightarrow[\mathcal{R}_i]^* t\sigma \text{ for all } s \twoheadrightarrow t \in c\} \end{aligned}$$

where $\rightarrow_{\mathcal{R}_i}$ denotes the rewrite relation of the (unconditional) TRS \mathcal{R}_i . We write $s \rightarrow_{\mathcal{R},i} t$ (or $s \rightarrow_i t$ whenever \mathcal{R} is clear from the context) if we have $s \rightarrow_{\mathcal{R}_i} t$. The rewrite relation of \mathcal{R} is defined as $\rightarrow_{\mathcal{R}} = \bigcup_{i \geq 0} \rightarrow_{\mathcal{R}_i}$. Furthermore, we write $\sigma, i \vdash c$ whenever $s\sigma \xrightarrow[\mathcal{R}_i]^* t\sigma$ for all $s \twoheadrightarrow t$ in c and we say that σ *satisfies* the set of conditions c . If the level i is not important we also write $\sigma \vdash c$. When there is no substitution σ such that $\sigma \vdash c$ we say that the set of conditions c is *infeasible*. Note that a *conditional rewrite step* $s \rightarrow_{\mathcal{R}} t$ employing $\ell \rightarrow r \Leftarrow c \in \mathcal{R}$ and substitution σ is only possible if σ satisfies c . A conditional rule $\ell \rightarrow r \Leftarrow c$ is said to be of *type 3* if $\mathcal{V}(r) \subseteq \mathcal{V}(\ell, c)$ and of *type 4* otherwise. A rule of type 3 for which $\mathcal{V}(s_i) \subseteq \mathcal{V}(\ell, t_1, \dots, t_{i-1})$ holds for all $1 \leq i \leq k$ is called *deterministic*. A term t is *strongly \mathcal{R} -irreducible* if $t\sigma$ is an \mathcal{R} -normal form for all \mathcal{R} -normalized substitutions σ . A deterministic rule for which the terms t_i for all $1 \leq i \leq k$ are strongly \mathcal{R} -irreducible is called *strongly deterministic*. If a CTRS only consists of rules of type 3 it is called a *3-CTRS*, if it only consists of deterministic rules it is called a *DCTRS*, and finally if it only consists of strongly deterministic rules it is called *SDCTRS*. Two variable-disjoint variants of rules $\ell_1 \rightarrow r_1 \Leftarrow c_1$ and $\ell_2 \rightarrow r_2 \Leftarrow c_2$ in \mathcal{R} such that $\ell_1|_p$ is not a variable and $\ell_1|_p\mu = \ell_2\mu$ with most general unifier (mgu) μ , constitute a *conditional overlap*. A conditional overlap that does not result from overlapping two variants of the same rule at the root gives rise to a *conditional critical pair* (CCP) $\ell_1\mu[r_2\mu]_p \approx r_1\mu \Leftarrow c_1\mu, c_2\mu$. A (C)TRS \mathcal{R} is *orthogonal* if it is left-linear and it has no (conditional) critical pairs. The following famous result (known as the Critical Pair Lemma [61]) shows why we can concentrate on critical peaks: A TRS is locally confluent if and only if all its critical pairs are joinable. So for a terminating TRS by Newman's Lemma and the Critical Pair Lemma we immediately get the following result: A terminating TRS is confluent if and only if all its critical pairs are joinable. Note that this means that in the unconditional case confluence is decidable for terminating TRSs. A CCP $u \twoheadrightarrow v \Leftarrow c$ is said to be *infeasible* if its conditions are infeasible. Moreover, a CCP is *joinable* if $u\sigma \downarrow_{\mathcal{R}} v\sigma$ for all substitutions σ that satisfy c . The topmost part of a term that

6. Certified Confluence of Conditional Rewriting

does not change under rewriting (sometimes called its “cap”) can be approximated for example by the tcap function [43]. Informally, $\text{tcap}(x)$ for a variable x results in a fresh variable, while $\text{tcap}(t)$ for a non-variable term $t = f(t_1, \dots, t_n)$ is obtained by recursively computing $u = f(\text{tcap}(t_1), \dots, \text{tcap}(t_n))$ and then asserting $\text{tcap}(t) = u$ in case u does not unify with any left-hand side of rules in \mathcal{R} , and a fresh variable, otherwise. It is well known that $\text{tcap}(s) \not\sim t$ implies non-reachability of t from s . For two terms s and t we write $s \mapsto_{\mathcal{R}} t$ if $s = t$, $s \rightarrow_{\mathcal{R}} t$, or $s = f(s_1, \dots, s_n)$, $t = f(t_1, \dots, t_n)$, and $s_i \mapsto_{\mathcal{R}} t_i$ for all $1 \leq i \leq n$. The relation $\mapsto_{\mathcal{R}}$ is called *parallel rewriting*. The well-known Parallel Moves Lemma states that for every orthogonal TRS \mathcal{R} its parallel rewrite relation $\mapsto_{\mathcal{R}}$ has the diamond property. Together with the well-known fact, that $\rightarrow \subseteq \mapsto \subseteq \rightarrow^*$, we get that orthogonal TRSs are confluent. A CTRS \mathcal{R} over signature \mathcal{F} is called *level-commuting* if for all levels m, n and terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V}\text{ar})$ whenever $s \xrightarrow{m}^* \cdot \rightarrow_n^* t$ also $s \rightarrow_n^* \cdot \xrightarrow{m}^* t$. If $n = m$ this property is called *level-confluence* of a CTRS. Level-commutation implies level-confluence which in turn implies confluence. We denote the proper superterm relation by \triangleright and define $\succ_{\text{st}} = (\succ \cup \triangleright)^+$ for any order \succ . A DCTRS \mathcal{R} over signature \mathcal{F} is *quasi-reductive* if there is an extension \mathcal{F}' of the signature (that is, $\mathcal{F} \subseteq \mathcal{F}'$) and a well-founded partial order \succ on $\mathcal{T}(\mathcal{F}', \mathcal{V})$ that is closed under contexts such that for every substitution $\sigma: \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F}', \mathcal{V})$ and every rule $\ell \rightarrow r \leftarrow s_1 \mapsto t_1, \dots, s_k \mapsto t_k$ in \mathcal{R} we have that $s_j \sigma \succeq t_j \sigma$ for $1 \leq j < i$ implies $\ell \sigma \succ_{\text{st}} s_i \sigma$ for all $1 \leq i \leq k$, and $s_j \sigma \succeq t_j \sigma$ for $1 \leq j \leq k$ implies $\ell \sigma \succ r \sigma$. A DCTRS \mathcal{R} over signature \mathcal{F} is *quasi-decreasing* if there is a well-founded order \succ on $\mathcal{T}(\mathcal{F}, \mathcal{V}\text{ar})$ such that $\succ = \succ_{\text{st}}$, $\rightarrow_{\mathcal{R}} \subseteq \succ$, and for all rules $\ell \rightarrow r \leftarrow s_1 \mapsto t_1, \dots, s_k \mapsto t_k$ in \mathcal{R} , all substitutions $\sigma: \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{V}\text{ar})$, and all $1 \leq i \leq k$, if $s_j \sigma \rightarrow_{\mathcal{R}}^* t_j \sigma$ for all $1 \leq j < i$ then $\ell \sigma \succ s_i \sigma$. Quasi-reductivity implies quasi-decreasingness—a fact that is available in `IsaFoR`.

A bottom-up non-deterministic finite *tree automaton* (TA) $\mathcal{A} = \langle \mathcal{F}, Q, Q_f, \Delta \rangle$ consists of four parts: a signature \mathcal{F} , a set of *states* Q disjoint from \mathcal{F} , a set of *final states* $Q_f \subseteq Q$, and a set of *transitions* Δ of the shape $f(q_1, \dots, q_n) \rightarrow q$ with $f/n \in \mathcal{F}$ and $q_1, \dots, q_n, q \in Q$ or $q \rightarrow p$ with $q, p \in Q$. The *language* of a TA \mathcal{A} is given by the set $L(\mathcal{A}) = \{t \in \mathcal{T}(\mathcal{F}) \mid \text{there is a } q \in Q_f \text{ such that } t \rightarrow_{\mathcal{A}}^* q\}$. We say that a set of ground terms E is *regular* (or a *regular language*) if there is a TA \mathcal{A} such that $L(\mathcal{A}) = E$. To represent a TA we usually only need to specify the transitions and mark the final states. The signature and the set of states is clear from the transitions. A substitution from variables to states is called a *state substitution*.

6.3. Roadmap of Formalized Methods

This section presents an overview of all confluence methods for CTRSs that are available in `CeTA`. There are basically three ways to show confluence of a CTRS:

1. use a transformation to reduce the problem to the unconditional case,
2. exploit orthogonality, or
3. employ a critical pair criterion.

All of them are supported by **CeTA**.

The second and third approaches among other things also analyze the critical pairs of a CTRS. Being able to ignore certain critical pairs simplifies this analysis. Now, if the conditions of a CCP are not satisfiable then the resulting equation can never be utilized and hence is harmless for the confluence of the CTRS under consideration. CCPs with unsatisfiable conditions are called *infeasible* and we can safely ignore them for the purpose of confluence. Our certifier **CeTA** supports the following reachability analysis based methods to show infeasibility of CCPs:

1. unification,
2. symbol transition graph analysis,
3. decomposition of reachability problems, and
4. tree automata completion.

When checking for confluence we are not only interested in positive answers. If a CTRS is *not* confluent we have to find a witness to prove it. This is also supported by **CeTA**.

Sometimes the above methods for (non-)confluence are not directly applicable. **CeTA** supports two sound methods that can “simplify” a given CTRS and make it amenable for them. The first one removes rules with infeasible conditions (because they do not influence the rewrite relation anyway), while the second one “reshapes” certain conditional rewrite rules in such a way that other methods become more applicable, without changing the induced rewrite relation.

Confluence Methods. Concerning the reduction to the unconditional case the following theorem is based on work by Nishida et al. [105] and has been formalized by Winkler and Thiemann [177, Theorem 20].

Theorem 6.3.1. *For a DCTRS \mathcal{R} and a source preserving unraveling U if the TRS $U(\mathcal{R})$ is left-linear and confluent then \mathcal{R} is confluent.* \square

Since this result is not part of our formalization effort we refer the interested reader to the work of Winkler and Thiemann [177] for further details.

The first orthogonality result for 3-CTRSs is due to Suzuki et al. [162, Corollary 4.7].

Theorem 6.3.2. *Almost orthogonal, extended properly oriented, right-stable, and oriented 3-CTRSs are confluent.* \square

We slightly extend the original result and formalize it. This formalization is the topic of Section 6.4.

A critical pair criterion for SDCTRSs was published by Avenhaus and Loría-Sáenz [7, Theorem 4.1].

Theorem 6.3.3. *Let the SDCTRS \mathcal{R} be quasi-decreasing. Then \mathcal{R} is confluent iff all CCPs are joinable.* \square

Again we slightly extend this result and formalize it. A textual description of this formalization is given in Section 6.5.

6. Certified Confluence of Conditional Rewriting

Non-Confluence Methods. The following non-confluence result directly follows from the definition of confluence:

Lemma 6.3.4. *Given a CTRS \mathcal{R} if we find a diverging situation $t \xrightarrow{\mathcal{R}}^+ s \rightarrow_{\mathcal{R}}^+ u$ where t and u are not joinable then \mathcal{R} is non-confluent.* \square

Further details are presented in Section 6.6.

Infeasibility and Supporting Methods. Infeasibility of conditions can be exploited for several subtasks when trying to decide confluence of a CTRS:

1. rules with infeasible conditions can be dropped from a CTRS,
2. CCPs with infeasible conditions are trivially joinable, and finally,
3. CCPs between infeasible rules can be ignored for the purpose of orthogonality.

The details on how exactly this works in CeTA are given in Sections 6.7 and 6.8. Finally, inlining of conditions (which is also explained in detail in Section 6.8) is inspired by inlining of let-constructs and where-expressions in compilers. In practice, exhaustive application of inlining increases the applicability of other methods like infeasibility via unification and non-confluence via plain rewriting.

In the next section we start by showing how to extend the well-known orthogonality criterion from the unconditional to the conditional case.

6.4. Orthogonality

Unlike orthogonal TRSs, orthogonal CTRSs are not confluent in general, as witnessed by Cops #524 (which is similar to an example by Ida and Okui [63]):

Example 6.4.1 (Cops #524). *Consider the 3-CTRS consisting of the three rules:*

$$a \rightarrow x \Leftarrow f(x) \rightarrow k \qquad f(b) \rightarrow k \qquad f(c) \rightarrow k$$

It is orthogonal but not confluent, since we have the peak $b \leftarrow a \rightarrow c$ but the terms b and c are both normal forms and thus not joinable.

We have to impose further restrictions on the distribution of variables in the rules such that during matching the substitution for the rewrite step can be built in a deterministic way. To this end we use the following notions due to Suzuki et al. [162].

Definition 6.4.2 (Right-stability \checkmark , proper orientedness \checkmark). *A conditional rewrite rule $\ell \rightarrow r \Leftarrow c$ with k conditions $c = s_1 \rightarrow t_1, \dots, s_k \rightarrow t_k$ is called*

- *right-stable whenever we have $\text{Var}(t_i) \cap \text{Var}(\ell, c_{i-1}, s_i) = \emptyset$ and t_i is either a linear constructor term or a ground \mathcal{R}_u -normal form, for all $1 \leq i \leq k$; and*
- *properly oriented if whenever $\text{Var}(r) \not\subseteq \text{Var}(\ell)$ then $\text{Var}(s_i) \subseteq \text{Var}(\ell, t_1, \dots, t_{i-1})$ for all $1 \leq i \leq k$.*

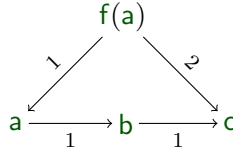


Figure 6.2.: For CTRSs parallel rewriting does not have the diamond property.


A CTRS consisting solely of right-stable rules is called *right-stable*. Likewise, a CTRS only containing properly oriented rules is called *properly oriented*.

Note that proper orientedness is really just a relaxation of determinism in that we only demand determinism if there are extra variables in right-hand sides of rules and not for all rules (see Section 6.2). Now the class of CTRSs we are targeting are orthogonal, properly oriented, right-stable, and oriented 3-CTRSs. Remember that in the unconditional case we employ the Parallel Moves Lemma to show confluence of orthogonal systems. So for a CTRS \mathcal{R} we write $s \mapsto_n t$ if t can be obtained from s by contracting a set of pairwise disjoint redexes in s using \mathcal{R}_n . Unfortunately, the Parallel Moves Lemma does not hold for our class of CTRSs as witnessed by Cops #334 [162, Example 4.4].

Example 6.4.3 (Cops #334). Consider the orthogonal, properly oriented, right-stable, and oriented 3-CTRS consisting of the three rules:

$$f(x) \rightarrow y \Leftarrow x \Rightarrow y \qquad a \rightarrow b \qquad b \rightarrow c$$

In the peak depicted in Figure 6.2 no parallel rewrite step from a to c is possible, but we still can rewrite a to the normal form c with a sequence whose level is smaller than the level of the step from $f(a)$ to c . Incorporating these findings into parallel rewriting for CTRSs we arrive at the notion of *extended parallel rewriting*:

Definition 6.4.4 (Extended parallel rewriting ). First we adopt the convention that the number of holes of a multi-hole context is denoted by the corresponding lower-case letter, for example, c for C , d for D , e for E etc. Then we say that there is an extended parallel rewrite step at level n from s to t , written $s \mapsto_n^* t$, whenever we have a multi-hole context C , and sequences of terms s_1, \dots, s_c and t_1, \dots, t_c , such that $s = C[s_1, \dots, s_c]$, $t = C[t_1, \dots, t_c]$, and for all $1 \leq i \leq c$ we have either

1. $(s_i, t_i) \in \mathcal{R}_n$ (that is, a root step at level n), or
2. $s_i \rightarrow_{n-1}^* t_i$.

It is easy to see that $\rightarrow_n \subseteq \mapsto_n^* \subseteq \rightarrow_n^*$. We are ready to state the following variation of the Parallel Moves Lemma.

Theorem 6.4.5. For orthogonal, properly oriented, right-stable, and oriented 3-CTRSs extended parallel rewriting has the commuting diamond property. \square

6. Certified Confluence of Conditional Rewriting

Because the commuting diamond property obviously implies level-commutation the above theorem yields the following confluence result.

Corollary 6.4.6. *Orthogonal, properly oriented, right-stable, and oriented 3-CTRSs are confluent.* \square

We can further improve upon this result by using a looser notion of proper orientedness.

Definition 6.4.7 (Extended proper orientedness \checkmark). *A conditional rule $\ell \rightarrow r \Leftarrow c$ with k conditions $c = s_1 \rightarrow t_1, \dots, s_k \rightarrow t_k$ is called extended properly oriented when either $\text{Var}(r) \subseteq \text{Var}(\ell)$ or there is some $0 \leq m \leq k$ such that $\text{Var}(s_i) \subseteq \text{Var}(\ell, t_1, \dots, t_{i-1})$ for all $1 \leq i \leq m$ and $\text{Var}(r) \cap \text{Var}(s_j \rightarrow t_j) \subseteq \text{Var}(\ell, t_1, \dots, t_m)$ for all $m < j \leq k$. A CTRS only containing extended properly oriented rules is called an extended properly oriented CTRS.*

There are CTRSs that are extended properly oriented but not properly oriented as witnessed by Cops #548:

Example 6.4.8 (Cops #548). *Consider the oriented 3-CTRS consisting of the single rule*

$$g(x) \rightarrow y \Leftarrow x \rightarrow y, z \rightarrow a$$

It is orthogonal, right-stable, and extended properly oriented but not properly oriented, because of the extra variable z in the second condition.

Observe the following property of a conditional rule $\ell \rightarrow r \Leftarrow c$ of type 3 with k conditions

$$\text{for some } 0 \leq m \leq k. \text{Var}(r) \subseteq \text{Var}(\ell, c_m) \cup (\text{Var}(r) \cap \text{Var}(c_{m+1,k})) \quad (\star)$$

which we will use later and which directly follows from $\text{Var}(r) \subseteq \text{Var}(\ell, c)$. Moreover, we additionally loosen the orthogonality restriction to allow overlaps that are harmless.

Definition 6.4.9 (Almost orthogonality modulo infeasibility \checkmark). *A left-linear CTRS \mathcal{R} is almost orthogonal (modulo infeasibility) if each overlap between rules $\ell_1 \rightarrow r_1 \Leftarrow c_1$ and $\ell_2 \rightarrow r_2 \Leftarrow c_2$ with mgu μ at position p either*

1. *results from overlapping two variants of the same rule at the root, or*
2. *is trivial (that is, $p = \epsilon$ and $r_1\mu = r_2\mu$), or*
3. *is infeasible in the following sense: for arbitrary m and n , whenever levels m and n commute, then it is impossible to satisfy the conditions stemming from the first rule on level m and at the same time the conditions stemming from the second rule on level n . More formally: $\forall m n. (\overset{*}{\leftarrow}_m \cdot \rightarrow_n^* \subseteq \rightarrow_n^* \cdot \overset{*}{\leftarrow}_m \implies \nexists \sigma. \sigma, m \vdash c_1\mu \wedge \sigma, n \vdash c_2\mu)$.*

Note that without 2 and 3, Definition 6.4.9 corresponds to plain orthogonality. Also note that by dropping 3, Definition 6.4.9 reduces to the definition of *almost orthogonality* given by Hanus [55]. In the following, whenever we talk about *almost orthogonality* we mean Definition 6.4.9. Observe that the level-commutation assumption of the third alternative in Definition 6.4.9 allows us to reduce non-meetability to non-joinability. That this is useful in practice is shown by the following example featuring Cops #547 [135, Example 3].

Example 6.4.10 (Non-meetability via tcap , Cops #547). *Consider the CTRS consisting of the two rules:*

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

It has the critical pair

$$\mathsf{a} \approx \mathsf{b} \Leftarrow x \rightarrow \mathsf{a}, x \rightarrow \mathsf{b}$$

Since $\mathsf{tcap}(\mathsf{cs}(x, x)) = \mathsf{cs}(y, z) \sim \mathsf{cs}(\mathsf{a}, \mathsf{b})$, where cs is a fresh auxiliary function symbol, we cannot conclude infeasibility via non-reachability analysis using tcap . However, $\mathsf{tcap}(\mathsf{a}) = \mathsf{a} \not\sim \mathsf{b} = \mathsf{tcap}(\mathsf{b})$ shows non-joinability of a and b . By Definition 6.4.9.3 this shows non-meetability of a and b and thereby infeasibility of the critical pair.

In general it is beneficial to test for non-meetability via non-joinability of conditions with identical left-hand sides (see also Lemma 6.7.41). Finally, we are ready to state this new variation of the Parallel Moves Lemma for extended parallel rewriting.

Theorem 6.4.11. *For almost orthogonal, extended properly oriented, right-stable, and oriented 3-CTRSs extended parallel rewriting has the commuting diamond property. \checkmark*

Proof. Let \mathcal{R} be a CTRS satisfying all required properties. The commuting diamond property for parallel rewriting states that $m \Leftarrow \cdot \Leftarrow_n \subseteq \Leftarrow_n \cdot m \Leftarrow$ for all m and n . We proceed by complete induction on $m + n$. By induction hypothesis (IH) we may assume the result for all $m' + n' < m + n$. Now consider the peak $t \Leftarrow_m s \Leftarrow_n u$. If any of m and n equals 0, we are done (since \Leftarrow_0 is the identity relation). Thus we may assume $m = m' + 1$ and $n = n' + 1$ for some m' and n' . By the definition of extended parallel rewriting, we obtain multihole contexts C and D , and sequences of terms s_1, \dots, s_c , t_1, \dots, t_c , u_1, \dots, u_d , v_1, \dots, v_d , such that $s = C[s_1, \dots, s_c]$ and $t = C[t_1, \dots, t_c]$, as well as $s = D[u_1, \dots, u_d]$ and $u = D[v_1, \dots, v_d]$; and $(s_i, t_i) \in \mathcal{R}_m$ or $s_i \rightarrow_{m'}^* t_i$ for all $1 \leq i \leq c$, as well as $(u_i, v_i) \in \mathcal{R}_n$ or $u_i \rightarrow_{n'}^* v_i$ for all $1 \leq i \leq d$.

It is relatively easy to define the greatest lower bound $C \sqcap D$ of two contexts C and D by a recursive function (that simultaneously traverses the two contexts in a top-down manner and replaces subcontexts that differ by a hole) and prove that we obtain a lower semilattice. Now we identify the common part E of C and D , employing the semilattice properties of multihole contexts, that is, $E = C \sqcap D$. Then $C = E[C_1, \dots, C_e]$ and $D = E[D_1, \dots, D_e]$ for some multihole contexts C_1, \dots, C_e and D_1, \dots, D_e such that for each $1 \leq i \leq e$ we have $C_i = \square$ or $D_i = \square$. This also means that there is a

6. Certified Confluence of Conditional Rewriting

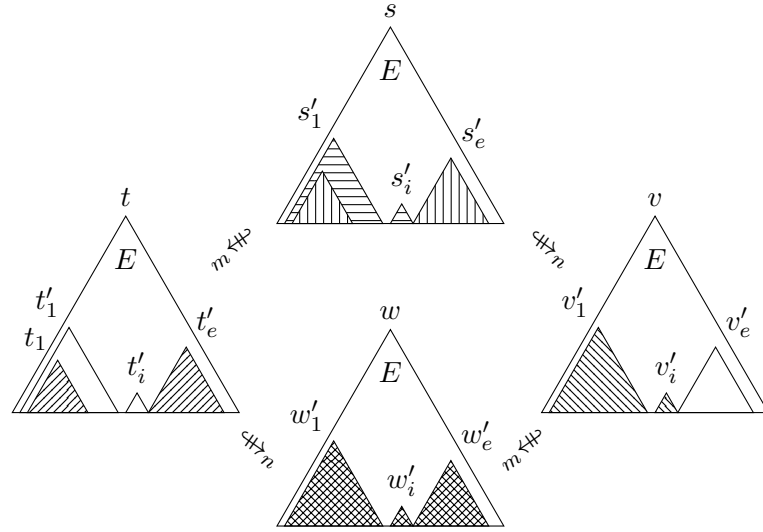


Figure 6.3.: Commuting diamond property of extended parallel rewriting.

sequence of terms s'_1, \dots, s'_e such that $s = E[s'_1, \dots, s'_e]$ and for all $1 \leq i \leq e$, we have $s'_i = C_i[s_{k_i}, \dots, s_{k_i+c_i-1}]$ for some subsequence $s_{k_i}, \dots, s_{k_i+c_i-1}$ of s_1, \dots, s_c (we denote similar terms for t , u , and v by t'_i , u'_i , and v'_i , respectively). Moreover, note that by construction $s'_i = u'_i$ for all $1 \leq i \leq e$. Since extended parallel rewriting is closed under multihole contexts, it suffices to show that for each $1 \leq i \leq e$ there is a term v such that $t'_i \leftarrow_n v \xleftarrow{m} v'_i$, in order to conclude the proof. This is depicted in Figure 6.3, where w'_i denotes the respective v 's. We concentrate on the case $C_i = \square$ (the case $D_i = \square$ is completely symmetric). Moreover, note that when we have $s'_i \rightarrow_{m'}^* t'_i$, the proof concludes by IH (together with some basic properties of the involved relations), and thus we remain with $(s'_i, t'_i) \in \mathcal{R}_m$. At this point we distinguish the following cases:

1. ($D_i = \square$). Also here, the non-root case $u'_i \rightarrow_{n'}^* v'_i$ is covered by the IH. Thus, we may restrict to $(u'_i, v'_i) \in \mathcal{R}_n$, giving rise to a root overlap. Since \mathcal{R} is almost orthogonal, this means that either the resulting conditions are not satisfiable or the resulting terms are the same (in both of these cases we are done), or two variable disjoint variants of the same rule $\ell \rightarrow r \Leftarrow c$ with conditions $c = s_1 \twoheadrightarrow t_1, \dots, s_j \twoheadrightarrow t_j$ were involved, that is, $u'_i = \ell\sigma_1 = \ell\sigma_2$ for some substitutions σ_1 and σ_2 that both satisfy all conditions in c . Without extra variables in r , this is the end of the story (since then $r\sigma_1 = r\sigma_2$); but we also want to cover the case where $\text{Var}(r) \not\subseteq \text{Var}(\ell)$, and thus have to reason why this does not cause any trouble. Together with the fact that $\ell \rightarrow r \Leftarrow c$ is extended properly oriented we obtain a $0 \leq k \leq j$ such that

- a) $\text{Var}(s_i) \subseteq \text{Var}(\ell, t_1, \dots, t_{i-1})$ for all $1 \leq i \leq k$ and
- b) $\text{Var}(r) \cap \text{Var}(s_i \twoheadrightarrow t_i) \subseteq \text{Var}(\ell, t_1, \dots, t_k)$ for all $k < i \leq j$

by Definition 6.4.7. Then we prove by an inner induction on $i \leq j$ that there is a substitution σ such that

- c) $\sigma(x) = \sigma_1(x) = \sigma_2(x)$ for all x in $\text{Var}(\ell)$, and
- d) $\sigma_1(x) \rightsquigarrow_{n'}^* \sigma(x)$ and $\sigma_2(x) \rightsquigarrow_{m'}^* \sigma(x)$ for all x in $\text{Var}(\ell, c_{\min\{i,k\}}) \cup (\text{Var}(r) \cap \text{Var}(c_{k+1,i}))$.

In the base case σ_1 satisfies the requirements. So suppose $i > 0$ and assume by IH that both properties hold for $i - 1$ and some substitution σ . If $i > k$ we are done by 1b. Otherwise $i \leq k$. Now consider the condition $s_i \rightarrow t_i$. By 1a we have $\text{Var}(s_i) \subseteq \text{Var}(\ell, c_{i-1})$. Using the IH for 1d we obtain $s_i\sigma_1 \rightsquigarrow_{n'}^* s_i\sigma$ and $s_i\sigma_2 \rightsquigarrow_{m'}^* s_i\sigma$. Moreover $s_i\sigma_1 \rightsquigarrow_{m'}^* t_i\sigma_1$ and $s_i\sigma_2 \rightsquigarrow_{n'}^* t_i\sigma_2$ since σ_1 and σ_2 satisfy c , and thus by the outer IH we obtain s' such that $t_i\sigma_1 \rightsquigarrow_{n'}^* s'$ and $t_i\sigma_2 \rightsquigarrow_{m'}^* s'$. Recall that by right-stability t_i is either a ground \mathcal{R}_u -normal form or a linear constructor term. In the former case $t_i\sigma_1 = t_i\sigma_2 = s'$ and hence σ satisfies 1c and 1d. In the latter case right-stability allows us to combine the restriction of σ_1 to $\text{Var}(t_i)$ and the restriction of σ to $\text{Var}(\ell, c_{i-1})$ into a substitution satisfying 1c and 1d. This concludes the inner induction. Since \mathcal{R} is an extended properly oriented 3-CTRS, using (\star) together with 1d shows $r\sigma_1 \rightsquigarrow_{n'}^* r\sigma$ and $r\sigma_2 \rightsquigarrow_{m'}^* r\sigma$. Since $\rightsquigarrow_{n'}^* \subseteq \rightsquigarrow_n$ and $\rightsquigarrow_{m'}^* \subseteq \rightsquigarrow_m$ we can take $v = r\sigma$ to conclude this case.

2. ($D_i \neq \square$). Then for some $1 \leq k \leq d$, we have $(u_j, v_j) \in \mathcal{R}_n$ or $u_j \rightarrow_{n'}^* v_j$ for all $k \leq j \leq k + d_i - 1$, that is, an extended parallel rewrite step of level n from $s'_i = u'_i = D_i[u_{k_i}, \dots, u_{k_i+d_i-1}]$ to $D_i[v_{k_i}, \dots, v_{k_i+d_i-1}] = v'_i$. Since \mathcal{R} is almost orthogonal and, by $D_i \neq \square$, root overlaps are excluded, the constituent parts of the extended parallel step from s'_i to v'_i take place exclusively inside the substitution of the root step to the left (which is somewhat obvious but surprisingly hard to formalize, even more so when having to deal with infeasibility). We again close this case by induction on the number of conditions making use of right-stability of \mathcal{R} . \square

The same reasoning as before immediately yields the main result of this section—Theorem 6.3.2. Clearly, applicability of Theorem 6.3.2 relies on having powerful techniques for proving infeasibility at our disposal. Those are the topic of Section 6.7.

6.4.1. Certification

Since all the properties of our target CTRSs are syntactical it is straightforward to implement the corresponding check functions. If there are no critical pairs the certificate only contains one element that states that the *almost orthogonal* criterion was used. The syntactic properties are checked by **CeTA**. More involved proofs contain subproofs of infeasibility for all the CCPs (see Section 6.7).

The formalization of the methods described in this section can be found in the following **IsaFoR** theory files:

thys/Conditional_Rewriting/ Conditional_Rewriting.thy Conditional_Critical_Pairs.thy	thys/Conditional_Rewriting/ Level_Confluence.thy Level_Confluence_Impl.thy
--	--

6. Certified Confluence of Conditional Rewriting

The next section explores a method to show confluence of quasi-decreasing CTRSs provided all of their conditional critical pairs are joinable.

6.5. A Critical Pair Criterion

In the previous section we have seen how to adapt the result that orthogonal TRSs are confluent to the conditional case. Here we do the same for terminating and locally confluent TRSs. Remember that for terminating TRSs confluence is decidable. We just have to check if all CPs are joinable. This well-known result of unconditional term rewriting directly follows from Newman's Lemma and the Critical Pair Lemma (see Section 6.2). Unfortunately, the same result does not hold in the conditional case. To begin with, the Critical Pair Lemma does not hold for CTRSs as witnessed by the following example featuring Cops #269, the oriented version of a join system due to Bergstra and Klop [17, Example 3.6].

Example 6.5.1 (Cops #269). *The CTRS \mathcal{R} consisting of the two rules*

$$c(x) \rightarrow e \Leftarrow x \rightarrow c(x) \qquad b \rightarrow c(b)$$

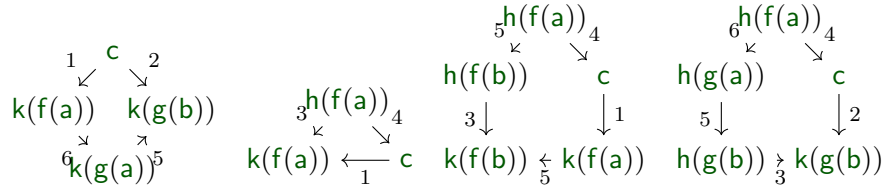
has no CCPs at all but is not (locally) confluent due to the peak $c(e) \leftarrow c(c(b)) \rightarrow e$, where both steps employ the first rule—the one to the left at position 1 and the one to the right at the root position—and the conditions are satisfied by the second rule. Since $c(e)$ and e are two different normal forms we have a non-joinable peak and hence a non-confluent system.

The above system is not terminating. Maybe terminating CTRSs where all CCPs are joinable are confluent? The next example featuring Cops #553, the oriented version of a join system due to Dershowitz et al. [33, Example B], shows that this is not the case.

Example 6.5.2 (Cops #553). *Consider the terminating CTRS \mathcal{R} :*

$$\begin{array}{llll} c \rightarrow k(f(a)) & (1) & h(x) \rightarrow k(x) & (3) \quad a \rightarrow b & (5) \\ c \rightarrow k(g(b)) & (2) & h(f(a)) \rightarrow c & (4) & f(x) \rightarrow g(x) \Leftarrow h(f(x)) \rightarrow k(g(b)) & (6) \end{array}$$

There are four CCPs (modulo symmetry) that are all joinable as shown in the diagrams below (where the number of the rule used in a step is attached to the corresponding arrow).



We still have the diverging situation $f(b) \leftarrow f(a) \rightarrow g(a) \rightarrow g(b)$ where $f(b)$ and $g(b)$ are two different normal forms. So \mathcal{R} is not confluent.

Clearly termination is not enough. Thus, we employ the stronger notion of quasi-decreasingness which in addition to termination also ensures that the rewrite relation is effectively computable and that there are no infinite computations in the conditions. Still, this is not yet sufficient. Because of the possibility of extra variables in conditional rules there are problems for confluence that can arise from overlaps of a rule with itself at the root position and even from variable-overlaps; two cases that are not dangerous at all when considering unconditional term rewriting. This will become clearer after looking at the following two examples featuring Cops #262 and #263 due to Avenhaus and Loría-Sáenz [7, Examples 4.1a,b].

Example 6.5.3 (Cops #262). Consider the quasi-decreasing CTRS \mathcal{R} consisting of the following three rules (where we write $s + t$ instead of $\text{plus}(s, t)$)

$$0 + y \rightarrow y \quad s(x) + y \rightarrow x + s(y) \quad f(x, y) \rightarrow z \Leftarrow x + y \rightarrow z + z[']$$

Now look at the (improper) overlap of the last rule with itself. This yields the (improper) CCP $z \approx w \Leftarrow x + y \rightarrow z + v, x + y \rightarrow w + u$. Remember that to show joinability of a CCP we have to check whether it is joinable for all satisfying substitutions. Let us see if we can find a satisfying substitution which makes the CCP non-joinable. Now, if we apply the substitution σ mapping x, z , and u to $s(0)$ and all other variables to 0 to the CCP, the result is

$$s(0) \approx 0 \Leftarrow s(0) + 0 \rightarrow s(0) + 0, s(0) + 0 \rightarrow 0 + s(0)$$

The first condition is trivially satisfied. To satisfy the second one we use the second rule of \mathcal{R} . Clearly σ satisfies the CCP. But $s(0)$ and 0 are two different normal forms and so the CCP is not joinable. Which in turn means that \mathcal{R} is not confluent.


Example 6.5.4 (Cops #263). The quasi-decreasing CTRS \mathcal{R} consists of four rules

$$a \rightarrow c \quad g(a) \rightarrow h(b) \quad h(b) \rightarrow g(c) \quad f(x) \rightarrow z \Leftarrow g(x) \rightarrow h(z)$$

From the variable-overlap between the first and the last rule we get the (improper) CCP $f(c) \approx z \Leftarrow g(a) \rightarrow h(z)$. With $\sigma(z) = b$ we have found a satisfying substitution that makes the left- and right-hand sides non-joinable.

To counter these problems we have to restrict the placement of extra variables in the conditions severely. To this end we focus our attention on *strongly deterministic* CTRSs in this section.

The main result of Avenhaus and Loría-Sáenz [7, Theorem 4.1] is the basis for Theorem 6.3.3.

Theorem 6.5.5. *If all CCPs of a quasi-decreasing SDCTRS are joinable, then it is confluent.* 

Proof. That all CCPs of a CTRS \mathcal{R} (no need for strong determinism or quasi-decreasingness) are joinable if \mathcal{R} is confluent is straightforward. Thus, we concentrate on the other

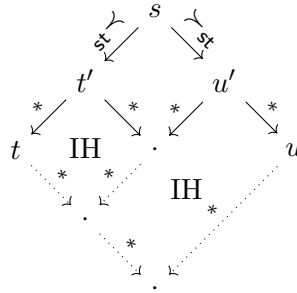
6. Certified Confluence of Conditional Rewriting

direction: Assume that all critical pairs are joinable. We consider an arbitrary diverging situation $t \xrightarrow{*}_{\mathcal{R}} s \xrightarrow{*}_{\mathcal{R}} u$ and prove $t \downarrow_{\mathcal{R}} u$ by well-founded induction with respect to \succ_{st} . Here \succ is the order from the definition of quasi-decreasingness.

By the induction hypothesis (IH) we have that for all terms t_0, t_1, t_2 such that $s \succ_{\text{st}} t_0$ and $t_1 \xrightarrow{*}_{\mathcal{R}} t_0 \xrightarrow{*}_{\mathcal{R}} t_2$ there exists a join $t_1 \rightarrow_{\mathcal{R}}^* \cdot \xrightarrow{*}_{\mathcal{R}} t_2$.

If $s = t$ or $s = u$ then t and u are trivially joinable and we are done. So we may assume that the diverging situation contains at least one step in each direction: $t \xrightarrow{*}_{\mathcal{R}} t' \xrightarrow{*}_{\mathcal{R}} s \xrightarrow{*}_{\mathcal{R}} u' \xrightarrow{*}_{\mathcal{R}} u$.

Let us show that $t' \downarrow_{\mathcal{R}} u'$ holds. Then $t \downarrow_{\mathcal{R}} u$ follows by two applications of the induction hypothesis, as shown in the following diagram:



Assume that $s = C[\ell_1\sigma_1]_p \rightarrow_{\mathcal{R}} C[r_1\sigma_1]_p = t'$ and $s = D[\ell_2\sigma_2]_q \rightarrow_{\mathcal{R}} D[r_2\sigma_2]_q = u'$ for rules $\rho_1 : \ell_1 \rightarrow r_1 \Leftarrow c_1$ and $\rho_2 : \ell_2 \rightarrow r_2 \Leftarrow c_2$ in \mathcal{R} , contexts C and D , positions p and q , and substitutions σ_1 and σ_2 such that $u\sigma_1 \rightarrow_{\mathcal{R}}^* v\sigma_1$ for all $u \rightarrow v \in c_1$ and $u\sigma_2 \rightarrow_{\mathcal{R}}^* v\sigma_2$ for all $u \rightarrow v \in c_2$. There are three possibilities: either the positions are parallel ($p \parallel q$), or p is above q ($p \leq q$), or q is above p ($q \leq p$). In the first case $t' \downarrow_{\mathcal{R}} u'$ holds because the two redexes do not interfere. The other two cases are symmetric and we only consider $p \leq q$ here. If $s \triangleright s|_p = \ell_1\sigma_1$ then $s \succ_{\text{st}} \ell_1\sigma_1$ (by definition of \succ_{st}) and there exists a position r such that $q = pr$ and so we have the diverging situation $r_1\sigma_1 \xrightarrow{*}_{\mathcal{R}} \ell_1\sigma_1 \xrightarrow{*}_{\mathcal{R}} \ell_1\sigma_1[r_2\sigma_2]_r$ which is joinable by the induction hypothesis. But then the diverging situation $t' = s[r_1\sigma_1]_p \xrightarrow{*}_{\mathcal{R}} s[\ell_1\sigma_1]_p \xrightarrow{*}_{\mathcal{R}} s[\ell_1\sigma_1[r_2\sigma_2]_r]_q = u'$ is also joinable (by closure under contexts) and we are done. So we may assume that $p = \epsilon$ and thus $s = \ell_1\sigma_1$. Now, either q is a function position in ℓ_1 or there exists a variable position q' in ℓ_1 such that $q' \leq q$. In the first case we either have

1. a conditional critical pair which is joinable by assumption or we have
2. a root-overlap of variants of the same rule. Unlike in the unconditional case this could lead to non-joinability of the ensuing critical pair because of the extra variables in the right-hand sides of conditional rules. We have $\rho_1\pi = \rho_2$ for some permutation π . Moreover, $s = \ell_1\sigma_1 = \ell_2\sigma_2$ and we have

$$\pi^- \sigma_1 = \sigma_2 [\mathcal{V}(\ell_2)] \quad (6.1)$$

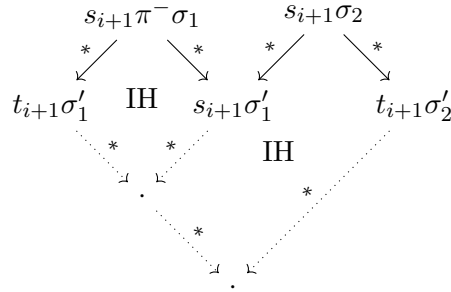
We prove $x\pi^- \sigma_1 \downarrow_{\mathcal{R}} x\sigma_2$ for all x in $\mathcal{V}(\rho_2)$. Since $t' = r_1\sigma_1 = r_2\pi^- \sigma_1$ and $u' = r_2\sigma_2$ this shows $t' \downarrow_{\mathcal{R}} u'$. Because \mathcal{R} is terminating (by quasi-decreasingness) we may

define two normalized substitutions σ'_i such that

$$x\pi^-\sigma_1 \xrightarrow[\mathcal{R}]{} x\sigma'_1 \text{ and } x\sigma_2 \xrightarrow[\mathcal{R}]{} x\sigma'_2 \text{ for all variables } x. \quad (6.2)$$

We prove $x\sigma'_1 = x\sigma'_2$ for $x \in \mathcal{EV}(\rho_2)$ by an inner induction on the length k of the conditions $c_2 = s_1 \twoheadrightarrow t_1, \dots, s_k \twoheadrightarrow t_k$. If ρ_2 has no conditions this holds vacuously because there are no extra variables. In the step case the inner induction hypothesis is that $x\sigma'_1 = x\sigma'_2$ for $x \in \mathcal{V}(s_1, t_1, \dots, s_i, t_i) - \mathcal{V}(\ell_2)$ and we have to show that $x\sigma'_1 = x\sigma'_2$ for $x \in \mathcal{V}(s_1, t_1, \dots, s_{i+1}, t_{i+1}) - \mathcal{V}(\ell_2)$. If $x \in \mathcal{V}(s_1, t_1, \dots, s_i, t_i, s_{i+1})$ we are done by the inner induction hypothesis and strong determinism of \mathcal{R} . So assume $x \in \mathcal{V}(t_{i+1})$. From strong determinism of \mathcal{R} , (6.1), (6.2), and the inner induction hypothesis we have that $y\sigma'_1 = y\sigma'_2$ for all $y \in \mathcal{V}(s_{i+1})$ and thus $s_{i+1}\sigma'_1 = s_{i+1}\sigma'_2$.

With this we can find a join between $t_{i+1}\sigma'_1$ and $t_{i+1}\sigma'_2$ by applying the induction hypothesis twice as shown in the diagram below:



Since t_{i+1} is strongly irreducible and σ'_1 and σ'_2 are normalized, this yields $t_{i+1}\sigma'_1 = t_{i+1}\sigma'_2$ and thus $x\sigma'_1 = x\sigma'_2$.

3. We are left with the case that there is a variable position q' in ℓ_1 such that $q = q'r'$ for some position r' . Let x be the variable $\ell_1|_{q'}$. Then $x\sigma_1|_{r'} = \ell_2\sigma_2$, which implies $x\sigma_1 \rightarrow_{\mathcal{R}}^* x\sigma_1[r_2\sigma_2]_{r'}$. Now let τ be the substitution such that $\tau(x) = x\sigma_1[r_2\sigma_2]_{r'}$ and $\tau(y) = \sigma_1(y)$ for all $y \neq x$. Further, let τ' be a normalized substitution where $\tau'(y)$ is some normal form of $\tau(y)$ (which we know must exist) for all y , so $y\tau \rightarrow_{\mathcal{R}}^* y\tau'$ for all y . Moreover, note that

$$y\sigma_1 \xrightarrow[\mathcal{R}]{} y\tau \text{ for all } y. \quad (6.3)$$

We have $u' = \ell_1\sigma_1[r_2\sigma_2]_q = \ell_1\sigma_1[x\tau]_{q'} \rightarrow_{\mathcal{R}}^* \ell_1\tau$, and thus $u' \rightarrow_{\mathcal{R}}^* \ell_1\tau'$. From (6.3) we have $r_1\sigma_1 \rightarrow_{\mathcal{R}}^* r_1\tau$ and thus $t' = r_1\sigma_1 \rightarrow_{\mathcal{R}}^* r_1\tau'$. Finally, we show that $\ell_1\tau' \rightarrow_{\mathcal{R}} r_1\tau'$, concluding the proof of $t' \downarrow_{\mathcal{R}} u'$. To this end, let $s_i \twoheadrightarrow t_i \in c_1$. By (6.3) and the definition of τ' we obtain $s_i\sigma_1 \rightarrow_{\mathcal{R}}^* t_i\sigma_1 \rightarrow_{\mathcal{R}}^* t_i\tau'$ and $s_i\sigma_1 \rightarrow_{\mathcal{R}}^* s_i\tau'$. Then $s_i\tau' \downarrow_{\mathcal{R}} t_i\tau'$ by the induction hypothesis and also $s_i\tau' \rightarrow_{\mathcal{R}}^* t_i\tau'$, since t_i is strongly irreducible. \square

6.5.1. Certification

There are some complications for employing Theorem 6.5.5 in practice. Quasi-decreasingness, strong irreducibility, and joinability of CCPs are all undecidable in general. For quasi-decreasingness we fall back to the sufficient criterion that a DCTRS is quasi-decreasing if its unraveling is terminating. This result was formalized by Winkler and Thiemann [177] and is already available in `IsaFoR`. A sufficient condition for strong irreducibility is *absolute irreducibility*.

Definition 6.5.6 (Absolute irreducibility ✓, absolute determinism ✓). *We call a term t absolutely \mathcal{R} -irreducible if none of its non-variable subterms unify with any variable-disjoint variant of left-hand sides of rules in the CTRS \mathcal{R} . A DCTRS is called absolutely deterministic (or ADCTRS for short) if for each rule all right-hand sides of conditions are absolutely \mathcal{R} -irreducible.*

The proof of the following result [7, Lemma 4.1(a,b)] is immediate.

Lemma 6.5.7. *For a term t and a CTRS \mathcal{R} :*

- *If t is absolutely \mathcal{R} -irreducible, then t is also strongly \mathcal{R} -irreducible.* ✓
- *If \mathcal{R} is absolutely deterministic, then \mathcal{R} is also strongly deterministic.* ✓

We replace joinability of CCPs by infeasibility [135] (already part of `IsaFoR`) together with two further criteria which rely on *contextual rewriting*.

Definition 6.5.8 (Contextual rewriting ✓). *Consider a set C of equations between terms which we call a context.⁵ First we define a function $\bar{\cdot}$ on terms such that \bar{t} is the term t where each variable $x \in \mathcal{V}(C)$ is replaced by a fresh constant \bar{x} . (Below we sometimes call such constants Skolem constants.) Moreover, let \bar{C} denote the set C where all variables have been replaced by fresh constants \bar{x} . For a CTRS \mathcal{R} we can make a contextual rewrite step, denoted by $s \rightarrow_{\mathcal{R},C} t$, if we can make a conditional rewrite step with respect to the CTRS $\mathcal{R} \cup \bar{C}$ from \bar{s} to \bar{t} .*

We formalize soundness of contextual rewriting [7, Lemma 4.2] as follows:

Lemma 6.5.9. *If $s \rightarrow_{\mathcal{R},C}^* t$ then $s\sigma \rightarrow_{\mathcal{R}}^* t\sigma$ for all substitutions σ satisfying C .* ✓

Proof. Consider the auxiliary function $[t]_\sigma$ that replaces each Skolem constant \bar{x} in t by $\sigma(x)$, that is, it works like applying a substitution to a term, but to Skolem constants instead of variables. Note that $[\bar{t}]_\sigma = t\sigma$ whenever $\mathcal{V}(t) \subseteq \mathcal{V}(C)$. Now we show by induction on n that

$$s \rightarrow_{\mathcal{R} \cup \bar{C},n} t \text{ implies } [s]_\sigma \rightarrow_{\mathcal{R},n}^* [t]_\sigma \quad (\star)$$

for any σ satisfying C . The base case is trivial. In the inductive step we have a rule $\ell \rightarrow r \leftarrow c \in \mathcal{R} \cup \bar{C}$, a position p , and a substitution τ such that $s|_p = \ell\tau$, $t = s[r\tau]_p$,

⁵This has nothing to do with the usual use of the word *context* in rewriting which refers to a term with a hole.

and $u\tau \rightarrow_{\mathcal{R} \cup \overline{C}, n}^* v\tau$ for all $u \approx v \in c$. If $\ell \rightarrow r \Leftarrow c$ is a rule in \mathcal{R} , then we obtain the contextual rewriting sequence $[u\tau]_\sigma \rightarrow_{\mathcal{R} \cup \overline{C}, n}^* [v\tau]_\sigma$ for all $u \approx v \in c$ by the induction hypothesis. Then we show $s \rightarrow_{\mathcal{R} \cup \overline{C}, n+1}^* t$ by induction on the context $s[\cdot]_p$. Otherwise, $\ell \rightarrow r \Leftarrow c \in \overline{C}$ and thus c is empty, $\ell\tau = \ell$, and $r\tau = r$, since \overline{C} is an unconditional ground TRS. Moreover, there is a rule $\ell' \rightarrow r' \in C$ (thus also $\mathcal{V}(\ell', r') \subseteq \mathcal{V}(C)$) such that $\overline{\ell'} = \ell$ and $\overline{r'} = r$. Again, the final result follows by induction on $s[\cdot]_p$.

Assume $s \rightarrow_{\mathcal{R}, C} t$. Then $\overline{s} \rightarrow_{\mathcal{R} \cup \overline{C}, n} \overline{t}$ for some level n . Let \tilde{t} denote the extension of \overline{t} where all variables x in t (not just those in $\mathcal{V}(C)$) are replaced by corresponding fresh constants \overline{x} . Note that $\tilde{t} = \overline{t}\{x \mapsto \overline{x} \mid x \in \text{Var}\}$ for every term t . But then we also have $\tilde{s} \rightarrow_{\mathcal{R} \cup \overline{C}, n} \tilde{t}$ since conditional rewriting is closed under substitutions. Note that $[\tilde{t}]_\sigma = t\sigma$ for all t . Thus taking \tilde{s} and \tilde{t} for s and t in (\star) we obtain $s\sigma \rightarrow_{\mathcal{R}, n}^* t\sigma$. Since we just established the desired property for single contextual rewrite steps it is straightforward to extend it to rewrite sequences. \square

In the following example we illustrate contextual rewriting and how to apply the above lemma.

Example 6.5.10. Consider the CTRS \mathcal{R} consisting of the four rules

$$\begin{array}{ll} f(x, y) \rightarrow f(g(s(x)), y) \Leftarrow c(g(x)) \rightarrow c(a) & g(s(x)) \rightarrow x \\ f(x, y) \rightarrow f(x, h(s(y))) \Leftarrow c(h(y)) \rightarrow c(a) & h(s(x)) \rightarrow x \end{array}$$

and the context C containing the two equations

$$c(g(x)) \approx c(a) \qquad c(h(y)) \approx c(a)$$

We have the sequence

$$f(x, y) \xrightarrow{\mathcal{R}, C} f(g(s(x)), y) \xrightarrow{\mathcal{R}, C} f(g(s(x)), h(s(y)))$$

The first step is justified by $f(\overline{x}, \overline{y}) \rightarrow_{\mathcal{R} \cup \overline{C}} f(g(s(\overline{x})), \overline{y})$ using the first rule of \mathcal{R} as well as the first equation of \overline{C} to satisfy its condition. For the second step we use $f(g(s(\overline{x})), \overline{y}) \rightarrow_{\mathcal{R} \cup \overline{C}} f(g(s(\overline{x})), h(s(\overline{y})))$ employing the second rule of \mathcal{R} and the second equation of \overline{C} to satisfy its condition. From Lemma 6.5.9 we get that for all substitutions σ that satisfy C we have $f(x, y)\sigma \rightarrow^* f(g(s(x)), h(s(y)))\sigma$. In this example the only satisfying substitution is $\sigma = \{x \mapsto s(a), y \mapsto s(a)\}$ employing rules three and four of \mathcal{R} .

Lemma 6.5.9 is the key to overcome the undecidability issues of conditional rewriting. For example, for joinability of CCPs the problem is that a single joining sequence (as is usual in certificates for TRSs) does not prove joinability for all satisfying substitutions. However, contextual rewriting has this property.

Now we are able to define the two promised criteria for CCPs that employ contextual rewriting: *context-joinability* and *unfeasibility*.

Definition 6.5.11 (Unfeasibility \checkmark , context-joinability \checkmark). Let $s \approx t \Leftarrow c$ be a CCP induced by an overlap between variable-disjoint variants $\ell_1 \rightarrow r_1 \Leftarrow c_1$ and $\ell_2 \rightarrow r_2 \Leftarrow c_2$ of rules in \mathcal{R} with mgu μ . We say that the CCP is *unfeasible* if we can find terms u, v , and w such that

6. Certified Confluence of Conditional Rewriting

1. for all σ that satisfy c we have $\ell_1\mu\sigma \succ u\sigma$,
2. $u \rightarrow_{\mathcal{R},c}^* v$,
3. $u \rightarrow_{\mathcal{R},c}^* w$, and
4. v and w are both strongly irreducible and $v \not\sim w$.

Moreover, we call the CCP context-joinable if there exists some term u such that $s \rightarrow_{\mathcal{R},c}^* u$ and $t \rightarrow_{\mathcal{R},c}^* u$.

Example 6.5.12. Consider the CTRS $\mathcal{R}_{\text{last}}$ consisting of the two rules

$$\text{last}(x : y) \rightarrow x \Leftarrow y \rightarrow \text{nil} \qquad \text{last}(x : y) \rightarrow \text{last}(y) \Leftarrow y \rightarrow z : v$$

$\mathcal{R}_{\text{last}}$ is quasi-decreasing with respect to some well-founded order \succ . Moreover, the CTRS has the CCP $x \approx \text{last}(y) \Leftarrow c$ with $c = \{y \rightarrow \text{nil}, y \rightarrow z : v\}$. This CCP is unfeasible because for all satisfying substitutions σ we have $\text{last}(x : y)\sigma \succ y\sigma$, $y \rightarrow_{\mathcal{R}_{\text{last}},c}^* z : v$, $y \rightarrow_{\mathcal{R}_{\text{last}},c}^* \text{nil}$, and both $z : v$ and nil are strongly irreducible and not unifiable.

Now, look at the arbitrary CCP $x \approx \text{min}(\text{nil}) \Leftarrow c$ with $c = \{\text{min}(\text{nil}) \rightarrow x\}$. Since $x \rightarrow_{\mathcal{R},c}^* x$ and $\text{min}(\text{nil}) \rightarrow_{\mathcal{R},c}^* x$ it is context-joinable (regardless of the actual CTRS \mathcal{R}).

Due to Lemma 6.5.9 above, context-joinability implies joinability of a CCP for arbitrary satisfying substitutions. The rationale for the definition of unfeasibility is a little bit more technical, since it only makes sense inside the proof (by induction) of the theorem below. Basically, unfeasibility is defined in such a way that unfeasible CCPs contradict the confluence of all \succ -smaller terms, which we obtain as induction hypothesis.

In the original paper the definition of quasi-reductivity requires its order to be closed under substitutions. This property is used in the proof of [7, Theorem 4.2]. By a small change to the definition of unfeasibility we avoid this requirement for our extension to quasi-decreasingness.

We are finally ready to state a more applicable version of Theorem 6.5.5:

Theorem 6.5.13. Let the ADCTRS \mathcal{R} be quasi-decreasing. Then \mathcal{R} is confluent if all CCPs are context-joinable, unfeasible, or infeasible. ✓

Proof. We denote the well-founded order on terms that we get from quasi-decreasingness by \succ . Unfortunately, we cannot directly reuse Theorem 6.5.5 and its proof, since we need our sufficient criteria in the induction hypothesis. However, the new proof is quite similar. It only differs in case (1), where we consider a CCP.

- a. If the CCP is context-joinable, we obtain a join with respect to contextual rewriting which we can easily transform into a join with respect to \mathcal{R} by an application of Lemma 6.5.9 because we have a substitution satisfying the conditions of the CCP.
- b. If the CCP is unfeasible, we obtain two diverging contextual rewrite sequences. Again since there is a substitution satisfying the conditions of the CCP we may employ Lemma 6.5.9 to get two diverging conditional \mathcal{R} -rewrite sequences. Because $\ell_1\sigma \succ_{\text{st}} t_0$

we can use the induction hypothesis to get a join between the two end terms. But from the definition of unfeasibility we also know that the end points are not unifiable (and hence are not the same) and cannot be rewritten (because of strong irreducibility), leading to a contradiction.

- c. Finally, if the CCP is infeasible, then there is no substitution that satisfies its conditions, contradicting the fact that we already have such a substitution. \square

Have a look at the following example to see Theorem 6.5.13 in action.

Example 6.5.14. Consider the quasi-decreasing ADCTRS \mathcal{R} consisting of the following six rules:

$$\text{min}(x : \text{nil}) \rightarrow x \quad (6.4) \quad x \leq 0 \rightarrow \text{false} \quad (6.7)$$

$$\text{min}(x : xs) \rightarrow x \Leftarrow x \leq \text{min}(xs) \rightarrow \text{true} \quad (6.5) \quad 0 \leq s(y) \rightarrow \text{true} \quad (6.8)$$

$$\text{min}(x : xs) \rightarrow \text{min}(xs) \Leftarrow x \leq \text{min}(xs) \rightarrow \text{false} \quad (6.6) \quad s(x) \leq s(y) \rightarrow x \leq y \quad (6.9)$$

\mathcal{R} has 6 CCPs, 3 modulo symmetry:

$$x \approx x \Leftarrow x \text{min}(\text{nil}) \rightarrow \text{true} \quad (6.1, 6.2)$$

$$x \approx \text{min}(\text{nil}) \Leftarrow x \text{min}(\text{nil}) \rightarrow \text{false} \quad (6.1, 6.3)$$

$$x \approx \text{min}(xs) \Leftarrow x \text{min}(xs) \rightarrow \text{true}, x \leq \text{min}(xs) \rightarrow \text{false} \quad (6.2, 6.3)$$

To conclude confluence of the system it remains to check its CCPs. The first one, (6.1,6.2), is trivially context-joinable because the left- and right-hand sides coincide, (6.1,6.3) is infeasible since $\text{tcap}(x \leq \text{min}(\text{nil})) = x \leq \text{min}(\text{nil})$ and false are not unifiable, and (6.2,6.3) is unfeasible because with contextual rewriting we can reach the two non-unifiable normal forms true and false starting from $x \leq \text{min}(xs)$. Hence, we conclude confluence of \mathcal{R} by Theorem 6.5.13.

6.5.2. Certification Challenges

One of the main challenges towards actual certification is typically disregarded on paper: the definition of critical pairs may yield an infinite set of CCPs even for finite CTRSs. This is because we have to consider arbitrary variable-disjoint variants of rules. However, a hypothetical certificate would only contain those CCPs that were obtained from some specific variable-disjoint variants of rules. Now the argument typically goes as follows: *modulo variable renaming there are only finitely many CCPs. Done.*

However, this reasoning is valid only for properties that are either closed under substitution or at least invariant under renaming of variables. For joinability of plain critical pairs—arguably the most investigated case—this is indeed easy. But when it comes to contextual rewriting we spent a considerable amount of work on some results about permutations that were not available in IsaFoR.

To illustrate the issue, consider the abstract specification of the check function *check-CCPs*, such that *isOK* (*check-CCPs* \mathcal{R}) implies that each of the CCPs of \mathcal{R} is either

6. Certified Confluence of Conditional Rewriting

unfeasible, context-joinable, or infeasible. To this end we work modulo the assumption that we already have sound check functions for the latter three properties, which is nicely supported by Isabelle’s locale mechanism:

```

locale a194-spec =
  fixes  $v_x$  and  $v_y$ 
    and check-context-joinable
    and check-infeasible
    and check-unfeasible
  assumes  $v_x$  and  $v_y$  are injective
    and  $\text{ran}(v_x) \cap \text{ran}(v_y) = \emptyset$ 
    and isOk (check-context-joinable  $\mathcal{R} \ s \ t \ C$ )  $\implies \exists u. s \rightarrow_{\mathcal{R},C}^* u \wedge t \rightarrow_{\mathcal{R},C}^* u$ 
  ...


```

For technical reasons, our formalization uses two locales (*a194-ops*, *a194-spec*) here. We just list the required properties of the renaming functions v_x and v_y and the soundness assumption for *check-context-joinable*.

Now what would a certificate contain and how would we have to check it? Amongst other things, the certificate would contain a finite set of CCPs \mathcal{C}' that were computed by some automated tool. Internally, our certifier computes its own finite set of CCPs \mathcal{C} where variable-disjoint variants of rules are created by fixed injective variable renaming functions v_x and v_y , whose ranges are guaranteed to be disjoint. The former prefixes the character “x” and the latter the character “y” to all variable names, hence the names. At this point we have to check that for each CCP in \mathcal{C} there is one in \mathcal{C}' that is its variant, which is not too difficult. More importantly, we have to prove that whenever some desired property P , say context-joinability, holds for any CCP, then P also holds for all of its variants (including the one that is part of \mathcal{C}).

To this end, assume that we have a CCP resulting from a critical overlap of the two rules $\ell_1 \rightarrow r_1 \Leftarrow c_1$ and $\ell_2 \rightarrow r_2 \Leftarrow c_2$ at position p with mgu μ . This means that there exist permutations π_1 and π_2 such that $(\ell_1 \rightarrow r_1 \Leftarrow c_1)\pi_1$ and $(\ell_2 \rightarrow r_2 \Leftarrow c_2)\pi_2$ are both in \mathcal{R} . In our certifier, mgus are computed by the function $\text{mgu}(s, t)$ which either results in *None*, if $s \not\sim t$, or in *Some* μ such that μ is an mgu of s and t , otherwise. Moreover, variable-disjointness of rules is ensured by v_x and v_y , so that we actually call $\text{mgu}(\ell_1|_p\pi_1v_x, \ell_2\pi_2v_x)$ for computing a concrete CCP corresponding to the one we assumed above. Thus, we need to show that $\text{mgu}(\ell_1|_p, \ell_2) = \text{Some } \mu$ also implies that $\text{mgu}(\ell_1|_p\pi_1v_x, \ell_2\pi_2v_y) = \text{Some } \mu'$ for some mgu μ' . Moreover, we are interested in the relationship between μ and μ' with respect to the variables in both rules. Previously—for an earlier formalization of infeasibility [134]—IsaFoR only contained a result that related both unifiers modulo some arbitrary substitution (that is, not necessarily a renaming).

Unfortunately, contextual rewriting is not closed under arbitrary substitutions. Nevertheless, contextual rewriting is closed under permutations, provided the permutation is also applied to C .

Lemma 6.5.15. *For every permutation π we have that $s\pi \rightarrow_{R,C\pi}^* t\pi$ iff $s \rightarrow_{R,C}^* t$. *

It remains to show that μ and μ' differ basically only by a renaming (at least on the variables of our two rules), which is covered by the following lemma.

Lemma 6.5.16. *Let $\text{mgu}(s, t) = \text{Some } \mu$ and $\mathcal{V}(s, t) \subseteq S \cup T$ for two finite sets of variables S and T with $S \cap T = \emptyset$. Then, there exist a substitution μ' and a permutation π such that for arbitrary permutations π_1 and π_2 : $\text{mgu}(s\pi_1 v_x, t\pi_2 v_y) = \text{Some } \mu'$, $\mu = \pi_1 \mu' v_x \pi$ $[S]$, and $\mu = \pi_2 \mu' v_y \pi$ $[T]$.* ✓

Proof. Let $h(x) = xv_x \pi_1$ if $x \in S$ and $h(x) = xv_y \pi_2$, otherwise. Then, since h is bijective between $S \cup T$ and $h(S \cup T)$ we can obtain a permutation π for which $\pi = h$ $[S \cup T]$. We define $\mu' = \pi^- \mu$ and abbreviate $s\pi_1 v_x$ and $t\pi_2 v_y$ to s' and t' , respectively. Note that $s' = s\pi$ and $t' = t\pi$. Since μ is an mgu of s and t we have $s\mu = t\mu$, which further implies $s'\mu' = t'\mu'$. But then μ' is a unifier of s' and t' and thus there exists some μ'' for which $\text{mgu}(s', t') = \text{Some } \mu''$ and $s'\mu'' = t'\mu''$.

We now show that μ' is also most general. Assume $s'\tau = t'\tau$ for some τ . Then $s\pi\tau = t\pi\tau$ and thus there exists some δ such that $\pi\tau = \mu\delta$ (since μ is most general). But then $\pi^- \pi\tau = \pi^- \mu\delta$ and thus $\tau = \mu'\delta$. Hence, μ' is most general.

Since μ'' is most general too, it only differs by a renaming, say π' , from μ' , that is, $\mu'' = \pi' \mu'$. This yields $\mu = \pi_1 \mu'' v_x \pi'^-$ $[S]$ and $\mu = \pi_2 \mu'' v_y \pi'^-$ $[T]$, and thus concludes the proof. □

6.5.3. Check Functions

Before we can actually certify the output of CTRS confluence tools with CeTA, we have to provide an executable check function for each property that is required to apply Theorem 6.5.13 and prove its soundness. For the check functions for infeasibility see Section 6.7. The check functions for quasi-decreasigness are not described in this article (see [177]). It remains to provide new check functions for absolute irreducibility, absolute determinism, contextual rewrite sequences, context-joinability, and unfeasibility together with their corresponding soundness proofs. For absolute irreducibility we provide the check function *check-airr*, employing existing machinery from IsaFoR for renaming and unification, and prove:

Lemma 6.5.17. *If $\text{isOK}(\text{check-airr } \mathcal{R} \ t)$, then the term t is absolutely \mathcal{R} -irreducible.* ✓

This, in turn, is used to define the check function *check-adtrs* and the accompanying lemma for ADCTRSs.

Lemma 6.5.18. *$\text{isOK}(\text{check-adtrs } \mathcal{R})$ iff \mathcal{R} is an ADCTRS.* ✓

Concerning contextual rewriting, we provide the check function *check-csteps* for conditional rewrite sequences together with the following lemma:

Lemma 6.5.19. *Given a CTRS \mathcal{R} , a set of conditions C , two terms s and t , and a list of conditional rewrite proofs ps , we have that $\text{isOK}(\text{check-csteps}(\mathcal{R} \cup \overline{C}) \ \overline{s} \ \overline{t} \ \overline{ps})$ implies $s \rightarrow_{\mathcal{R}, C}^* t$.* ✓

6. Certified Confluence of Conditional Rewriting

Although conditional rewriting is decidable in our setting (strong determinism and quasi-decreasingness), we require a *conditional rewrite proof* to provide all the necessary information for checking a single conditional rewrite step (the employed rule, position, and substitution; source and target terms; and recursively, a list of rewrite proofs for each condition of the applied rule). That way, we avoid having to formalize a rewriting engine for conditional rewriting in `IsaFoR`. With a check function for contextual rewrite sequences in place, we can easily give the check function *check-context-joinable* with the corresponding lemma:

Lemma 6.5.20. *Given a CTRS \mathcal{R} , three terms s , t , and u , conditions C , and two lists of conditional rewrite proofs ps and qs , we have that*

$$\text{isOK } (\text{check-context-joinable } (u, ps, qs) \mathcal{R} s t C)$$

implies that there exists some term u' such that $s \rightarrow_{\mathcal{R}, C}^ u' \mathcal{R}_{C^*}^* t$.* ✓

Here *check-context-joinable* is a concrete implementation of the homonymous function from the `a194-spec` locale. We further give the check function *check-unfeasible* and the accompanying soundness lemma:

Lemma 6.5.21. *Given a quasi-decreasing CTRS \mathcal{R} , two variable-disjoint variants of rules $\rho_1: \ell_1 \rightarrow r_1 \Leftarrow c_1$ and $\rho_2: \ell_2 \rightarrow r_2 \Leftarrow c_2$ in \mathcal{R} , an mgu μ of $\ell_1|_p$ and ℓ_2 for some position p , a set of conditions C such that $C = c_1\mu, c_2\mu$, three terms t , u , and v , and two lists of conditional rewrite proofs ps and qs , we have that*

$$\text{isOK } (\text{check-unfeasible } (t, u, v, ps, qs, \rho_1, \rho_2) \mathcal{R} \ell_1 \mu C)$$

implies that there exist three terms t' , u' , and v' such that for all σ we have $\ell_1\mu\sigma \succ t'\sigma$, whenever σ satisfies C , $u' \mathcal{R}_{C^}^* t' \rightarrow_{\mathcal{R}, C}^* v'$, u' and v' are both strongly irreducible, and $u' \not\sim v'$.* ✓

Again, *check-unfeasible* is a concrete implementation of the function of the same name from the `a194-spec` locale and it additionally performs various sanity checks.

At this point, interpreting the `a194-spec` locale using the three check functions *check-context-joinable*, *check-infeasible*, and *check-unfeasible* from above yields the concrete function *check-CCPs*, which is used in the final check *check-al94*.

Lemma 6.5.22. *Given a quasi-decreasing CTRS \mathcal{R} , a list of context-joinability certificates c , a list of infeasibility certificates i , and a list of unfeasibility certificates u . Then, $\text{isOK } (\text{check-al94 } c i u \mathcal{R})$ implies confluence of \mathcal{R} .* ✓

The formalization of the methods described in this section can be found in the following `IsaFoR` theory files:

```
thys/Conditional_Rewriting/    thys/Rewriting/
  AL94.thy                    Renaming_Interpretations.thy
  AL94_Impl.thy
  Quasi_Decreasingness.thy
```

In the next section we turn our attention to non-confluence. More specifically, we present simple methods for finding witnesses that establish non-confluence.

6.6. Finding Witnesses for Non-Confluence of CTRSs

To prove non-confluence of a CTRS we have to find a witness, that is, two diverging rewrite sequences starting at the same term whose end points are not joinable.

The first criterion only works for CTRSs that contain at least one unconditional rule of type 4, that is, with extra-variables in the right-hand side.

Lemma 6.6.1. *Given a 4-CTRS \mathcal{R} and an unconditional rule $\rho: \ell \rightarrow r$ in \mathcal{R} where $\text{Var}(r) \not\subseteq \text{Var}(\ell)$ and r is a normal form with respect to \mathcal{R}_u then \mathcal{R} is non-confluent. \checkmark*

Proof. Since $\text{Var}(r) \not\subseteq \text{Var}(\ell)$ we can always find two renamings μ_1 and μ_2 restricted to $\text{Var}(r) \setminus \text{Var}(\ell)$ such that $r\mu_1 \rho \leftarrow \ell\mu_1 = \ell\mu_2 \rightarrow_\rho r\mu_2$ and $r\mu_1 \neq r\mu_2$. As r is a normal form with respect to \mathcal{R}_u also $r\mu_1$ and $r\mu_2$ are (different) normal forms with respect to \mathcal{R}_u (and hence also with respect to \mathcal{R}). Because we found a non-joinable peak \mathcal{R} is non-confluent. \square

The following example features Cops #320 [105, Example 4.18]:

Example 6.6.2 (Cops #320). *Consider the 4-CTRS \mathcal{R} consisting of the two rules*

$$e \rightarrow f(x) \leftarrow l \rightarrow d \qquad A \rightarrow h(x, x)$$

The right-hand side $h(x, x)$ of the second (unconditional) rule is a normal form with respect to the underlying TRS \mathcal{R}_u and the only variable occurring in it does not appear in its left-hand side A . So by Lemma 6.6.1 \mathcal{R} is non-confluent.

A natural candidate for diverging situations are the critical peaks of a CTRS. We base our next criterion on the analysis of *unconditional* critical pairs (CPs) of CTRSs. This restriction is necessary to guarantee the existence of the actual peak. If we would also allow conditional CPs, we first would have to check for infeasibility, since infeasibility is undecidable in general these checks are potentially very costly (see for example [156]).

Lemma 6.6.3. *Given a CTRS \mathcal{R} and an unconditional CP $s \approx t$ of it. If s and t are not joinable with respect to \mathcal{R}_u then \mathcal{R} is non-confluent. \checkmark*

Proof. The CP $s \approx t$ originates from a critical overlap between two unconditional rules $\rho_1: \ell_1 \rightarrow r_1$ and $\rho_2: \ell_2 \rightarrow r_2$ for some mgu μ of $\ell_1|_p$ and ℓ_2 such that $s = \ell_1\mu[r_2\mu]_p \leftarrow \ell_1\mu[\ell_2\mu]_p \rightarrow r_1\mu = t$. Since s and t are not joinable with respect to \mathcal{R}_u they are of course also not joinable with respect to \mathcal{R} and we have found a non-joinable peak. So \mathcal{R} is non-confluent. \square

The following example features Cops #271 [17]:

Example 6.6.4 (Cops #271). *Consider the 3-CTRS \mathcal{R} consisting of the four rules*

$$p(q(x)) \rightarrow p(r(x)) \quad q(h(x)) \rightarrow r(x) \quad r(x) \rightarrow r(h(x)) \leftarrow s(x) \rightarrow 0 \quad s(x) \rightarrow 1$$

First of all we can immediately drop the third rule because we can never satisfy its condition and so it does not influence the rewrite relation of \mathcal{R} . This results in the TRS

6. Certified Confluence of Conditional Rewriting

\mathcal{R}' . Now the left- and right-hand sides of the unconditional CP $\mathbf{p}(\mathbf{r}(z)) \approx \mathbf{p}(\mathbf{r}(\mathbf{h}(z)))$ are not joinable because they are two different normal forms with respect to the underlying TRS $\mathcal{R}'_{\mathbf{u}}$. Hence \mathcal{R} is not confluent by Lemma 6.6.3.

While the above lemmas are easy to check and we have fast methods to do so they are also rather ad hoc. A more general but potentially very expensive way to search for non-joinable forks is to use conditional narrowing [90].

Definition 6.6.5 (Conditional narrowing). *Given a CTRS \mathcal{R} we say that s (conditionally) narrows to t , written $s \rightsquigarrow_{\sigma} t$ if there is a variant of a rule $\rho: \ell \rightarrow r \Leftarrow c \in \mathcal{R}$, such that $\text{Var}(s) \cap \text{Var}(\rho) = \emptyset$ and $u \rightsquigarrow_{\sigma}^* v$ for all $u \approx v \in c$, a position $p \in \text{Pos}_{\mathcal{F}}(s)$, a unifier⁶ σ of $s|_p$ and ℓ , and $t = s[r]_p\sigma$. For a narrowing sequence $s_1 \rightsquigarrow_{\sigma_1} s_2 \rightsquigarrow_{\sigma_2} \cdots \rightsquigarrow_{\sigma_{n-1}} s_n$ of length n we write $s_1 \rightsquigarrow_{\sigma}^n s_n$ where $\sigma = \sigma_1\sigma_2 \cdots \sigma_{n-1}$. If we are not interested in the length we also write $s \rightsquigarrow_{\sigma}^* t$.*

The following property of narrowing carries over from the unconditional case:

Property 6.6.6. *If $s \rightsquigarrow_{\sigma} t$ then $s\sigma \rightarrow t\sigma$ with the same rule that was employed in the narrowing step. Moreover, if $s_1 \rightsquigarrow_{\sigma_1} s_2 \rightsquigarrow_{\sigma_2} \cdots \rightsquigarrow_{\sigma_{n-1}} s_n$ then $s_1\sigma_1\sigma_2 \cdots \sigma_{n-1} \rightarrow s_2\sigma_2 \cdots \sigma_{n-1} \rightarrow \cdots \rightarrow s_n$. Again employing the same rule for each rewrite step as in the corresponding narrowing step.*

Using conditional narrowing we can now formulate a more general non-confluence criterion.

Lemma 6.6.7. *Given a CTRS \mathcal{R} , if we can find two narrowing sequences $u \rightsquigarrow_{\sigma}^* s$ and $v \rightsquigarrow_{\tau}^* t$ such that $u\sigma\mu = v\tau\mu$ for some mgu μ and $s\sigma\mu$ and $t\tau\mu$ are not $\mathcal{R}_{\mathbf{u}}$ -joinable then \mathcal{R} is non-confluent.*

Proof. Employing 6.6.6 we immediately get the two rewriting sequences $u\sigma \rightarrow_{\mathcal{R}}^* s\sigma$ and $v\tau \rightarrow_{\mathcal{R}}^* t\tau$. Since rewriting is closed under substitutions we have $s\sigma\mu \xrightarrow{\mathcal{R}}^* u\sigma\mu = v\tau\mu \rightarrow_{\mathcal{R}}^* t\tau\mu$. As the two endpoints of these forking sequences $s\sigma\mu$ and $t\tau\mu$ are not joinable by $\mathcal{R}_{\mathbf{u}}$ they are certainly also not joinable by \mathcal{R} . This establishes non-confluence of the CTRS \mathcal{R} . \square

Example 6.6.8. *Remember the 3-CTRS from Example 6.5.3 consisting of the three rules*

$$0 + y \rightarrow y \qquad \mathbf{s}(x) + y \rightarrow x + \mathbf{s}(y) \qquad \mathbf{f}(x, y) \rightarrow z \Leftarrow x + y \Rightarrow z + v$$

Starting from a variant of the left-hand side of the third rule $u = \mathbf{f}(x', y')$ we have a narrowing sequence $\mathbf{f}(x', y') \rightsquigarrow_{\sigma} x_1$ using the variant $\mathbf{f}(x_1, x_2) \rightarrow x_3 \Leftarrow x_1 + x_2 \Rightarrow x_3 + x_4$ of the third rule and the substitution $\sigma = \{x' \mapsto x_1, x_3 \mapsto x_1, x_4 \mapsto x_2\}$. We also have another narrowing sequence $\mathbf{f}(x', y') \rightsquigarrow_{\tau} x_3$ using the same variant of rule three and substitution $\tau = \{x \mapsto x_3 + x_4, x' \mapsto 0, y' \mapsto x_3 + x_4, x_1 \mapsto 0, x_2 \mapsto x_3 + x_4\}$ where for the condition $x_1 + x_2 \Rightarrow x_3 + x_4$ we have the narrowing sequence $x_1 + x_2 \rightsquigarrow_{\tau} x_3 + x_4$, using a variant of the first rule $0 + x \rightarrow x$. Finally, there is an mgu $\mu = \{x_1 \mapsto 0, x_2 \mapsto x_3 + x_4\}$ such that $u\sigma\mu = \mathbf{f}(0, x_3 + x_4) = u\tau\mu$. Moreover, $x_1\sigma\mu = 0$ and $x_3\tau\mu = x_3$ are two different normal forms. Hence \mathcal{R} is non-confluent by Lemma 6.6.7.

⁶In our implementation we start from an mgu of $s|_p$ and ℓ and extend it while trying to satisfy the conditions.

6.6.1. Implementation

Starting from its first participation in the confluence competition (CoCo)⁷ in 2014 ConCon 1.2.0.3 came equipped with some non-confluence heuristics. Back then it only used Lemmas 6.6.1 and 6.6.3 and had no support for certification of the output. In the next two years (ConCon 1.3.0 and 1.3.2) we focused on other developments [134, 135, 156, 157] and nothing changed for the non-confluence part. For CoCo 2017 we have added Lemma 6.6.7 employing conditional narrowing to ConCon 1.5.1 and the output of all of the non-confluence methods is now certifiable by CeTA.

Our implementation of Lemma 6.6.1 takes an unconditional rule $\rho: \ell \rightarrow r$, a substitution $\sigma = \{x \mapsto y\}$ with $x \in \text{Var}(r) \setminus \text{Var}(\ell)$ and y fresh with respect to ρ and builds the non-joinable fork $r \rho \leftarrow \ell \rightarrow_\rho r\sigma$.

For Lemma 6.6.3 we have three concrete implementations that consider an overlap from which an unconditional CP $s \approx t$ arises: The first of which just takes this overlap and then checks that s and t are two different normal forms with respect to \mathcal{R}_u . The second employs the `tcap`-function to check for non-joinability, that is, it checks whether `tcap(s)` and `tcap(t)` are not unifiable. The third makes a special call to the TRS confluence tool CSI [184] providing the underlying TRS \mathcal{R}_u as well as the unconditional CP $s \approx t$ where all variables in s and t have been replaced by fresh constants. We issue the following command:

```
csi -s '(nonconfluence -nonjoinability -steps 0 -tree)[30]' -C RT
```

The strategy `'(nonconfluence -nonjoinability -steps 0 -tree)[30]'` tells CSI to check non-joinability of two terms using tree automata techniques. Here `'-steps 0'` means that CSI does not rewrite the input terms further before checking non-joinability (this would be unsound in our setting). The timeout is set to 30 seconds. To encode the two terms for which we want to check non-joinability in the input we set CSI to read relative-rewriting input (`'-C RT'`). We provide \mathcal{R}_u in the usual Cops-format and add one line for the CP $s \approx t$ where its “grounded” left- and right-hand sides are related by `'->='`, that is, we encode it as a relative-rule. This is necessary to distinguish the unconditional CP from the rewrite rules.

Now, for an implementation of Lemma 6.6.7 we have to be careful to respect the freshness requirement of the variables in the used rule for every narrowing step with respect to all the previous terms and rules. The crucial point is to efficiently find the two narrowing sequences, to this end we first restrict the set of terms from which to start narrowing. As a heuristic we only consider the left-hand sides of rules of the CTRS under consideration. Next we also prune the search space for narrowing. Here we restrict the length of the narrowing sequences to at most three. In experiments on Cops allowing sequences of length four or more did not yield additional non-confluence proofs but slowed down the tool significantly to the point where we lost other proofs. Further, we also limit the recursion depth of conditional narrowing by restricting the level (see the definition of the conditional rewrite relation in the Preliminaries) to at most two. Again, we set this limit as tradeoff after thorough experiments on Cops. Finally, we use 6.6.6 to

⁷<http://project-coco.uibk.ac.at>

6. Certified Confluence of Conditional Rewriting

translate the forking narrowing sequences into forking conditional rewriting sequences. In this way we generate a lot of forking sequences so we only use fast methods, like non-unifiability of the `tcaps` of the endpoints or that they are different normal forms, to check for non-joinability of the endpoints. Calls to `CSI` are too expensive in this context.

6.6.2. Certification

Certification is quite similar for all of the described methods. We have to provide a non-confluence witness, that is, a non-joinable fork. So besides the CTRS \mathcal{R} under investigation we also need to provide the starting term s , the two endpoints of the fork t and u , as well as certificates for $s \rightarrow_{\mathcal{R}}^+ t$ and $s \rightarrow_{\mathcal{R}}^+ u$, and a certificate that t and u are not joinable. For the forking rewrite sequences we reuse one of our recent formalization [136] to build certificates. We also want to stress that because of 6.6.6 we did not have to formalize conditional narrowing because going from narrowing to rewrite sequences is already done in `ConCon` and in the certificate only the rewrite sequences show up. For the non-joinability certificate of t and u there are three options: either we state that t and u are two different normal forms or that `tcap`(t) and `tcap`(u) are not unifiable; both of these checks are performed within `CeTA`; or, when the witness was found by an external call to `CSI`, we just include the generated non-joinability certificate.

The formalization of the methods described in this section can be found in the following `IsaFoR` theory files:

`thys/Conditional_Rewriting/Non_Confluence2.thy`

The next section discusses several methods to show infeasibility of conditional critical pairs. Specifically the techniques described in Sections 6.4 and 6.5 benefit from these infeasibility results.

6.7. Infeasibility of Conditional Critical Pairs

The confluence methods detailed in Sections 6.4 and 6.5 among other things also analyze the conditional critical pairs of a CTRS. Being able to ignore so called *infeasible* critical pairs simplifies this analysis. See for example `Cops #326` [105, \mathcal{R}_{20} in Example 6.7]:

Example 6.7.1 (`Cops #326`). *The single CCP (modulo symmetry)*

$$\text{tp}_2(\text{nil}, x) \approx \text{tp}_2(x : y, z) \Leftarrow \text{isnoc}(\text{nil}) \Rightarrow \text{tp}_2(y, z)$$

of the oriented 3-CTRS \mathcal{R} consisting of the two rules

$$\text{isnoc}(y : \text{nil}) \rightarrow \text{tp}_2(\text{nil}, y) \quad \text{isnoc}(x : ys) \rightarrow \text{tp}_2(x : xs, y) \Leftarrow \text{isnoc}(ys) \Rightarrow \text{tp}_2(xs, y)$$

is infeasible since `isnoc`(`nil`) is in normal form. Hence \mathcal{R} is orthogonal (modulo infeasibility) and thus confluent.


In this section we present infeasibility methods for oriented 3-CTRSs, one of the most popular types of conditional rewriting. In such systems extra variables in conditions and right-hand sides of rewrite rules are allowed to a certain extent. Moreover, for oriented CTRSs satisfiability of the conditions amounts to reachability. As a consequence of the latter, establishing infeasibility is similar to the problem of eliminating edges in dependency graph approximations, a problem which has been investigated extensively in the literature. The difference is that we deal with CTRSs and the terms we test may share variables.

For brevity, we speak about non-reachability, non-meetability, and non-joinability of two terms s and t , when we actually mean that the respective property holds for arbitrary substitution instances $s\sigma$ and $t\tau$.

In the sequel we summarize the methods that we have analyzed and adapted for infeasibility.

6.7.1. Unification


A widely-used method to check for non-reachability is to try to unify the **tcap** of the source term with the target term; which is the de facto standard for approximating dependency graphs for termination proofs. Remember, the **tcap**-function approximates the topmost part of a term, its “cap,” that does not change under rewriting (see Section 6.2). It is well known that $\text{tcap}(s) \not\sim t$ implies non-reachability of t from s . Typical “pen and paper” definitions (like the one in the preliminaries) rely on replacing subterms by “fresh variables” making them somewhat hard to formalize as already remarked by Thiemann and Sternagel [165]. Instead of inventing fresh variables out of thin air, the **lsaFoR**-version of **tcap** replaces every variable occurrence by the symbol \square . The resulting terms behave like ground multihole contexts—we call them *ground contexts*—and they are intended to represent the set of all terms resulting from replacing all “holes” by arbitrary terms. This is made formal by the *substitution instance class* of a ground context.

Definition 6.7.2 (Substitution instance class ). *The substitution instance class $\llbracket t \rrbracket$ of a ground context t is defined recursively by.*

$$\llbracket t \rrbracket := \begin{cases} \mathcal{T}(\mathcal{F}, \text{Var}) & \text{if } t = \square \\ \{f(s_1, \dots, s_n) \mid s_i \in \llbracket t_i \rrbracket\} & \text{if } t = f(t_1, \dots, t_n) \end{cases}$$

Note that for variable-disjoint terms s and t , unifiability coincides with $s\sigma = t\tau$ for some, not necessarily identical, substitutions σ and τ . Thus asking whether a term t unifies with a variable-disjoint term represented by the ground context s is equivalent to checking whether $t\sigma \in \llbracket s \rrbracket$ for some substitution σ . The latter is called *ground context matching* and shown to be decidable by an efficient algorithm by Thiemann and Sternagel [165]. Thus we can define an efficient executable version of **tcap** by:


6. Certified Confluence of Conditional Rewriting

Definition 6.7.3 (Efficient \mathbf{tcap} ) .

$$\mathbf{tcap}_{\mathcal{R}}(t) = \begin{cases} \square & \text{if } t \text{ is a variable} \\ \square & \text{if } t = f(t_1, \dots, t_n) \text{ and } \ell\sigma \in \llbracket u \rrbracket \text{ for some } \sigma \text{ and } \ell \rightarrow r \in \mathcal{R} \\ u & \text{otherwise} \end{cases}$$


where $u = f(\mathbf{tcap}_{\mathcal{R}}(t_1), \dots, \mathbf{tcap}_{\mathcal{R}}(t_n))$ and \mathcal{R} is a TRS. We omit \mathcal{R} if it is clear from context.

This version of \mathbf{tcap} is sound, that is, whenever we can reach a term t from an instance of a term s then t is in the substitution instance class of $\mathbf{tcap}(s)$.

Lemma 6.7.4. *If $s\sigma \rightarrow_{\mathcal{R}}^* t$ then $t \in \llbracket \mathbf{tcap}(s) \rrbracket$.* 

Then checking non-reachability of t from s amounts to deciding whether there does not exist a substitution τ such that $t\tau \in \llbracket \mathbf{tcap}(s) \rrbracket$, for which we use the more succinct notation $\mathbf{tcap}(s) \not\sim t$ almost everywhere else in this article.

While the above definition of \mathbf{tcap} and the corresponding soundness lemma were already present in *IsaFoR*, the following easy extension also allows us to test for non-joinability.

Lemma 6.7.5. *If $s\sigma \rightarrow_{\mathcal{R}}^* \cdot \mathcal{R}^* \leftarrow t\tau$ then $\llbracket \mathbf{tcap}(s) \rrbracket \cap \llbracket \mathbf{tcap}(t) \rrbracket \neq \emptyset$.* 



Proof. We have $s\sigma \rightarrow_{\mathcal{R}}^* u$ and $t\tau \rightarrow_{\mathcal{R}}^* u$ for some u . By Lemma 6.7.4 we have $u \in \mathbf{tcap}(s)$ and $u \in \mathbf{tcap}(t)$. \square

Fortunately the same techniques that are used to obtain an algorithm for ground context matching can be reused for *ground context unifiability*, that is, checking $\llbracket \mathbf{tcap}(s) \rrbracket \cap \llbracket \mathbf{tcap}(t) \rrbracket \neq \emptyset$ (elsewhere in this article we use the notation $\mathbf{tcap}(s) \not\sim \mathbf{tcap}(t)$).

Now for checking infeasibility of a CCP we use the underlying TRS and if there is more than one condition we collect the left- and right-hand sides separately under a fresh function symbol as follows:

Corollary 6.7.6 (Infeasibility via \mathbf{tcap}). *Let \mathcal{R} be an oriented 3-CTRS. A CCP*

$$u \approx v \Leftarrow s_1 \rightarrow t_1, \dots, s_k \rightarrow t_k$$

is infeasible if $\mathbf{tcap}_{\mathcal{R}_u}(\mathbf{cs}(s_1, \dots, s_k)) \not\sim \mathbf{cs}(t_1, \dots, t_k)$ (where \mathbf{cs} is a fresh function symbol of arity k). (Combination of  and )


Example 6.7.7. *Consider Cops #326 from Example 6.7.1. The CCP*

$$\mathbf{tp}_2(\mathbf{nil}, x) \approx \mathbf{tp}_2(x : y, z) \Leftarrow \mathbf{isnoc}(\mathbf{nil}) \rightarrow \mathbf{tp}_2(y, z)$$

is infeasible by Corollary 6.7.6 because $\mathbf{tcap}_{\mathcal{R}_u}(\mathbf{isnoc}(\mathbf{nil})) = \mathbf{isnoc}(\mathbf{nil}) \not\sim \mathbf{tp}_2(x_1, x_2) = \mathbf{tcap}_{\mathcal{R}_u}(\mathbf{tp}_2(y, z))$.

6.7.2. Symbol Transition Graph

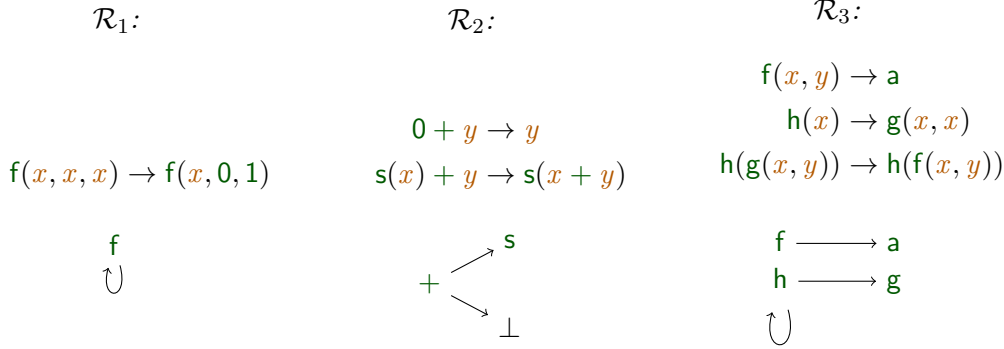
One shortcoming of the `tcap` method is that although later during unification we have to consider the target term anyway it first only considers the starting term. Maybe we could gain power if we use knowledge about the structure of the target from the start? This consideration is the basis for the so called *symbol transition graph* [153]. The idea is simple enough, we consider the root symbols of left- and right-hand sides of rewrite rules of a TRS (for CTRSs we perform an overapproximation by using the underlying TRS)⁸ and collect the resulting dependencies in a graph.

Definition 6.7.8 (Symbol transition graph ). *Given a TRS \mathcal{R} the edges of its symbol transition graph are given by the relation*

$$\sqsubset_{\text{stg}} := \{(\text{root}(\ell), \text{root}(r)) \mid \ell \rightarrow r \in \mathcal{R}\}$$

To clarify this concept let us first look at some examples.


Example 6.7.9. *Below we give the symbol transition graphs of the following three TRSs \mathcal{R}_1 , \mathcal{R}_2 , and \mathcal{R}_3 , respectively.*



Example 6.7.10. *For the CTRS of Example 6.7.1 the symbol transition graph consists of a single edge:*

$$\text{isnoc} \longrightarrow \text{tp}_2$$

Now we can use the following lemma for reachability analysis of TRSs.

Lemma 6.7.11. *For two terms $s = f(\dots)$ and $t = g(\dots)$ if $s \rightarrow^* t$ then either $f = g$, $f \sqsubset_{\text{stg}}^+ g$, or $f \sqsubset_{\text{stg}}^+ \perp$. *

Proof. We prove this by induction on the length of the rewrite sequence $s \rightarrow^k t$. In the base case we have $s = t$ and hence $f = g$. Now in the step case we look at the sequence $s \rightarrow u \rightarrow^k t$. If u is a variable then $s \rightarrow u$ is a root step with a collapsing rule and hence

⁸There is an inductive version of the symbol transition graph specifically for CTRSs due to Sternagel and Yamada [153]. However, it is not yet formalized and thus we concentrate on the TRS version, which we formalized in Isabelle/HOL.

6. Certified Confluence of Conditional Rewriting

$f \sqsubset_{\text{stg}} \perp$ and thus also $f \sqsubset_{\text{stg}}^+ \perp$. Otherwise there is some function symbol h such that $u = h(\dots)$. If we have a non-root step then $f = h$ and we are done. Otherwise we have $f \sqsubset_{\text{stg}} h$. From the induction hypothesis we know that one of $h = g$, $h \sqsubset_{\text{stg}}^+ g$, or $h \sqsubset_{\text{stg}}^+ \perp$ holds. So either $f \sqsubset_{\text{stg}} h = g$, $f \sqsubset_{\text{stg}} h \sqsubset_{\text{stg}}^+ g$, or $f \sqsubset_{\text{stg}} h \sqsubset_{\text{stg}}^+ \perp$, which finishes the proof. \square

From the previous lemma we immediately get the following non-reachability result:

Corollary 6.7.12. *If $f \neq g$ and neither $f \sqsubset_{\text{stg}}^+ g$ nor $f \sqsubset_{\text{stg}}^+ \perp$ then $f(\dots) \not\rightarrow^* g(\dots)$.*

Example 6.7.13. *Consider the TRS \mathcal{R}_3 from Example 6.7.9. Do we have a rewrite sequence $\mathbf{f}(x, y)\sigma \rightarrow^* \mathbf{g}(x, y)\tau$ for some substitutions σ and τ ? Since $\text{tcap}(\mathbf{f}(x, y)) = z \sim \mathbf{g}(x, y)$ we cannot conclude non-reachability using tcap . Then again $\mathbf{f} \neq \mathbf{g}$, $\mathbf{f} \not\sqsubset_{\text{stg}}^+ \mathbf{g}$, and $\mathbf{f} \not\sqsubset_{\text{stg}}^+ \perp$. So $\mathbf{g}(x, y)$ is not reachable from $\mathbf{f}(x, y)$ with respect to \mathcal{R}_3 by Corollary 6.7.12.*

Example 6.7.14. *For Cops #547 (see also Example 6.4.10) to get infeasibility of its CCP we have to show that either $x\sigma \not\rightarrow^* a\tau$, $x\sigma \not\rightarrow^* b\tau$, or $\mathbf{cs}(x, x)\sigma \not\rightarrow^* \mathbf{cs}(a, b)\tau$ for a fresh compound symbol \mathbf{cs} and any two substitutions σ and τ . Unfortunately, we cannot employ Corollary 6.7.12 here. If we look at the two conditions separately the left-hand sides are variables and looking at the combined conditions we have the same function symbol \mathbf{cs} on the left- and right-hand sides.*

In the next section we will see a way to combine the best of both tcap and the symbol transition graph.

6.7.3. Decomposing Reachability Problems


Both, the tcap method and the symbol transition graph do have their pros and cons. For example, in contrast to the tcap method, the symbol transition graph takes information about the target term into account, then again, it does not recursively look at the whole term, like tcap , but only at the root symbols.

In this section we introduce a modular framework for reachability, that allows us to decompose a given problem into several smaller ones. The binary relation \sqsubset that we employ to show non-reachability between two terms has to have certain properties to be usable for decomposition. These properties are summarized in the following definition (where $\xrightarrow{\epsilon}$ denotes rewrite steps below the root position).

Definition 6.7.15 (Decomposition \checkmark). *A binary relation on terms \sqsubset admits decomposition if both*


1. $s\sigma \sqsubset t\tau$ implies $s \sqsubset t$ for all substitutions σ and τ and
2. $s \not\sqsubset t$ together with $s \rightarrow^* t$ implies $s \xrightarrow{\epsilon}^* t$.

Using the binary relation \sqsubset we can define a *decomposition function* that deconstructs a single reachability problem into several (smaller) ones.


Definition 6.7.16 (Abstract decomposition function ). The abstract decomposition function $D_{s,t}$ takes a reachability problem (s, t) (meaning: “Is t reachable from s ?”) as input and recursively computes a set of (possibly easier) reachability problems with respect to the relation \sqsubset .

$$D_{s,t} := \begin{cases} D_{s_1,t_1} \cup \dots \cup D_{s_n,t_n} & \text{if } s = f(s_1, \dots, s_n), t = f(t_1, \dots, t_n), \text{ and } s \not\sqsubset t \\ \{(s, t)\} & \text{otherwise} \end{cases}$$

With this function in place we can now formally define when decomposition is admissible for a reachability problem.


Lemma 6.7.17. If \sqsubset admits decomposition and $s\sigma \rightarrow^* t\tau$ then also $u\sigma \rightarrow^* v\tau$ for all $(u, v) \in D_{s,t}$. 

Of course in our setting we want to use decomposition of a reachability problem to show non-reachability. To do that we first have to specify an abstract non-reachability test that we later instantiate with concrete checks.

Definition 6.7.18 (Abstract non-reachability check ). If two functional terms $s = f(\dots)$ and $t = g(\dots)$ have different root symbols and s and t are not in the relation \sqsubset then t is not reachable from s . Otherwise it is. This is tested by the abstract non-reachability check $\text{nonreach}(s, t)$ defined as follows:

$$\text{nonreach}(s, t) := \begin{cases} f \neq g \wedge s \not\sqsubset t & \text{if } s = f(\dots) \text{ and } t = g(\dots) \\ \text{false} & \text{otherwise} \end{cases}$$

Finally, we are ready to state a non-reachability lemma.

Lemma 6.7.19. If \sqsubset admits decomposition and $\text{nonreach}(s, t)$ holds then $s\sigma \not\rightarrow^* t\tau$. 



Proof. Assume to the contrary that $s\sigma \rightarrow^* t\tau$. Because \sqsubset admits decomposition we have

1. $s\sigma \sqsubset t\tau \implies s \sqsubset t$
2. $s \not\sqsubset t \implies s \rightarrow^* t \implies s \xrightarrow{\epsilon}^* t$

From 1 and $\text{nonreach}(s, t)$ we have that $s\sigma \not\sqsubset t\tau$. From this, our starting assumption, and 2 we have $s\sigma \xrightarrow{\epsilon}^* t\tau$. But this implies $\text{root}(s) = \text{root}(t)$ which contradicts $\text{nonreach}(s, t)$. \square

Below we give two possible instances of \sqsubset one using the tcap -function of Section 6.7.1 and the other using the symbol transition graph \sqsubset_{stg}^+ of Section 6.7.2.

Definition 6.7.20 ($\sqsubset_{\text{tcap}}, \sqsubset_{\text{rs}}$). We define

- $s \sqsubset_{\text{tcap}} t$ iff there exists a rule $\ell \rightarrow r$ in \mathcal{R} such that $\text{tcap}(s) \sim \ell$, and 
- $s \sqsubset_{\text{rs}} t$ iff at least one of the following properties holds: s is a variable, t is a variable, $\text{root}(s) \sqsubset_{\text{stg}}^+ \text{root}(t)$, or $\text{root}(s) \sqsubset_{\text{stg}}^+ \perp$. 

6. Certified Confluence of Conditional Rewriting

Example 6.7.21. Consider the TRS \mathcal{R}_4 consisting of the four rules

$$f(x) \rightarrow a \quad p(s(x)) \rightarrow b \quad h(x) \rightarrow g(x) \quad h(f(b)) \rightarrow h(f(p(a)))$$

and the two terms $s = f(p(a))$ and $t = f(b)$. We have $s \sqsubset_{\text{tcap}} t$ since $\text{tcap}(s) = x_1$ unifies with the left-hand side of every rule in \mathcal{R}_4 but $s \not\sqsubset_{\text{rs}} t$ because s and t are neither variables, nor $f \sqsubset_{\text{stg}}^+ f$, nor $f \sqsubset_{\text{stg}}^+ \perp$. Moreover, for the two terms $s' = p(a)$ and $t' = b$ we have $s' \not\sqsubset_{\text{tcap}} t'$ since $\text{tcap}(s') = p(a)$ does not unify with the left-hand side of any rule in \mathcal{R}_4 but $s' \sqsubset_{\text{rs}} t'$ because the second rule of \mathcal{R}_4 yields $p \sqsubset_{\text{stg}} b$.

Both of these relations admit decomposition of reachability problems.

Lemma 6.7.22. Both, \sqsubset_{tcap} and \sqsubset_{rs} admit decomposition. ✓✓

Proof. We first consider \sqsubset_{tcap} . Assume $s\sigma \sqsubset_{\text{tcap}} t\tau$, so there exists a rule $\ell \rightarrow r \in \mathcal{R}$ such that $\text{tcap}(s\sigma) \sim \ell$. Remember that the latter is just a shorthand for the existence of a substitution μ such that $\ell\mu \in \llbracket s\sigma \rrbracket$. By induction on u we have that $\llbracket \text{tcap}(u\mu) \rrbracket \subseteq \llbracket \text{tcap}(u) \rrbracket$ for any substitution μ . Hence also $\ell\mu \in \llbracket s \rrbracket$ and thus $\text{tcap}(s) \sim \ell$. But then by definition $s \sqsubset_{\text{tcap}} t$. Now assume $s \rightarrow^* t$ and $s \not\sqsubset_{\text{tcap}} t$. From the latter we have that $\text{tcap}(s) \not\sim \ell$ for all $\ell \rightarrow r \in \mathcal{R}$. If there would be any root step in $s \rightarrow^* t$ then $\text{tcap}(s) \sim \ell$ for some rule $\ell \rightarrow r \in \mathcal{R}$. Hence $s \xrightarrow{\epsilon}^* t$. So \sqsubset_{tcap} admits decomposition.

Next consider \sqsubset_{rs} . Assume $s\sigma \sqsubset_{\text{rs}} t\tau$. So by definition either $s\sigma \in \text{Var}$, $t\tau \in \text{Var}$, $\text{root}(s\sigma) \sqsubset_{\text{stg}}^+ \text{root}(t\tau)$, or $\text{root}(s\sigma) \sqsubset_{\text{stg}}^+ \perp$. But then obviously $s \sqsubset_{\text{rs}} t$ because for any term u and any substitution μ if $u\mu \in \text{Var}$ certainly also $u \in \text{Var}$ and if $u \notin \text{Var}$ then $\text{root}(u\mu) = \text{root}(u)$. Now assume $s \rightarrow^* t$ and $s \not\sqsubset_{\text{rs}} t$. From the latter we have that neither s nor t are variables, as well as $\text{root}(s) \not\sqsubset_{\text{stg}}^+ \text{root}(t)$ and $\text{root}(s) \not\sqsubset_{\text{stg}}^+ \perp$. But that means that there is no root step in $s \rightarrow^* t$ and hence $s \xrightarrow{\epsilon}^* t$. So also \sqsubset_{rs} admits decomposition. □

Furthermore, if two relations admit decomposition then also their intersection does.

Lemma 6.7.23. If \sqsubset_1 and \sqsubset_2 admit decomposition then so does $\sqsubset_1 \cap \sqsubset_2$. ✓

So in our implementation we employ the intersection of the two relations defined earlier and obtain the following result by combining Lemma 6.7.19, Lemma 6.7.22, and Lemma 6.7.23:

Corollary 6.7.24. If $\text{nonreach}(s, t)$ holds for $\sqsubset = \sqsubset_{\text{tcap}} \cap \sqsubset_{\text{rs}}$, then $s\sigma \not\rightarrow^* t\tau$.

We call the above instance of **nonreach** *generalized tcap*.

This is stronger than relying on the two relations separately as shown in the following example.

Example 6.7.25. Consider the TRS \mathcal{R}_4 from Example 6.7.21 above. Look at the two terms $s = f(p(a))$ and $t = f(b)$ and assume we want to know whether $s\sigma \rightarrow^* t\tau$ for any substitutions σ and τ . We employ the abstract non-reachability check and the abstract decomposition function. In our first attempt we instantiate \sqsubset with \sqsubset_{rs} . Since the root symbols of s and t are the same $\text{nonreach}(s, t)$ does not hold. We try to decompose the

problem and get $D_{s,t} = \{(p(a), b)\}$ because $s \not\sqsubseteq_{rs} t$. Unfortunately, $p(a) \sqsubseteq_{rs} b$ which means that $\text{nonreach}(p(a), b)$ is not true. Clearly \sqsubseteq_{rs} does not work.

Let us try to instantiate \sqsubseteq with \sqsubseteq_{tcap} . Since $s \sqsubseteq_{tcap} t$ neither $\text{nonreach}(s, t)$ holds nor can we decompose the problem and we immediately have to give up.

So let's finally try to instantiate \sqsubseteq with $\sqsubseteq_{tcap} \cap \sqsubseteq_{rs}$. The root symbols of s and t are the same so $\text{nonreach}(s, t)$ does not hold. Since $s \not\sqsubseteq_{rs} t$ also $s (\sqsubseteq_{tcap} \cap \sqsubseteq_{rs}) t$ does not hold and thus $D_{s,t} = \{(p(a), b)\}$. We have $p(a) \not\sqsubseteq_{tcap} b$ and thus $\text{nonreach}(p(a), b)$, which together with Lemma 6.7.19 yields $p(a)\sigma \not\vdash^* b\tau$. From this we get non-reachability of t from s by Lemma 6.7.17 and we are done.

6.7.4. Exact Tree Automata Completion

What is generally known as tree automata completion today was introduced by Genet in 1998 [41, 42]. Already in 1996 Jacquemard [64] used a similar concept to show decidability of reachability for linear and growing TRSs. His proof was based on the construction of a tree automaton that accepts the set of ground terms which are normalizable with respect to a given linear and growing TRS \mathcal{R} . If we replace the automaton recognizing \mathcal{R} -normal forms in Jacquemard's construction by an arbitrary automaton \mathcal{A} we arrive at a tree automaton that accepts the \mathcal{R} -ancestors of the language of \mathcal{A} .

We need some basic definitions and auxiliary lemmas before we present the construction of this *ancestor automaton* in detail.

Definition 6.7.26 (Ground-instances \checkmark). *The set of ground-instances of a term t , that is, the set of terms s such that $s = t\sigma$ for some ground substitution σ is denoted by $\Sigma(t)$.*

Definition 6.7.27 (Growingness, linear growing approximation). *A TRS \mathcal{R} is called growing if for all $\ell \rightarrow r \in \mathcal{R}$ the variables in $\text{Var}(\ell) \cap \text{Var}(r)$ occur at depth at most one in ℓ . Given a TRS \mathcal{R} the linear growing approximation is defined as any linear growing TRS obtained from \mathcal{R} by linearizing the left-hand sides, renaming the variables in the right-hand sides that occur at a depth greater than one in the corresponding left-hand side, and finally also linearizing the right-hand sides.⁹ The linear growing approximation of a TRS \mathcal{R} is denoted by $g(\mathcal{R})$.*

Definition 6.7.28 (Ground-instance transitions Δ_t \checkmark). *Let $[t]$ denote a term $t \in \mathcal{T}(\mathcal{F}, \text{Var})$ where all variable-occurrences have been replaced by a fresh symbol \square (similar to Definitions 6.7.2 and 6.7.3). Using such terms as states we define the set Δ_t that contains all transitions which are needed to recognize all ground-instances of a term $t \in \mathcal{T}(\mathcal{F}, \text{Var})$ in state $[t]$.*

$$\Delta_t = \begin{cases} \{f([t_1], \dots, [t_n]) \rightarrow [t]\} \cup \bigcup_{1 \leq i \leq n} \Delta_{t_i} & \text{if } t = f(t_1, \dots, t_n) \\ \{f(\square, \dots, \square) \rightarrow \square \mid f \in \mathcal{F}\} & \text{otherwise} \end{cases}$$

Note that if t is not linear this actually gives an overapproximation. The next lemma holds by definition of Δ_t .

⁹Note that this definition of the linear growing approximation is ambiguous, because the second step depends on the first step and we have a choice of how to linearize the variables (see also [89]).

6. Certified Confluence of Conditional Rewriting

Lemma 6.7.29. *For any subterm s of any term t if there is a sequence $u \rightarrow_{\Delta_t}^+ [s]$ then u is a ground-instance of s , and vice versa if t is linear.* ✓

We now use Δ_t to define an automaton for the ground-instances of t .

Definition 6.7.30 (Ground-instance automaton $\mathcal{A}_{\Sigma(t)}$ ✓). *Let Q_t denote the set of states occurring in Δ_t then we call the tree automaton $\mathcal{A}_{\Sigma(t)} = \langle \mathcal{F}, Q_t, \{[t]\}, \Delta_t \rangle$ the ground-instance automaton for t .*

Lemma 6.7.31. *The language of $\mathcal{A}_{\Sigma(t)}$ is an overapproximation of the set of ground-instances of t in general and an exact characterization if t is linear.* ✓

Using the concept of ground-instance automaton we are now able to define a tree automaton which accepts all \mathcal{R} -ancestors of a given regular set of ground terms using exact tree automata completion.

Definition 6.7.32 (Ancestor automaton $\text{anc}_{\mathcal{R}}(\mathcal{A})$ ✓). *Given a tree automaton $\mathcal{A} = \langle \mathcal{F}, Q_{\mathcal{A}}, Q_f, \Delta \rangle$ whose states are all accessible, and a linear and growing TRS \mathcal{R} the construction proceeds as follows.*

First we extend the set of transitions of \mathcal{A} in such a way that we can match left-hand sides of rules in \mathcal{R} . This yields the set of transitions $\Delta_0 = \Delta \cup \bigcup_{\ell \rightarrow r \in \mathcal{R}} \Delta_{\ell}$. Let $\mathcal{A}_0 = \langle \mathcal{F}, Q, Q_f, \Delta_0 \rangle$ where Q denotes the set of states in Δ_0 . We have to ensure (for example by using the disjoint union of states) that for any state q which is used in both Δ and some Δ_{ℓ} , the terms which can reach it are the same ($\{t \mid t \rightarrow_{\Delta}^+ q\} = \{t \mid t \rightarrow_{\Delta_{\ell}}^+ q\}$). Then the language does not change, that is, $L(\mathcal{A}_0) = L(\mathcal{A})$.

Finally, we saturate Δ_0 by inference rule (\dagger) in order to extend the language by \mathcal{R} -ancestors, that is, if we can reach a state q from an instance of a right-hand side of a rule in \mathcal{R} we add a transition which ensures that q is reachable from the corresponding left-hand side.¹⁰

$$\frac{f(\ell_1, \dots, \ell_n) \rightarrow r \in \mathcal{R} \quad r\theta \rightarrow_{\Delta_k}^* q}{f(q_1, \dots, q_n) \rightarrow q \in \Delta_{k+1}} \quad (\dagger)$$

Here $\theta: \text{Var}(r) \rightarrow Q$ is a state substitution and $q_i = \ell_i\theta$ if ℓ_i is a variable in r and $q_i = [\ell_i]$ otherwise. Note that this inductive definition possibly adds many new transitions from Δ_k to Δ_{k+1} .

Since \mathcal{R} is finite, the number of states is finite, and we do not introduce new states using (\dagger) , this process terminates after finitely many steps resulting in the set of transitions Δ_m . Also note that Δ_k is monotone with respect to k , that is, $\Delta_k \subseteq \Delta_{k+1}$ for all $k \geq 0$. We call $\text{anc}_{\mathcal{R}}(\mathcal{A}) = \langle \mathcal{F}, Q, Q_f, \Delta_m \rangle$ the \mathcal{R} -ancestors automaton for \mathcal{A} . It is easy to show that $L(\mathcal{A}_0) \subseteq L(\text{anc}_{\mathcal{R}}(\mathcal{A}))$.

Theorem 6.7.33. *Given a tree automaton \mathcal{A} as well as a linear and growing TRS \mathcal{R} the language of $\text{anc}_{\mathcal{R}}(\mathcal{A})$ is exactly the set of \mathcal{R} -ancestors of $L(\mathcal{A})$.* ✓

¹⁰This is symmetric to resolving compatibility violations in the tree automata completion by Genet [41, 42].

Proof. First we prove that $(\rightarrow_{\mathcal{R}}^*)[L(\mathcal{A})] \subseteq L(\text{anc}_{\mathcal{R}}(\mathcal{A}))$. Pick a term $s \in (\rightarrow_{\mathcal{R}}^*)[L(\mathcal{A})]$. That means that there is a rewrite sequence $s \rightarrow_{\mathcal{R}}^k t$ of length $k \geq 0$ for some $t \in L(\mathcal{A})$. We proceed by induction on k . If $k = 0$ then $s = t$ and hence $s \in L(\text{anc}_{\mathcal{R}}(\mathcal{A}))$. Now assume $k = k' + 1$ for some $k' \geq 0$ then there is a rewrite sequence $s = C[f(\ell_1, \dots, \ell_n)\sigma] \rightarrow_{\mathcal{R}} C[r\sigma] \rightarrow_{\mathcal{R}}^{k'} t$ for some context C , rewrite rule $f(\ell_1, \dots, \ell_n) \rightarrow r \in \mathcal{R}$, and substitution σ . By induction hypothesis $C[r\sigma] \in L(\text{anc}_{\mathcal{R}}(\mathcal{A}))$. But that means that there is a state substitution $\theta: \text{Var}(r) \rightarrow Q$, a state $q \in Q$, and a final state $q_f \in Q_f$ such that $C[r\sigma] \rightarrow_{\Delta_m}^* C[r\theta] \rightarrow_{\Delta_m}^* C[q] \rightarrow_{\Delta_m}^* q_f$. From the construction using rule (\dagger) we know that there is a transition $f(q_1, \dots, q_n) \rightarrow q \in \Delta_m$ such that $q_i = \ell_i\theta$ if $\ell_i \in \text{Var}(r)$ and $q_i = [\ell_i]$ otherwise. If $\ell_i \in \text{Var}(r)$ then $\ell_i\sigma \rightarrow_{\Delta_m}^+ \ell_i\theta$ and otherwise $\ell_i\sigma \rightarrow_{\Delta_m}^+ [\ell_i]$ for all $1 \leq i \leq n$. Hence in both cases $\ell_i\sigma \rightarrow_{\Delta_m}^+ q_i$. But then we can construct the sequence $s = C[f(\ell_1\sigma, \dots, \ell_n\sigma)] \rightarrow_{\Delta_m}^* C[f(q_1, \dots, q_n)] \rightarrow_{\Delta_m} C[q] \rightarrow_{\Delta_m}^* q_f$ and hence $s \in L(\text{anc}_{\mathcal{R}}(\mathcal{A}))$.

For the other direction we prove the following two properties for all sequences $s \rightarrow_{\Delta_m}^+ q$:

1. If $q = [t]$ for some subterm of a left-hand side of a rule in \mathcal{R} then $s \in (\rightarrow_{\mathcal{R}}^*)[\Sigma(t)]$.
2. If $q \in Q_f$ then $s \in (\rightarrow_{\mathcal{R}}^*)[L(\mathcal{A})]$.

The proof for both properties works along the same lines. We sketch the one for the first property here. From the construction using rule (\dagger) we know that $s \rightarrow_{\Delta_k}^+ [t]$ for some $k \geq 0$. We proceed by induction on k . If $k = 0$ then $s \rightarrow_{\Delta_0}^+ [t]$. By construction of \mathcal{A}_0 and Lemma 6.7.29 we have $s \in \Sigma(t)$ and hence also $s \in (\rightarrow_{\mathcal{R}}^*)[\Sigma(t)]$. Now assume that $k = k' + 1$ for some $k' \geq 0$. By induction hypothesis $s \rightarrow_{\Delta_{k'}}^+ [t]$ implies $s \in (\rightarrow_{\mathcal{R}}^*)[\Sigma(t)]$ for all terms s and t . Consider the set $\Delta_{k'+1} \setminus \Delta_{k'}$ of transitions which were newly added in $\Delta_{k'+1}$. We continue by a second induction on the size of $\Delta_{k'+1} \setminus \Delta_{k'}$. If it is empty we have a $\Delta_{k'}$ -sequence and may simply close the proof with an application of the first induction hypothesis. Otherwise we have some set Δ and transition $\rho: f(q_1, \dots, q_n) \rightarrow q'$ that was created from some rule $\ell \rightarrow r \in \mathcal{R}$ with $\ell = f(\ell_1, \dots, \ell_n)$ and the sequence $r\theta \rightarrow_{\Delta_{k'}}^* q'$ by an application of rule (\dagger) such that $\{\rho\} \uplus \Delta \subseteq \Delta_{k'+1} \setminus \Delta_{k'}$. The second induction hypothesis is if $s \rightarrow_{\Delta \cup \Delta_{k'}}^+ [t]$ then $s \in (\rightarrow_{\mathcal{R}}^*)[\Sigma(t)]$. Let m denote the number of steps that use ρ . We continue by a third induction on m . If $m = 0$ the sequence from s to $[t]$ only used transitions in $\Delta \cup \Delta_{k'}$ and using the second induction hypothesis we are done. Otherwise there is some $m' \geq 0$ such that $m = m' + 1$ and the induction hypothesis is that for all terms s, t if $s \rightarrow_{\Delta \cup \Delta_{k'}}^+ [t]$ using ρ only m' times then $s \in (\rightarrow_{\mathcal{R}}^*)[\Sigma(t)]$. Now we look at the first step using ρ in the sequence, that is, $s = D[f(s_1, \dots, s_n)] \rightarrow_{\Delta \cup \Delta_{k'}}^* C[f(q_1, \dots, q_n)] \rightarrow_{\rho} C[q'] \rightarrow_{\Delta_{k'+1}}^* [t]$. Note that from this we get $D[u] \rightarrow_{\Delta \cup \Delta_{k'}}^* C[u]$ for all terms u .

Next we define a substitution τ such that

$$s \rightarrow_{\mathcal{R}}^* D[\ell\tau] \rightarrow_{\mathcal{R}} D[r\tau] \rightarrow_{\Delta \cup \Delta_{k'}}^* C[r\tau] \rightarrow_{\Delta \cup \Delta_{k'}}^* C[q']$$

This allows us to bypass the ρ -step and so we arrive at a $\Delta_{k'+1}$ -sequence from $D[r\tau]$ to $[t]$ containing one less ρ -step as shown in Figure 6.4. The construction of τ proceeds as follows: We fix $1 \leq i \leq n$. If ℓ_i is a variable in r define τ_i to be $\{\ell_i \mapsto s_i\}$. Otherwise we

6. Certified Confluence of Conditional Rewriting

$$\begin{array}{c}
s = D[f(s_1, \dots, s_n)] \xrightarrow[\Delta \cup \Delta_{k'}]{*} D[f(q_1, \dots, q_n)] \xrightarrow[\Delta \cup \Delta_{k'}]{*} C[f(q_1, \dots, q_n)] \xrightarrow[\rho]{*} C[q'] \xrightarrow[\Delta_{k'+1}]{*} [t] \\
\downarrow \mathcal{R}^* \qquad \qquad \qquad \downarrow \mathcal{R}^* \qquad \qquad \qquad \downarrow \mathcal{R}^* \qquad \qquad \qquad \downarrow \mathcal{R}^* \qquad \qquad \qquad \downarrow \mathcal{R}^* \\
D[\ell\tau] \xrightarrow[\mathcal{R}]{*} D[r\tau] \xrightarrow[\Delta \cup \Delta_{k'}]{*} C[r\tau] \xrightarrow[\Delta \cup \Delta_{k'}]{*} C[r\theta]
\end{array}$$

Figure 6.4.: Bypassing ρ to close the induction step.

know from the definition of inference rule (\dagger) that $q_i = [\ell_i]$ and $s_i \xrightarrow[\Delta \cup \Delta_{k'}]{+} [\ell_i]$. From that we have that $s_i \in (\rightarrow_{\mathcal{R}}^*)[\Sigma(\ell_i)]$ but that means that there is some substitution τ_i such that $s_i \xrightarrow[\mathcal{R}]{*} \ell_i \tau_i$. Moreover let $\tau' = \{x \mapsto u_x \mid x \in \text{Var}(r) \setminus \text{Var}(\ell)\}$ where u_x is an arbitrary but fixed ground term such that $u_x \xrightarrow[\Delta_0]{*} x\theta$. Since all states in \mathcal{A}_0 are accessible we can always find such a term u_x . Now let τ be the disjoint union of $\tau_1, \dots, \tau_n, \tau'$. This substitution is well-defined because ℓ is linear. By construction of τ we get $s \xrightarrow[\mathcal{R}]{*} D[\ell\tau]$.

Consider a variable x occurring in r . If x also occurs in ℓ we have $x = \ell_i$ for some unique $1 \leq i \leq n$ because \mathcal{R} is growing. But then by construction of τ_i we get $x\tau = \ell_i \tau_i = s_i$. Moreover from the definition of q_i in inference rule (\dagger) we have $q_i = \ell_i \theta = x\theta$. But then $x\tau \xrightarrow[\Delta \cup \Delta_{k'}]{+} x\theta$ from $s_i \xrightarrow[\Delta \cup \Delta_{k'}]{+} q_i$. On the other hand, if x does not occur in ℓ then $x\tau = x\tau'$ and $x\tau' \xrightarrow[\Delta_0]{*} x\theta$ by construction of τ' . So in both cases $r\tau \xrightarrow[\Delta \cup \Delta_{k'}]{*} r\theta$. Together with $r\theta \xrightarrow[\Delta_{k'}]{*} q'$ and $C[q'] \xrightarrow[\Delta_{k'+1}]{*} [t]$ we may construct the sequence $D[r\tau] \xrightarrow[\Delta_{k'+1}]{+} q_f$ which uses ρ only m' times. Using the induction hypothesis we arrive at $D[r\tau] \in (\rightarrow_{\mathcal{R}}^*)[\Sigma(t)]$. Together with $s \xrightarrow[\mathcal{R}]{*} D[\ell\tau] \xrightarrow[\mathcal{R}]{*} D[r\tau]$ this means that $s \in (\rightarrow_{\mathcal{R}}^*)[\Sigma(t)]$ and we are done. \square

Lemma 6.7.34 (Non-reachability via anc). *Let \mathcal{R} be a linear and growing TRS over signature \mathcal{F} . We may conclude non-reachability of t from s if*

$$L(\mathcal{A}_{\Sigma(s)}) \cap L(\text{anc}_{\mathcal{R}}(\mathcal{A}_{\Sigma(t)})) = \emptyset$$

\checkmark

To see this in action look at the following example featuring Cops #494 [135, Example 16]:

Example 6.7.35 (Infeasibility via anc, Cops #494). *Consider the CTRS \mathcal{R} consisting of the following five rules:*

$$\begin{array}{lll}
g(a, x) \rightarrow c \Leftarrow f(x, a) \rightarrow a & f(a, x) \rightarrow a & c \rightarrow c \\
g(x, a) \rightarrow d \Leftarrow f(x, b) \rightarrow b & f(b, x) \rightarrow b &
\end{array}$$

It has two critical pairs

$$c \approx d \Leftarrow f(a, b) \rightarrow b, f(a, a) = a$$

and the symmetric one. Since $\text{tcap}(f(a, b)) = x \sim b$ and $\text{tcap}(f(a, a)) = x \sim a$ as well as $f \sqsupset_{\text{stg}}^+ b$ and $f \sqsupset_{\text{stg}}^+ a$ neither unification nor the symbol transition graph are sufficient to show infeasibility of these critical pairs. On the other hand, since the underlying TRS

6.7. Infeasibility of Conditional Critical Pairs

\mathcal{R}_u is linear and growing, we may construct the tree automaton $\mathcal{A}_{\Sigma(\mathbf{f}(\mathbf{a}, \mathbf{b}))}$ consisting of the three transitions

$$\mathbf{a} \rightarrow [\mathbf{a}] \qquad \mathbf{b} \rightarrow [\mathbf{b}] \qquad \mathbf{f}([\mathbf{a}], [\mathbf{b}]) \rightarrow [\mathbf{f}(\mathbf{a}, \mathbf{b})]$$

where $[\mathbf{f}(\mathbf{a}, \mathbf{b})]$ is the final state, as well as the tree automaton $\mathbf{anc}_{\mathcal{R}_u}(\mathcal{A}_{\Sigma(\mathbf{b})})$ consisting of 20 transitions

$$\begin{array}{llllll} \mathbf{a} \rightarrow \square & \mathbf{a} \rightarrow [\mathbf{a}] & \mathbf{g}(\square, \square) \rightarrow \square & \mathbf{f}([\mathbf{b}], \square) \rightarrow \square & \mathbf{g}(\square, [\mathbf{a}]) \rightarrow [\mathbf{g}(\square, \mathbf{a})] \\ \mathbf{b} \rightarrow \square & \mathbf{b} \rightarrow [\mathbf{b}] & \mathbf{g}(\square, [\mathbf{a}]) \rightarrow \square & \mathbf{f}([\mathbf{a}], \square) \rightarrow [\mathbf{a}] & \mathbf{f}([\mathbf{a}], \square) \rightarrow [\mathbf{f}(\mathbf{a}, \square)] \\ \mathbf{c} \rightarrow \square & \mathbf{c} \rightarrow [\mathbf{c}] & \mathbf{g}([\mathbf{a}], \square) \rightarrow \square & \mathbf{f}([\mathbf{b}], \square) \rightarrow [\mathbf{b}] & \mathbf{g}([\mathbf{a}], \square) \rightarrow [\mathbf{g}(\mathbf{a}, \square)] \\ \mathbf{d} \rightarrow \square & \mathbf{f}(\square, \square) \rightarrow \square & \mathbf{f}([\mathbf{a}], \square) \rightarrow \square & \mathbf{g}([\mathbf{a}], \square) \rightarrow [\mathbf{c}] & \mathbf{f}([\mathbf{b}], \square) \rightarrow [\mathbf{f}(\mathbf{b}, \square)] \end{array}$$

with final state $[\mathbf{b}]$. Because the language of the intersection automaton is empty we have shown infeasibility of the condition $\mathbf{f}(\mathbf{a}, \mathbf{b}) \twoheadrightarrow \mathbf{b}$ by Lemma 6.7.34. So both critical pairs are infeasible.

In the setting of Section 6.4 the right-hand sides of conditions are always linear terms (because of right-stability; see Definition 6.4.2). Hence it is beneficial to start with the ground-instance automaton for the right-hand side of a condition (which in this case is exact) and compute the set of ancestors rather than taking the possibly non-linear left-hand side of a condition, overapproximating the ground-instances and only then computing the descendants of this set. Although this is not necessarily true in the setting of Section 6.5 (there we have no linearity-restriction on the right-hand sides of conditions) we employ the same setup for the sake of convenience.

For one of our main use cases, Theorem 6.4.11, we are restricted to left-linear CTRSs (via almost orthogonality) and linear right-hand sides of conditions (via right-stability). The latter also holds for right-hand sides that are combined by a compound symbol (again by right-stability). We show that in this setting \mathbf{anc} subsumes \mathbf{tcap} (at least in theory and for the forward direction).

Lemma 6.7.36. *Let \mathcal{R} be a left-linear CTRS and t a linear term. If \mathbf{tcap} can show non-reachability of t from s , then so can \mathbf{anc} .*

Proof. Below we write $\mathbf{ren}(t)$ for a linearization of the term t using fresh variables. We proof the contrapositive and assume that \mathbf{anc} cannot show non-reachability. Moreover, let \mathcal{R}' denote the result of applying the linear growing approximation to \mathcal{R}_u . Then there is some term u such that $u \in L(\mathcal{A}_{\Sigma(s)})$ and $u \in L(\mathbf{anc}_{\mathcal{R}'}(\mathcal{A}_{\Sigma(t)}))$. Since t is linear and \mathcal{R}' is linear and growing the latter implies that $u \in (\rightarrow_{\mathcal{R}'}^*)[\Sigma(t)]$ by Theorem 6.7.33 and thus $u \rightarrow_{\mathcal{R}'}^* t\tau$ for some substitution τ . By Lemma 6.7.4, this means that $t\tau \in \llbracket \mathbf{tcap}_{\mathcal{R}'}(u) \rrbracket$. Since $u \in L(\mathcal{A}_{\Sigma(s)})$, it is clearly the case that $u \in \Sigma(\mathbf{ren}(s))$ and thus $u = \mathbf{ren}(s)\sigma$ for some substitution σ . Moreover $\llbracket \mathbf{tcap}_{\mathcal{R}'}(u) \rrbracket \subseteq \llbracket \mathbf{tcap}_{\mathcal{R}'}(\mathbf{ren}(s)) \rrbracket = \llbracket \mathbf{tcap}_{\mathcal{R}'}(s) \rrbracket$. Together with $t\tau \in \llbracket \mathbf{tcap}_{\mathcal{R}'}(u) \rrbracket$ from above, we obtain $t\tau \in \llbracket \mathbf{tcap}_{\mathcal{R}'}(s) \rrbracket$. However, \mathbf{tcap} does only consider the left-hand sides of rules, which are the same in \mathcal{R}' and \mathcal{R}_u , thus also $t\tau \in \llbracket \mathbf{tcap}_{\mathcal{R}_u}(s) \rrbracket$ which implies $\mathbf{tcap}_{\mathcal{R}_u}(s) \sim t$. \square

6. Certified Confluence of Conditional Rewriting

If we also consider the reverse direction, that is, checking if the term s is reachable from t by \mathcal{R}_u^{-1} for some condition $s \twoheadrightarrow t$ in Theorem 6.4.11, then **tcap** may well succeed where **anc** fails, as shown by the next example featuring Cops #546 [135, Example 23].

Example 6.7.37 (**anc** vs. **tcap**, Cops #546). *The oriented 3-CTRS \mathcal{R} consisting of the two rules*

$$g(x) \rightarrow g(x) \Leftarrow g(x) \twoheadrightarrow f(a, b) \qquad g(x) \rightarrow f(x, x)$$

is right-stable and extended properly oriented. It has two symmetric CCPs of the form

$$f(x, x) \approx g(x) \Leftarrow g(x) \twoheadrightarrow f(a, b).$$

*The underlying TRS \mathcal{R}_u is not linear and growing, so if we want to apply **anc** we have to apply a linear growing approximation, resulting for example in \mathcal{R}'*

$$g(x) \rightarrow f(x, y) \qquad g(x) \rightarrow g(x)$$

*But then **anc** is not able to show infeasibility since the language accepted by the tree automaton $\mathcal{A}_{\Sigma(g(x))} \cap \mathbf{anc}_{\mathcal{R}'}(\mathcal{A}_{\Sigma(f(a,b))})$ is not empty and also for the reverse direction $\mathcal{A}_{\Sigma(f(a,b))} \cap \mathbf{anc}_{\mathcal{R}'^{-1}}(\mathcal{A}_{\Sigma(g(x))})$ we get a non-empty language. On the other hand, using the reversed underlying system \mathcal{R}_u^{-1}*

$$f(x, x) \rightarrow g(x) \qquad g(x) \rightarrow g(x)$$

*we have that $\mathbf{tcap}_{\mathcal{R}_u^{-1}}(f(a, b)) = f(a, b) \not\sim g(x)$. So in this case **tcap** succeeds where **anc** fails.*

6.7.5. Exploiting Equalities

Finally, there are four cases where the equality between some terms in the conditions can be exploited to check a CCP for infeasibility. These four situations are depicted in Figure 6.5 and explained below.

Assume we have a CCP $\ell \approx r \Leftarrow c$ where c contains at least two different conditions $s \twoheadrightarrow t$ and $u \twoheadrightarrow v$. Now $s = v$ implies that for c to be satisfiable t has to be reachable from u (see 6.5a). So if we can show that t is not reachable from u we know that the conditions c are infeasible. Similarly for $t = u$ to have any chance to satisfy the conditions v has to be reachable from s otherwise c is infeasible (see 6.5b). On the other hand if $t = v$ then to satisfy the conditions s and u have to be joinable and so from non-joinability of s and u we can conclude infeasibility of c (see 6.5c). Finally, if we are in the setting described in Section 6.4 then if $s = u$ to satisfy the conditions there would have to exist a diverging situation (see 6.5d) but by the assumption of Definition 6.4.9.3 this implies that there is a join between t and v so to proof infeasibility of c it suffices to show non-joinability of t and v .

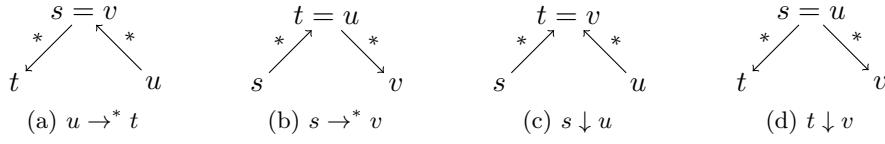


Figure 6.5.: Requirements for the conditions c containing $s \rightarrow t$ and $u \rightarrow v$ to be satisfiable.

6.7.6. Certification

In this section we give an overview of all infeasibility and non-reachability techniques that are supported by our certifier **CeTA** and what kind of information it requires from a certificate in CPF [144] (short for *certification problem format*). Before we come to the special infeasibility condition of Definition 6.4.9, we handle the common case where, given a list of conditions c , we are interested in proving $\sigma, n \not\models c$ for every substitution σ and level n .

Lemma 6.7.38 (Infeasibility certificates). *Given (\mathcal{R}, c) consisting of a CTRS \mathcal{R} and a list of conditions $c = s_1 \rightarrow t_1, \dots, s_k \rightarrow t_k$, infeasibility of c with respect to \mathcal{R} can be certified in one of the following ways:* ✓

1. Provide a fresh function symbol \mathbf{cs} of arity n together with a non-reachability certificate for $(\mathcal{R}_u, \mathbf{cs}(s_1, \dots, s_k), \mathbf{cs}(t_1, \dots, t_k))$.
2. Provide two terms s and t with $s \rightarrow t \in c$, and a non-reachability certificate for (\mathcal{R}_u, s, t) .
3. For an arbitrary subset c' of c , provide an infeasibility certificate for (\mathcal{R}, c') .
4. Provide three terms s , t , and u such that $s \rightarrow u$ and $t \rightarrow u$ are equations in c together with a non-joinability certificate for (\mathcal{R}_u, s, t) .
5. Provide three terms s , t , and u such that $s \rightarrow t$ and $t \rightarrow u$ are equations in c together with a non-reachability certificate for (\mathcal{R}_u, s, u) .

Proof. Note that 3 is obvious and (1) only exists for tool-author convenience but is subsumed by the combination of (2) and 3. Moreover, (2) follows from the fact that $\mathbf{cs}(s_1, \dots, s_k)\sigma \not\rightarrow_{\mathcal{R}_u}^* \mathbf{cs}(t_1, \dots, t_k)\tau$ for all σ and τ , implies the existence of at least one $1 \leq i \leq k$ such that $s_i\sigma \not\rightarrow_{\mathcal{R}_u}^* t_i\tau$ for all σ and τ . Finally, for 4, whenever $s\sigma$ and $t\tau$ are not joinable for arbitrary σ and τ , the existence of μ and n such that $\mu, n \vdash s \rightarrow u, t \rightarrow u$ is impossible. □

Note that in (2) we check for non-reachability between left-hand sides and their corresponding right-hand sides, while in 4 we check for non-joinability between two left-hand sides. Thus, while in general non-joinability is more difficult to show than non-reachability, 4 is not directly subsumed by (2).


6. Certified Confluence of Conditional Rewriting

Lemma 6.7.39 (Non-reachability certificates). *Given (\mathcal{R}, s, t) consisting of a TRS \mathcal{R} and two terms s and t , \mathcal{R} -non-reachability of t from s can be certified in one of the following ways:* ✓


1. Indicate that $\text{tcap}(s)$ does not unify with t .
2. Indicate that generalized tcap shows non-reachability of s from t .
3. Provide a TRS \mathcal{R}' such that for each $\ell \rightarrow r \in \mathcal{R}$ there is $\ell' \rightarrow r' \in \mathcal{R}'$ and a substitution σ with $\ell = \ell'\sigma$ and $r = r'\sigma$, together with a non-reachability certificate for (\mathcal{R}', s, t) .
4. Provide a non-reachability certificate for (\mathcal{R}^{-1}, t, s) .
5. Use ordered completion to conclude non-reachability [152].
6. Make sure that \mathcal{R} is linear and growing and provide a finite signature \mathcal{F} and two constants \mathbf{a} and \square such that $\mathbf{a} \in \mathcal{F}$ but $\square \notin \mathcal{F}$, together with a tree automaton \mathcal{A} that is an overapproximation of $\text{anc}_{\mathcal{R}}(\mathcal{A}_{\Sigma(t)})$ and that satisfies $L(\mathcal{A}_{\Sigma(s)}) \cap L(\mathcal{A}) = \emptyset$.


Proof. If $\text{tcap}_{\mathcal{R}}(s) \not\sim t$, then 1 holds by Lemma 6.7.4. The same holds for 2 using Corollary 6.7.24. Further note that $\rightarrow_{\mathcal{R}} \subseteq \rightarrow_{\mathcal{R}'}$ and thus 3 is immediate. Moreover, 4 is obvious and 5 is shown by Sternagel and Winkler [152], leaving us with 6. From a certification perspective this is the most interesting case. To begin with, there are two reasons why we do not want to repeat the full construction of anc inside CeTA . Firstly, we would unnecessarily repeat an operation with at least exponential worst-case complexity. Secondly, a fully-verified executable algorithm is not even part of our formalization, instead we heavily rely on inductive definitions. While turning the existing inductive definitions into executable recursive functions would definitely be possible, we stress that this is not necessary. In CeTA we check that \mathcal{A} is an overapproximation of $\text{anc}_{\mathcal{R}}(\mathcal{A}_{\Sigma(t)})$ as follows: firstly, we ensure that \mathcal{A} does not contain epsilon transitions, that $[t]$ is included in the final states of \mathcal{A} , and that Δ_t as well as the matching rules with respect to the signature \mathcal{F} are part of the transitions of \mathcal{A} ; secondly, we check that \mathcal{A} is closed with respect to inference rule (\dagger) . Since $\mathcal{A}_{\Sigma(s)}$ is an overapproximation of $\Sigma(s)$ and by the required conditions together with Theorem 6.7.33, $L(\mathcal{A})$ overapproximates $[\rightarrow_{\mathcal{R}}^*](\Sigma(t))$, we can conclude $\Sigma(s) \cap [\rightarrow_{\mathcal{R}}^*](\Sigma(t)) = \emptyset$. Thus there are no *ground* substitutions σ and τ such that $s\sigma, t\tau \in \mathcal{T}(\mathcal{F})$ and $s\sigma \rightarrow_{\mathcal{R}}^* t\tau$. In order to conclude that the same holds true for *arbitrary* substitutions (not necessarily restricted to \mathcal{F}), we rely on an earlier result [137] that implies that whenever $s\sigma \rightarrow_{\mathcal{R}}^* t\tau$ for arbitrary σ and τ and $s, t \in \mathcal{T}(\mathcal{F}, \text{Var})$ then there are σ' and τ' such that $s\sigma', t\tau' \in \mathcal{T}(\mathcal{F})$ and $s\sigma' \rightarrow_{\mathcal{R}}^* t\tau'$. □

Note that 3 allows us to certify the linear growing approximation of a TRS without actually having to formalize it in Isabelle/HOL. More specifically, whenever \mathcal{R}' is the result of applying the linear growing approximation to \mathcal{R} , then the corresponding certificate will pass 3 and \mathcal{R}' will be linear and growing in the check for 6; otherwise 6 will fail.


Lemma 6.7.40 (Non-joinability certificates). *Given (\mathcal{R}, s, t) consisting of a TRS \mathcal{R} and two terms s and t , \mathcal{R} -non-joinability of s and t can be certified in one of the following ways:* 

1. Indicate that $\text{tcap}(s)$ does not unify with $\text{tcap}(t)$.
2. If at least one of the terms, say t , is a ground \mathcal{R} -normal form provide a non-reachability certificate for (\mathcal{R}, s, t) .

Proof. We prove (1) by Lemma 6.7.5 and (2) by Lemma 6.7.4 since non-joinability reduces to non-reachability when one of the terms is an \mathcal{R} -normal form. 

Lemma 6.7.41 (Ao-infeasibility certificates). *Given the triple (\mathcal{R}, c_1, c_2) consisting of a CTRS \mathcal{R} fulfilling all syntactic requirements of Theorem 6.4.11 and two lists of conditions c_1 and c_2 , infeasibility with respect to almost orthogonality can be certified in one of the following ways:* 

1. Provide an infeasibility certificate for (\mathcal{R}, c) where c is the concatenation of c_1 and c_2 .
2. Provide three terms s, t and u such that $s \rightarrow t$ is an equation in c_1 and $s \rightarrow u$ an equation in c_2 , together with a non-joinability certificate for (\mathcal{R}_u, t, u) .

Proof. While 1 follows from Lemma 6.7.38, in 2 we make use of the level-commutation assumption of Definition 6.4.9 to deduce non-meetability of the terms t and u from non-joinability of t and u . 

The formalization of the methods described in this section can be found in the following IsaFoR theory files:

thys/Rewriting/	thys/Tree_Automata/
Tcap.thy	Exact_Tree_Automata_Completion.thy
Ground_Context.thy	Exact_Tree_Automata_Completion_Impl.thy
thys/Nonreachability/	thys/Conditional_Rewriting/
Gtcap.thy	Infeasibility.thy
Gtcap_Impl.thy	Level_Confluence_Impl.thy
Nonreachability.thy	

In the next section we will look at two supporting methods that facilitate the techniques described in this and the previous sections.

6.8. Supporting Methods

Sometimes directly using the methods for (non-)confluence that are described in earlier sections is not possible. But there are sound methods that can “simplify” a given CTRS and make it amenable for them. In this section we will see two such methods.

6. Certified Confluence of Conditional Rewriting

The first one is about infeasibility. Already in Section 6.7 we have seen that infeasibility of CCPs can be expedient in analyzing confluence of CTRSs. Here we show that also infeasibility of conditions of rewrite rules can be beneficial for confluence analysis.

The second method can “reshape” certain conditional rewrite rules in such a way that other methods become more applicable.

6.8.1. Infeasible Rule Removal

If the conditions of a conditional rewrite rule are infeasible we can just remove this rule from the CTRS without changing the rewrite relation because the rule could never be fired anyway.

Definition 6.8.1 (Infeasible rule). *A conditional rewrite rule $\ell \rightarrow r \Leftarrow c$ is called infeasible if c is infeasible.*

You might ask, why anyone would add an infeasible rule to a CTRS in the first place? On the one hand, it might be a bug by a programmer. On the other hand, there are transformations from programs to CTRSs that occasionally result in infeasible rules. In fact, when looking into the Cops database, we find that from 111 DCTRSs a stunning 11 systems contain rules that we can show infeasible. And that is just using fast and easy checks so there might be more where infeasibility of the conditions is not so obvious. All 11 DCTRSs with infeasible rules are from the literature. For instance the example below is due to Bergstra and Klop.

Example 6.8.2. *Remember the DCTRS \mathcal{R} from Example 6.6.4 consisting of the following four rules*

$$p(q(x)) \rightarrow p(r(x)) \quad q(h(x)) \rightarrow r(x) \quad r(x) \rightarrow r(h(x)) \Leftarrow s(x) \rightarrow 0 \quad s(x) \rightarrow 1$$

The right-hand side 0 of the single condition of the only conditional rule of \mathcal{R} is an \mathcal{R} -normal form. The left-hand side $s(x)$ does only rewrite to the different \mathcal{R} -normal form 1. Hence the condition is infeasible and we can just remove the rule, because it does not affect the rewrite relation of \mathcal{R} in any way.


In principle we can use all the methods from Section 6.7 to remove infeasible rules before we employ any of the actual (non-)confluence checks. In the following example we utilize `anc` (see Section 6.7.4).

Example 6.8.3. *Consider the CTRS \mathcal{R} consisting of the two rules*

$$h(x) \rightarrow a \quad g(x) \rightarrow a \Leftarrow h(x) \rightarrow b$$


The condition of the only conditional rewrite rule is infeasible because $h(x)$ only rewrites to a and not to b . Unification fails to show that because $tcap(h(x)) = y \sim b = tcap(b)$. Fortunately we have $h \not\vdash_{rs} b$ and hence the condition $h(x) \rightarrow b$ is infeasible by Lemma 6.7.19. Or we can use exact tree automata completion. The underlying TRS \mathcal{R}_u is linear and growing and hence we can construct the tree automata $\mathcal{A}_{\Sigma(h(x))}$ and $anc_{\mathcal{R}_u}(\mathcal{A}_{\Sigma(b)})$. The language of the intersection automaton is empty and the condition $h(x) \rightarrow b$ is infeasible by Lemma 6.7.34.

Another useful result, due to Sternagel and Yamada [153, Theorem 3], that we formalized, is that we may ignore those rules of a CTRS that we are trying to show infeasible during our investigation:¹¹

Theorem 6.8.4. *A set of rules \mathcal{S} is infeasible with respect to a CTRS \mathcal{R} iff it is infeasible with respect to $\mathcal{R} \setminus \mathcal{S}$.* 

6.8.2. Inlining of Conditions

In this section we look at another simple method that is inspired by inlining of let-constructs and where-expressions in compilers. We give a transformation on CTRSs which is often helpful in practice.


Definition 6.8.5 (Inlining of Conditions ). *Given a conditional rewrite rule*

$$\rho: \ell \rightarrow r \Leftarrow s_1 \rightarrow t_1, \dots, s_k \rightarrow t_k$$

and an index $1 \leq i \leq k$ such that $t_i = x$ for some variable x , let $\text{inl}_i(\rho)$ denote the rule resulting from inlining the i -th condition of ρ , that is,

$$\ell \rightarrow r\sigma \Leftarrow s_1\sigma \rightarrow t_1, \dots, s_{i-1}\sigma \rightarrow t_{i-1}, s_{i+1}\sigma \rightarrow t_{i+1}, \dots, s_k\sigma \rightarrow t_k$$

with $\sigma = \{x \mapsto s_i\}$.

Lemma 6.8.6. *Let $\rho \in \mathcal{R}$ and $s \rightarrow x$ be the i -th condition of ρ . Whenever we have $x \notin \mathcal{V}(\ell, s, t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_k)$, then for $\mathcal{R}' = (\mathcal{R} \setminus \{\rho\}) \cup \{\text{inl}_i(\rho)\}$ the relations $\rightarrow_{\mathcal{R}}^*$ and $\rightarrow_{\mathcal{R}'}^*$ coincide.* 

Proof. We show $\rightarrow_{\mathcal{R},n} \subseteq \rightarrow_{\mathcal{R}',n}^*$ and $\rightarrow_{\mathcal{R}',n} \subseteq \rightarrow_{\mathcal{R},n}$ by induction on the level n . For $n = 0$ the result is immediate. Consider a step $s = C[\ell\sigma] \rightarrow_{\mathcal{R},n+1} C[r\sigma] = t$ employing rule ρ (for the other rules of \mathcal{R} the result is trivial). Thus, $u\sigma \rightarrow_{\mathcal{R},n}^* v\sigma$ for all $u \rightarrow v \in c$. In particular $s\sigma \rightarrow_{\mathcal{R},n}^* x\sigma$. Thus, using the IH, for each condition $u \rightarrow v$ of $\text{inl}_i(\rho)$ we have $u\sigma = s_j\{x \mapsto s\}\sigma \rightarrow_{\mathcal{R}',n}^* s_j\sigma \rightarrow_{\mathcal{R}',n}^* t_j\sigma = v\sigma$ for some $1 \leq j \leq k$. Hence, $\ell\sigma \rightarrow_{\mathcal{R}',n+1} r\{x \mapsto s\}\sigma \rightarrow_{\mathcal{R}',n+1}^* r\sigma$ and thus $s \rightarrow_{\mathcal{R}',n+1}^* t$.

Now, consider a step $s = C[\ell\sigma] \rightarrow_{\mathcal{R}',n+1} C[r\{x \mapsto s\}\sigma]$ employing rule $\text{inl}_i(\rho)$. Together with the IH this implies that $u\sigma \rightarrow_{\mathcal{R},n}^* v\sigma$ for all conditions $u \rightarrow v$ in $\text{inl}_i(\rho)$. Let τ be a substitution such that $\tau(x) = s\sigma$ and $\tau(y) = \sigma(y)$ for all $y \neq x$. We have $s_i\tau = s\tau = x\tau = t_i\tau$ and $s_j\tau = s_j\{x \mapsto s\}\sigma \rightarrow_{\mathcal{R},n}^* t_j\sigma = t_j\tau$ for all $1 \leq j \leq k$ with $i \neq j$, since x occurs neither in s nor in the right-hand sides of conditions in $\text{inl}_i(\rho)$. Therefore, $u \rightarrow_{\mathcal{R},n}^* v$ for all $u \rightarrow v \in c$. In summary, we have $s = C[\ell\sigma] = C[\ell\tau] \rightarrow_{\mathcal{R},n+1} C[r\tau] = C[r\{x \mapsto s\}\sigma]$, concluding the proof. \square

We are not aware of any mention of this simple method in the literature, but found that in practice, exhaustive application of inlining increases the applicability of other methods like infeasibility via **tcap** and non-confluence via plain rewriting: for the former

¹¹This result is applied implicitly by **CeTA** whenever checking a certificate that claims to remove infeasible rules.

6. Certified Confluence of Conditional Rewriting

inlining yields more term structure, which may prevent `tcap` from replacing a subterm by a fresh variable and thus makes non-unifiability more likely; while for the latter inlining may yield CCPs without conditions and thereby make them amenable to non-joinability techniques for plain term rewriting [184] as witnessed by Cops #551.

Example 6.8.7 (Cops #551). *Consider the quasi-decreasing ADCTRS \mathcal{R} consisting of the following six rules:*

$$\text{min}(x : \text{nil}) \rightarrow x \quad (6.1) \quad x \leq 0 \rightarrow \text{false} \quad (6.4)$$

$$\text{min}(x : xs) \rightarrow x \Leftarrow \text{min}(xs) \rightarrow y, x \leq y \rightarrow \text{true} \quad (6.2) \quad 0 \leq s(y) \rightarrow \text{true} \quad (6.5)$$

$$\text{min}(x : xs) \rightarrow y \Leftarrow \text{min}(xs) \rightarrow y, x \leq y \rightarrow \text{false} \quad (6.3) \quad s(x) \leq s(y) \rightarrow x \leq y \quad (6.6)$$

\mathcal{R} has 6 CCPs, 3 modulo symmetry:

$$x \approx x \Leftarrow \text{min}(\text{nil}) \rightarrow y, x \leq y \rightarrow \text{true} \quad (6.1,6.2)$$

$$x \approx y \Leftarrow \text{min}(\text{nil}) \rightarrow y, x \leq y \rightarrow \text{false} \quad (6.1,6.3)$$

$$x \approx y \Leftarrow \text{min}(xs) \rightarrow z, x \leq z \rightarrow \text{true}, \text{min}(xs) \rightarrow y, x \leq y \rightarrow \text{false} \quad (6.2,6.3)$$

To conclude confluence of the system it remains to check its CCPs. The first one, (6.1,6.2), is trivially context-joinable because the left- and right-hand sides coincide. Unfortunately, the methods of **ConCon** are not able to handle either of the CCPs (6.1,6.3) and (6.2,6.3). So we are not able to conclude confluence of \mathcal{R} just yet. But Rules (6.2) and (6.3) of \mathcal{R} are both susceptible to inlining of conditions. For each of them, we may remove the first condition and replace y by $\text{min}(xs)$ resulting in

$$\text{min}(x : xs) \rightarrow x \Leftarrow x \leq \text{min}(xs) \approx \text{true} \quad (2')$$

$$\text{min}(x : xs) \rightarrow \text{min}(xs) \Leftarrow x \leq \text{min}(xs) \approx \text{false} \quad (3')$$

Now we actually arrive at the CTRS from Example 6.5.14 which is shown to be confluent in Section 6.5.

6.8.3. Certification and Implementation

Infeasible rule removal and inlining of conditions are both implemented in **ConCon** as a preprocessing step and are certifiable by **CeTA**.

For the certification of infeasible rule removal we can reuse machinery in **ConCon** and **CeTA** that is also used to show infeasibility of CCPs. The certificate just has to provide the infeasible rules together with the infeasibility proofs for their conditions. Most of the time CTRSs do not contain infeasible rules (see Section 6.9) and some of the available infeasibility checks are rather expensive (especially tree automata techniques). So in our implementation we only employ the *symbol transition graph* (see Section 6.7.2) method in this case because it is fast and lightweight.

To certify inlining **CeTA** requires **ConCon** to output the inlined CTRS \mathcal{R}' together with a list of pairs that in the first component contain all modified rules and in the second component the corresponding list of conditions that have been inlined in this rule.

Internally CeTA employs this information to reverse the inlining and finally checks if the result corresponds to the original input CTRS \mathcal{R} .

The formalization of the methods described in this section can be found in the following IsaFoR theory files:

thys/Conditional_Rewriting/ Infeasibility.thy Inline_Conditions.thy	thys/Conditional_Rewriting/ Inline_Conditions_Impl.thy
---	---

In the next section we present the results of our extensive experiments comparing the different (non-)confluence and infeasibility methods on the confluence problems database.

6.9. Experiments

In the previous sections we have established the theory underlying several confluence and infeasibility methods and gave an overview of our corresponding Isabelle/HOL formalization as part of IsaFoR.

In the following, we compare these methods experimentally in order to get empirical evidence on their strengths and weaknesses. To this end we implemented the methods supported by CeTA in the CTRS confluence tool ConCon.¹² The tool is implemented in Scala, an object-functional programming language that runs on the Java virtual machine.

ConCon also implements two methods that are not certifiable by CeTA, specifically a variant of Theorem 6.3.1 (see [48]) relying on the notion of weak left-linearity of a DCTRS [47] and the inductive symbol transition graph due to Sternagel and Yamada [153]. For more details on ConCon see [155].

All experiments have been carried out on a 64bit GNU/Linux machine with 12 Intel® Core™ i7-5930K processors clocked at 3.50GHz and 32GB of RAM. The kernel version is 4.9.0-9-amd64. The version of Java on this machine is 1.8.0.222. We had to increase the stack size used by ConCon to 20MB using the JVM flag ‘-Xss20M’ to prevent stack overflows caused by parsing deep terms like in Cops #313.

6.9.1. Comparing ConCon’s Confluence Methods

In this first part of our experiments we take a closer look at ConCon’s confluence methods in comparison to each other. We use the 149 oriented CTRSs from Cops¹³ version 1137, and set the timeout to one minute. The subfigures of Figure 6.6 compare the three confluence methods Theorem 6.5.5 (A), Theorem 6.3.2 (B), and Theorem 6.3.1 (C) for various settings of ConCon.

If we look at ConCon’s absolute power, employing the supporting methods from Section 6.8, infeasibility of CCPs from Section 6.7, and also uncertified methods (which are not part of this article) we get the numbers in the Venn diagram depicted in 6.6a.

¹²<http://cl-informatik.uibk.ac.at/software/concon>

¹³<http://cops.uibk.ac.at?q=1..1137+oriented+ctrs>

6. Certified Confluence of Conditional Rewriting

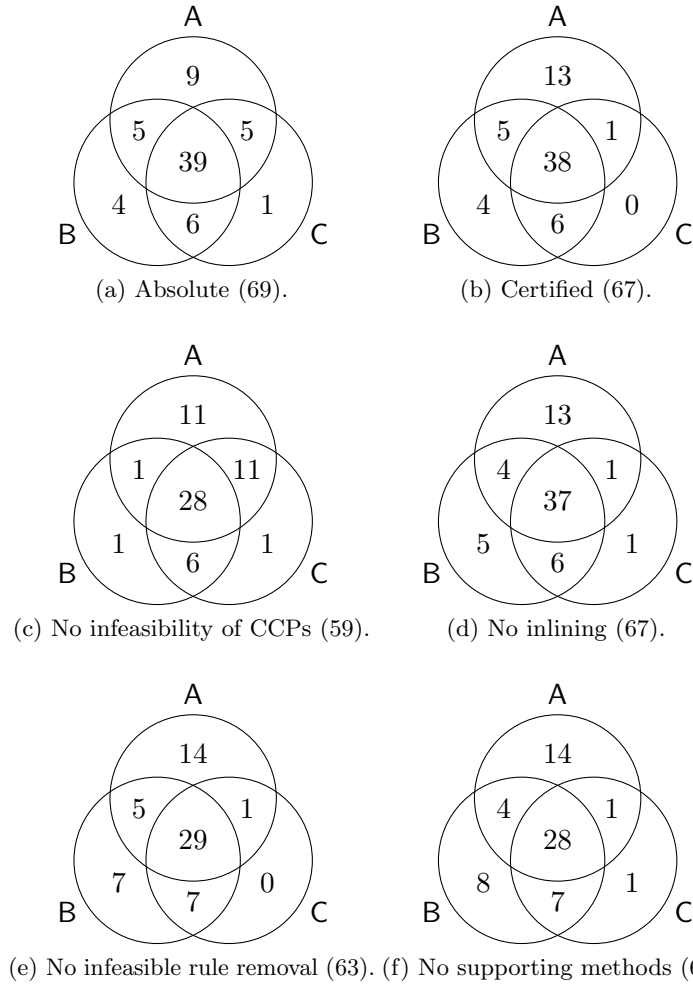


Figure 6.6.: Comparison of ConCon's confluence methods.

There is a total of 69 systems that are shown to be confluent. On its own method A is the strongest, succeeding on 58 CTRSs, closely followed by method B, which succeeds on 54 CTRSs, and finally method C, which can show confluence of 51 systems. Moreover, there are 9 CTRSs where only method A succeeds, 4 where only method B succeeds, and 1 where only method C succeeds.

The results for restricting to the methods that can be certified by *CeTA* can be found in 6.6b. In this case, the total is reduced by two systems ([#286](#) and [#793](#)) to 67 confluent systems when compared to 6.6a. More specifically, methods A (57 total) and B (53 total) lose one system each, and method C six systems (45 total). The system lost by methods A and B is Cops [#793](#). In both cases, it is lost because the certified infeasibility methods are not able to conclude that the first rule is infeasible. Concerning method C, Cops [#286](#), [#326](#), [#319](#), [#362](#), [#793](#), and [#806](#) are all weakly-left linear (and a variant

of Theorem 6.3.1 for weakly-left linear CTRSs succeeds if we use uncertified methods) but their unravelings are not left-linear. Hence our formalization of Theorem 6.3.1 is not applicable. Of these six Cops #286 is the system where in 6.6a only method C succeeds.

Next, in 6.6c, we see the confluence results when switching off all infeasibility checks for conditional critical pairs. The total number of systems that can be shown to be confluent is reduced by eight (#288, #292, #326, #340, #361, #494, #551, and #552) to 59 when compared to 6.6b. As expected—because it does not profit from infeasible CCPs—method C is not effected by this change. Method A loses six systems: #288, #292, #326, #340, #551, and #552. (While it does rely on infeasibility of CCPs, it also employs context-joinability and unfeasibility of CCPs (see Definition 6.5.11).) Method B is affected the most, losing 17 systems, because without infeasibility of CCPs method B is reduced to a purely syntactic check that relies on the absence of non-trivial critical pairs.

Switching off inlining (but still using infeasible rule removal) does not change the overall number of confluent systems (see 6.6d) compared to 6.6b. However, when looking at the specific systems that could be shown to be confluent, we observe that we actually gained #529 (which times out without inlining) but lost #551. Recall that inlining can give terms more structure, which is good to show non-reachability using *tcap*. On the one hand, for Cops #551 without inlining ConCon’s infeasibility methods are not able to handle all of the CCPs (as explained in Example 6.8.7). On the other hand, if we need tree automata completion to show infeasibility of a conditional critical pair more structure may lead to a larger tree automaton that we have to compute and hence to a timeout. Apparently this is what happens for Cops #529: while ConCon times out when inlining is used it succeeds when we switch it off. We also want to remark that although without inlining ConCon is able to produce a certificate for Cops #529, CeTA is actually not able to certify it because to do so it would have to compute 21^{11} state substitutions to check the given tree automaton and consequently we run out of memory. Besides, Cops #493 is now only proven confluent by method C because without inlining ConCon’s infeasibility checks do not succeed on two of its CCPs.

Now, if we switch off infeasible rule removal (but keep inlining) the overall number of confluent systems is reduced by four to 63 when compared to 6.6b (see 6.6e). The lost systems are #317, #792, #794, and #806. Here methods A (49 total) and B (37 total) lose eight systems each, while method C (48 total) loses five systems.

For sake of completeness we also include 6.6f which shows the results when switching off both inlining and infeasible rule removal. As expected the results are basically a combination of the previous two diagrams.

6.9.2. Comparing ConCon’s Non-Confluence Methods

In this section we take a closer look at ConCon’s non-confluence methods in comparison to each other. We use the same 149 oriented CTRSs as in the previous section and the same timeout of one minute. Results for the three non-confluence methods Lemma 6.6.7 (N1), Lemma 6.6.3 (N2), and Lemma 6.6.1 (N3) of ConCon are summarized in Figure 6.7.

When it comes to non-confluence ConCon is able to handle 44 systems when employing the supporting methods from Section 6.8. 6.7a compares the three non-confluence methods

6. Certified Confluence of Conditional Rewriting

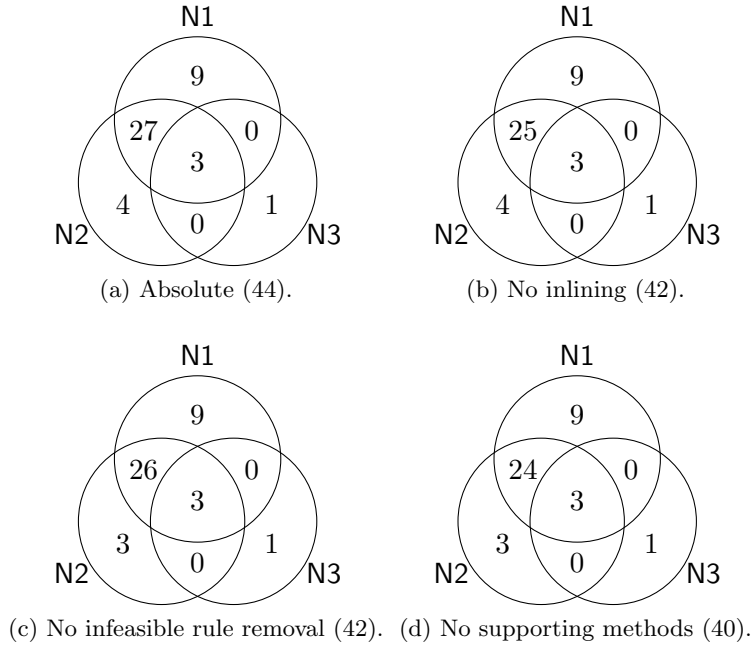


Figure 6.7.: Comparison of ConCon's non-confluence methods.

of ConCon. On its own method N1 is the strongest succeeding on 39 systems, followed by method N2 succeeding on 34 systems, and finally method N3 which is specifically targeted at 4-CTRSs and hence can only handle the 4 systems of this kind.

As shown in 6.7b when we switch off inlining method N2 loses 2 systems. Inlining transforms Cops #351 to an unconditional system and cannot be handled by ConCon without this method. Cops #353 can only be handled by method N1 without inlining. The total number of non-confluent systems is reduced to 41.

Similarly, as shown in 6.7c when switching off infeasible rule removal method N2 loses one system. This system, Cops #271, is reduced to a TRS when employing infeasible rule removal and cannot be handled by ConCon without this method.

Finally, the results for switching off both supporting methods are depicted in 6.7d. The total number of non-confluent systems is reduced to 40 because as explained earlier we lose Cops #351 and #271. Unlike in the case of the confluence methods, where inlining also causes ConCon to lose one system, in the non-confluence case it is exclusively beneficial.

6.9.3. Comparing ConCon's Infeasibility Methods

In this last section regarding our experiments we compare the five infeasibility methods of ConCon, that is, the `tcap` check (`tcap`; see Section 6.7.1, the generalized `tcap` check with symbol transition graph and decomposition of reachability problems (`gtcap`; see Section 6.7.2 and and Section 6.7.3), exploiting equalities in conditions (`exeq`; see Section 6.7.5), equational reasoning using MædMax [152] (`oc`; not part of this article), and finally, exact

	tcap	gtcap	exeq	oc	etac	total
infeasible	19 (114)	22 (111)	4 (129)	20 (113)	35 (98)	40 (93)
timeout	0	0	0	18	40	40

Table 6.1.: Infeasibility results on 133 oriented infeasibility problems from Cops.

tree automata completion (**etac**; see Section 6.7.4).

To this end we take the 133 oriented infeasibility problems from Cops¹⁴ and apply ConCon’s infeasibility methods separately with a timeout of 60 seconds. (As an aside: infeasibility problems are a part of Cops since the International Confluence Competition in 2019.) The results of this experiments are listed in Table 6.1 on page 131. The first row, labeled “infeasible,” lists the number of problems which could be shown to be infeasible for each method (the numbers in parentheses indicate how many problems are still open for that column). The row labeled “timeout” gives the number of problems for which a timeout occurred. On its own method **etac** is the strongest showing infeasibility of 35 problems. However, it also is very computation intensive and causes the most timeouts (40). There are seven problems where only method **etac** is able to show infeasibility. Next, method **gtcap** shows 22 problems infeasible and has no timeouts. Unsurprisingly, it subsumes method **tcap**. Method **oc** shows 20 problems infeasible and causes 18 timeouts. There are 4 problems where only method **oc** is able to show infeasibility. Method **exeq** shows four problems infeasible and does not cause any timeouts.

In summary, all methods together succeed on 40 problems and time out on 40.

While, at least on this benchmark, **oc** and **etac** together subsume all other methods, in practice it is still important to first employ fast methods like **exeq** and **gtcap**, and only then expensive methods like **oc** and **etac** in order to avoid timeouts. For removal of infeasible rules during confluence proofs (Section 6.8.1), fast methods are even more important, since only a small fraction of the overall time can be spared on checking infeasibility of rules.

This brings us to the final section of this article where we reflect on the findings of all previous sections and lay out some possible future work.

6.10. Conclusion

In the last few sections we presented an overview of the confluence and infeasibility results for CTRSs that we have formalized in **IsaFoR**. Through code generation these are now available in the certifier **CeTA**.

We first presented two of the three main confluence methods available in **CeTA**: the result that almost orthogonal (modulo infeasibility), right-stable, and extended properly oriented CTRSs are confluent in Section 6.4 and that quasi-decreasing, strongly deterministic CTRSs are confluent if all their conditional critical pairs are joinable in Section 6.5. Section

¹⁴<http://cops.uibk.ac.at?q=1..1137+infeasibility> but without **#1137** and **#1136** which are both semi-equational and not oriented.

6. Certified Confluence of Conditional Rewriting

6.6 presented some checks, most notably a method employing conditional narrowing, to find witnesses for non-confluence of CTRSs. The topic of Section 6.7 were several methods to check for non-reachability between terms. In our context these are used to get infeasibility of conditional critical pairs as well as conditional rules in order to make the methods of Sections 6.4 and 6.5, that both rely on critical pair analysis, more applicable. In detail the non-reachability checks are by unification, the symbol transition graph employing decomposition of reachability problems, exact tree automata completion, and the exploitation of certain equalities in the conditions. After that we touched on two supporting methods in Section 6.8. One that employs some of the results of Section 6.7 to get rid of infeasible rules and another that inlines certain conditions of rules. Finally, Section 6.9 presented our experiments on the confluence problems database. We first compared ConCon’s three confluence methods to each other. Following that, we presented experimental results on ConCon’s non-confluence methods. Finally, we investigated ConCon’s infeasibility methods in some detail.

6.10.1. Formalization and Implementation

A lot of work went into the formalization of the presented results. Here is a rough impression of the involved effort: our formalization comprises 92 definitions, 37 recursive functions, and 518 lemmas with proofs, on approximately 10,000 lines of Isabelle code (in addition to everything that we could reuse from the IsaFoR library).

To give some additional measure on how much work went into the formalization we take a look at the *de Bruijn factor* of some of its parts. The de Bruijn factor has been defined by Freek Wiedijk¹⁵ to be the quotient of the size of a formalization of a mathematical text and the size of its informal original. Because our formalization depends on a lot of prior results from IsaFoR the scope of a specific result is not so easy to define and hence somewhat arbitrary. Nevertheless, when comparing the textual description in this article to our formalization in IsaFoR (without the additional setup for code generation and CeTA’s parser) we get the following (approximate) numbers: the results of Section 6.7.4 have a de Bruijn factor of 9.8, Theorem 6.4.11 has a de Bruijn factor of 4.2, and finally the de Bruijn factor of the results in Section 6.5 is 4.5.

There are some points of notice: the textual description of all of the above mentioned proofs to some extent contain diagrams to convince the reader of certain steps. Such diagrams are notoriously hard to formalize. Further, the results in Section 6.7.4 have been the second author’s first larger IsaFoR-development and he was still trying to get to grips with Isabelle/HOL. So he probably did not exploit Isabelle/HOL’s automatic methods to their full potential, for that reason these proofs are quite verbose, which explains the much higher de Bruijn factor (in comparison to the later developments).

But also the work put in the automatic tool ConCon, although somewhat paling in comparison to the formalization, was significant. ConCon 1.9.1 consists of approximately 14,000 lines of Scala code and has been very successful in the confluence competition. At the time of writing it won the CPF-CTRS category for confluence of CTRSs with proof

¹⁵<http://www.cs.ru.nl/~freek/factor>

certificates four times in a row and is (after four years) still the only tool for CTRSs to provide certifiable output. Moreover, ConCon can decide confluence of roughly 75% of the oriented CTRSs in Cops and certify approximately 98% of these with the help of CēTA.

6.10.2. Future Work

In our opinion the presented work satisfactorily discharges the goal to produce a reliable and automatic tool to check confluence of conditional term rewrite systems. Nevertheless, there are some open issues:

Infeasibility. Currently 36 out of the 149 oriented CTRSs in Cops cannot be solved by ConCon 1.9.1. One of those is Cops #327 for computing the greatest common divisor of two natural numbers:

$$\begin{array}{lll}
 \text{gcd}(x, x) \rightarrow x & x \leq 0 \rightarrow \text{false} & 0 - s(y) \rightarrow 0 \\
 \text{gcd}(s(x), 0) \rightarrow s(x) & 0 \leq s(y) \rightarrow \text{true} & x - 0 \rightarrow x \\
 \text{gcd}(0, s(y)) \rightarrow s(y) & s(x) \leq s(y) \rightarrow x \leq y & s(x) - s(y) \rightarrow x - y \\
 \text{gcd}(s(x), s(y)) \rightarrow \text{gcd}(x - y, s(y)) \Leftarrow y \leq x \rightarrow \text{true} & & \\
 \text{gcd}(s(x), s(y)) \rightarrow \text{gcd}(s(x), y - x) \Leftarrow x \leq y \rightarrow \text{true} & &
 \end{array}$$

Because the system is not left-linear Theorem 6.4.11 is not applicable. Its unraveling is not left-linear, so Theorem 6.3.1 is also not applicable. But Cops #327 is a quasi-decreasing ADCTRS, making Theorem 6.5.13 applicable in principle. It only remains to show joinability (or infeasibility for that matter) of its CCPs. The CTRS has three CCPs (modulo symmetry) of which we show two (because the conditions of the third are similar):

$$\begin{array}{l}
 \text{gcd}(s(x), y - x) \approx \text{gcd}(x - y, s(y)) \Leftarrow y \leq x \rightarrow \text{true}, x \leq y \rightarrow \text{true} \\
 \text{gcd}(x - x, s(x)) \approx s(x) \Leftarrow x \leq x \rightarrow \text{true}
 \end{array}$$

Under (the reasonable assumption) that \leq is supposed to encode “strictly less than,” these CCPs are obviously infeasible (y cannot be strictly smaller and strictly greater than x at the same time for the first CCP and x cannot be strictly smaller than itself for the second one). Unfortunately, this cannot be shown by the methods presented in Section 6.7. By using \mathcal{R}_u (for example when approximating non-reachability) we open the door for inconsistencies:

$$s(0) \xleftarrow{*} \text{gcd}(s(0), s(0)) \xleftarrow{*} \text{gcd}(s(s(0)), s(0)) \xrightarrow{*} \text{gcd}(0, s(s(0))) \xrightarrow{*} s(s(0))$$

and thus $\text{gcd}(s(s(0)), s(0)) \leq \text{gcd}(s(s(0)), s(0)) \rightarrow^* s(0) \leq s(s(0)) \rightarrow^* \text{true}$. Consequently, we may substitute $\text{gcd}(s(s(0)), s(0))$ for both x and y to satisfy the conditions of the CCPs.

It seems that there is still a lot of room for improvement with regard to infeasibility checking. So far all of the infeasibility methods employed by ConCon are unconditional

6. Certified Confluence of Conditional Rewriting

and only approximate the conditional rewrite relation. Maybe we could overcome this problem by employing some external tool that takes conditions into account. Since our tree automata techniques are quite successful in showing infeasibility, the tree automata completion tool *Timbuk* [37, 42], which implements conditional tree automata completion, would be a natural candidate.

Another method that could be extended to allow conditions is the symbol transition graph. This was investigated further by Sternagel and Yamada [153], but was not yet formalized as part of *IsaFoR*.

In general, it would make sense to design, implement and (hopefully) verify a dedicated reachability tool for (conditional) term rewriting. The decomposition of reachability problems from Section 6.7.3 gives a nice modular framework for that. Not only *ConCon* would benefit from such a tool.

Other Flavors. For the time being *CeTA* only supports oriented CTRSs. In principle it should be possible to extend most of the presented methods for join CTRSs. Moreover, when employing conditional linearization [30] in order to show the unique normal form property with respect to conversions for non-left-linear TRSs, methods for semi-equational CTRSs are needed and could be formalized in future releases.

Certification. The main reason for the gap of two systems between the problems that *ConCon* can show confluent and the ones that *CeTA* can certify is due to the inductive symbol transition graph by Sternagel and Yamada [153].

6.10.3. Related Work

With the sole exception of the formalized unraveling result due to Winkler and Thiemann [177], we are not aware of any other attempt to formally verify techniques for proving confluence of conditional term rewrite systems.

Concerning automated tools for proving CTRS confluence, *ConCon* is in good company: *ACP* [4], *CO3*¹⁶ [104], *CoScart*¹⁷ [46]. Among these tools *ConCon* is the only one that produces proof certificates.

¹⁶<https://www.trs.css.i.nagoya-u.ac.jp/co3/>

¹⁷<https://github.com/searles/RewriteTool/>

7. Abstract Completion, Formalized

Publication Details

Nao Hirokawa, Aart Middeldorp, Christian Sternagel, and Sarah Winkler. Abstract Completion, Formalized. *Logical Methods in Computer Science*, 15(3):19:1–19:42, 2019 [doi:10.23638/LMCS-15\(3:19\)2019](https://doi.org/10.23638/LMCS-15(3:19)2019)

Nao Hirokawa, Aart Middeldorp, Christian Sternagel, and Sarah Winkler. Infinite Runs in Abstract Completion. In *Proceedings of the 2nd International Conference on Formal Structures in Computation and Deduction*, volume 84 of *Leibniz International Proceedings in Informatics*, pages 19:1–19:16, Schloss Dagstuhl, 2017 [doi:10.4230/LIPIcs.FSCD.2017.19](https://doi.org/10.4230/LIPIcs.FSCD.2017.19)

Nao Hirokawa, Aart Middeldorp, Christian Sternagel. A New and Formalized Proof of Abstract Completion. In *Proceedings of the 5th International Conference on Interactive Theorem Proving*, volume 8558 of *Lecture Notes in Computer Science*, pages 292–307, Springer, 2014 [doi:10.1007/978-3-319-08970-6_19](https://doi.org/10.1007/978-3-319-08970-6_19)

Abstract

Completion is one of the most studied techniques in term rewriting and fundamental to automated reasoning with equalities. In this paper we present new correctness proofs of abstract completion, both for finite and infinite runs. For the special case of ground completion we present a new proof based on random descent. We moreover extend the results to ordered completion, an important extension of completion that aims to produce ground-complete presentations of the initial equations. We present new proofs concerning the completeness of ordered completion for two settings. Moreover, we revisit and extend results of Métivier concerning canonicity of rewrite systems. All proofs presented in the paper have been formalized in Isabelle/HOL.

7.1. Introduction

Reasoning with equalities is pervasive in computer science and mathematics, and has consequently been one of the main research areas of automated deduction. Indeed completion as introduced by Knuth and Bendix [71] has evolved into a fundamental technique whose ideas appear throughout automated reasoning whenever equalities are present. Many variants of the original calculus have since been proposed.

Bachmair, Dershowitz, and Hsiang [13] recast completion procedures as inference systems. This style of presentation, *abstract completion*, has become the standard to

7. Abstract Completion, Formalized

describe completion procedures and *proof orders* the accompanying tool to establish correctness [10, 13, 14], that is, that under certain conditions, exhaustive application of the inference rules results in a terminating and confluent rewrite system whose equational theory is equivalent to the initial set of equations.

In this paper we present new, modular correctness proofs, not relying on proof orders, for five abstract completion systems presented in the literature. Here, we use *modular* in the following sense: Proof orders have to be powerful (and thus complex) enough to cover all intermediate results (that is, proof orders are a *global* method), while for our new proofs, we *locally* apply well-founded induction with an order that is just strong enough for the current intermediate result. All proofs are fully formalized in Isabelle/HOL. First, we consider finite (KB_f) and infinite (KB_i) runs of classical Knuth-Bendix completion [71]. These two settings demand different proofs since in the latter case the inference system exhibits a stronger side condition. While our correctness proof for KB_f relies on a new notion we dub *peak decreasingness*, for the case of KB_i we employ a simpler version of this criterion called *source decreasingness*. To enhance applicability by covering efficient implementations, our proofs support the critical pair criterion known as primality [67].

The relevance of infinite runs is illustrated by the following example.

Example 7.1.1. Consider the set of equations $\mathcal{E} = \{\text{aba} \approx \text{bab}\}$ of the three-strand positive braid monoid. Kapur and Narendran [66] proved that \mathcal{E} admits no finite complete presentation. However, taking the Knuth-Bendix order [71] with **a** and **b** of weight 1 and $\text{a} > \text{b}$ in the precedence, completion produces in the limit the following infinite complete presentation of \mathcal{E}

$$\{\text{aba} \rightarrow \text{bab}\} \cup \{\text{ab}^n \text{ab} \rightarrow \text{babba}^{n-1} \mid n \geq 2\}$$

which can be used to decide the validity problem for \mathcal{E} .¹

Completion procedures, when successful, produce a complete system. Natural questions include whether such systems are unique and whether all complete systems for a given set of equations can be obtained by completion. For canonical systems, which are complete systems that satisfy an additional normalization requirement, Métivier [87] obtained interesting results. In this paper we revisit and extend his work.

A special case of KB_f that is known to be decidable is the completion of ground systems [128]. We present new correctness and completeness proofs for the corresponding inference system KB_g , based on the recent notion of random descent [109].

On a given set of input equalities, Knuth-Bendix completion can behave in three different ways: it may (1) succeed to compute a complete system in finitely many steps, (2) fail due to unorientable equalities, or (3) continuously compute approximations of a complete system without ever terminating. As a remedy to problem (2), ordered completion was developed by Bachmair, Dershowitz, and Plaisted [14]. Ordered completion never fails and can produce a ground-complete system in the limit. Although the price to be paid is that the resulting system is in general only complete on ground terms, this is actually

¹Burckel [24] constructed a complete rewrite system consisting of four rules with an additional symbol, which is no longer a complete presentation of \mathcal{E} but can be also used to decide the validity problem for \mathcal{E} .

Table 7.1.: Roadmap.

	KB _f	KB _g	KB _i	KB _o	KB _l
inference system	7.3.1	7.5.1	7.6.2	7.7.2	7.8.13
fairness	7.3.5	–	7.6.3	7.7.14	7.8.16
correctness	7.3.8	7.5.5	7.6.12	7.7.17 7.7.16	7.8.17
completeness	–	7.5.12	–	7.8.10	7.8.23

sufficient for many applications in theorem proving. Refutational theorem proving [14] owes its semi-decidability to the unfailing nature of ordered completion. Again employing peak decreasingness, we obtain a new correctness proof of ordered completion (KB_o). Next, we turn to completeness results for ordered completion, that is, to sufficient criteria for an ordered completion procedure to produce a complete system. We first reprove the case of a total reduction order, which assumes a slightly stronger notion of simplifiedness than the original result [14] though. Then we consider the completeness result for linear completion (KB_l) due to Devie [34].

For easy reference, Table 7.1 provides pointers to the main definitions and results we present in this paper.


The remainder of this paper is organized as follows. We present required preliminaries in Section 7.2, followed by the abstract confluence criteria of peak and source decreasingness, as well as a fairly detailed analysis of critical pairs. In Section 7.3 we recall the inference rules for (abstract) Knuth-Bendix completion and present our formalized correctness proof for finite runs. In Section 7.4 we present our results on canonical systems and normalization equivalence. We discuss ground completion in Section 7.5. Infinite runs are the subject of Section 7.6 and in Section 7.7 we extend our correctness results to ordered completion. Completeness of ordered completion is the topic of Section 7.8. We conclude in Section 7.9 with a few suggestions for future research.

Our formalizations are part of the *Isabelle Formalization of Rewriting IsaFoR* [165]² version 2.37. Below we list the relevant Isabelle theory files grouped by their subdirectories inside IsaFoR:

thys/Abstract_Completion/ Abstract_Completion.thy Completion_Fairness.thy CP.thy Ground_Completion.thy Peak_Decreasingness.thy Prime_Critical_Pairs.thy	thys/Confluence_and_Completion/ Ordered_Completion.thy thys/Normalization_Equivalence/ Encompassment.thy Normalization_Equivalence.thy
---	--

²<http://cl-informatik.uibk.ac.at/isafor>

7. Abstract Completion, Formalized

In the remainder we provide hyperlinks (marked by ) to an HTML rendering of our formalization. Moreover, whenever we say that a proof is “formalized,” what we mean is that it is “formalized in Isabelle/HOL.” And when we “present a formalized proof,” we give a textual representation of a formalized proof.

This paper and the accompanying formalization are substantially extended and revised versions of some of our previous work we published in the ITP [58] and FSCD [59] conferences. The former presented a new correctness proof for finite runs of Knuth-Bendix completion. Its modular design separates concerns rather than relying on a single proof order, thus rendering it more formalization friendly. In revised form, these results are included in Section 7.3. The FSCD contribution extended this novel proof approach to both infinite runs and ordered completion (see Sections 7.6 and 7.7). It moreover incorporated canonicity results (Section 7.4). In addition to these results we present new and formalized proofs of correctness and completeness of ground completion (Section 7.5), as well as completeness of ordered completion for two different cases (Section 7.8). At the end of each section, we remark on the novelty of the respective results and their proofs.

7.2. Preliminaries

We assume familiarity with the basic notions of abstract rewrite systems, term rewrite systems, and completion [9, 10], but nevertheless shortly recapitulate terminology and notation that we use in the remainder.

7.2.1. Rewrite Systems

For an arbitrary binary relation \rightarrow_α , we write $\alpha \leftarrow$, $\xleftrightarrow{\alpha}$, \rightarrow_α^- , \rightarrow_α^+ , and \rightarrow_α^* to denote its *inverse*, its *symmetric closure*, its *reflexive closure*, its *transitive closure*, and its *reflexive transitive closure*, respectively. The reflexive, transitive, and symmetric closure $\xleftrightarrow{\alpha}^*$ of \rightarrow_α is called *conversion*, and a sequence of the form $c_0 \xleftrightarrow{\alpha} c_1 \xleftrightarrow{\alpha} \cdots \xleftrightarrow{\alpha} c_n$ is referred to as a conversion between c_0 and c_n (of length n). For a binary relation R without arrow notation, we also write R^{-1} for its *inverse* and R^\pm for its *symmetric closure* $R \cup R^{-1}$. We further use \downarrow_α as abbreviation for the *joinability relation* $\rightarrow_\alpha^* \cdot \xleftrightarrow{\alpha}^*$, where from here on \cdot denotes relation composition. If $a \rightarrow_\alpha b$ for no b then we say that a is a (\rightarrow_α) -*normal form*. The set of all normal forms of a given relation \rightarrow_α is denoted by $\text{NF}(\rightarrow_\alpha)$. By $a \rightarrow_\alpha^! b$ we abbreviate $a \rightarrow_\alpha^* b \wedge b \in \text{NF}(\rightarrow_\alpha)$ and we call b a *normal form of a* . Given two binary relations \rightarrow_α and \rightarrow_β , we use $\rightarrow_\alpha / \rightarrow_\beta$ as shorthand for the *relative rewrite relation* $\rightarrow_\beta^* \cdot \rightarrow_\alpha \cdot \rightarrow_\beta^*$. An *abstract rewrite system* (ARS for short) \mathcal{A} is a set A , the carrier, equipped with a binary relation \rightarrow . Sometimes we partition the binary relation into parts according to a set I of indices (or labels). Then we write $\mathcal{A} = \langle A, \{\rightarrow_\alpha\}_{\alpha \in I} \rangle$ where we denote the part of the relation with label α by \rightarrow_α , that is, $\rightarrow = \bigcup \{\rightarrow_\alpha \mid \alpha \in I\}$.

We assume a given signature \mathcal{F} and a set of variables \mathcal{V} . The set of terms built up from \mathcal{F} and \mathcal{V} is denoted by $\mathcal{T}(\mathcal{F}, \mathcal{V})$, while $\mathcal{T}(\mathcal{F})$ denotes the set of ground terms. *Positions* are strings of positive integers which are used to address subterms. The set of positions

in a term t is denoted by $\text{Pos}(t)$. The subset consisting of the positions addressing function symbols in t is denoted by $\text{Pos}_{\mathcal{F}}(t)$ whereas $\text{Pos}_{\mathcal{V}}(t) = \text{Pos}(t) - \text{Pos}_{\mathcal{F}}(t)$ is the set of variable positions in t . We write $p \leq q$ if p is a prefix of q and $p \parallel q$ if neither $p \leq q$ nor $q \leq p$. If $p \leq q$ then the unique position r such that $pr = q$ is denoted by $q \setminus p$. A *substitution* is a mapping σ from variables to terms such that its domain $\{x \in \mathcal{V} \mid \sigma(x) \neq x\}$ is finite. Applying a substitution σ to a term t is written $t\sigma$. A variable substitution is a substitution from \mathcal{V} to \mathcal{V} and a *renaming* is a bijective variable substitution. A term s is a *variant* of a term t if $s = t\sigma$ for some renaming σ . A pair of terms (s, t) is sometimes considered an *equation*, then we write $s \approx t$, and sometimes a *(rewrite) rule*, then we write $s \rightarrow t$. In the latter case we assume the *variable condition*, that is, that the left-hand side s is not a variable and that variables of the right-hand side t are all contained in s . A set \mathcal{E} of equations is called an *equational system* (ES for short) and a set \mathcal{R} of rules a *term rewrite system* (TRS for short). Sets of pairs of terms \mathcal{E} induce a *rewrite relation* $\rightarrow_{\mathcal{E}}$ by closing their components under contexts and substitutions. A rewrite step $s \rightarrow_{\mathcal{E}} t$ at a position $p \in \text{Pos}(s)$ is called *innermost* and denoted by $s \xrightarrow{i}_{\mathcal{E}} t$ if no proper subterm of $s|_p$ is reducible in \mathcal{E} . The *equational theory* induced by \mathcal{E} consists of all pairs of terms s and t such that $s \leftrightarrow_{\mathcal{E}}^* t$. If $\ell \rightarrow r$ is a rewrite rule and σ is a renaming then the rewrite rule $\ell\sigma \rightarrow r\sigma$ is a *variant* of $\ell \rightarrow r$. A TRS is said to be *variant-free* if it does not contain rewrite rules that are variants of each other.

Two terms s and t are called *literally similar*, written $s \doteq t$, if $s\sigma = t$ and $s = t\tau$ for some substitutions σ and τ . Two TRSs \mathcal{R}_1 and \mathcal{R}_2 are called *literally similar*, denoted by $\mathcal{R}_1 \doteq \mathcal{R}_2$, if every rewrite rule in \mathcal{R}_1 has a variant in \mathcal{R}_2 and vice versa. The following result is folklore; we formalized the non-trivial proof.

Lemma 7.2.1. *Two terms s and t are variants of each other if and only if $s \doteq t$.* ✓

We say that s *encompasses* t , written $s \trianglerighteq t$, whenever $s = C[t\sigma]$ for some context C and substitution σ . *Proper encompassment* is defined by $\triangleright = \trianglerighteq \setminus \trianglelefteq$ and known to be well-founded. The identity $\trianglerighteq = \triangleright \cup \trianglelefteq$ is well-known. For a well-founded order $>$, we write $>_{\text{mul}}$ to denote its *multiset extension* and $>_{\text{lex}}$ to denote its *lexicographic extension* as defined by Baader and Nipkow [9].

A TRS \mathcal{R} is *terminating* if $\rightarrow_{\mathcal{R}}$ is well-founded, and *weakly normalizing* if every term has a normal form. It is *(ground-)confluent* if $s \xrightarrow{*}_{\mathcal{R}} \cdot \rightarrow_{\mathcal{R}}^* t$ implies $s \rightarrow_{\mathcal{R}}^* \cdot \xrightarrow{*}_{\mathcal{R}} t$ for all (ground) terms s and t . It is *(ground-)complete* if it is terminating and (ground) confluent. We say that \mathcal{R} is a *complete presentation* of an ES \mathcal{E} if \mathcal{R} is complete and $\leftrightarrow_{\mathcal{R}}^* = \leftrightarrow_{\mathcal{E}}^*$. A TRS \mathcal{R} is *left-reduced* if $\ell \in \text{NF}(\mathcal{R} \setminus \{\ell \rightarrow r\})$ for every rewrite rule $\ell \rightarrow r$ in \mathcal{R} , and *right-reduced* if $r \in \text{NF}(\mathcal{R})$ for every rewrite rule $\ell \rightarrow r$ in \mathcal{R} . A *reduced* TRS is left- and right-reduced. A reduced complete TRS is called *canonical*.

We make use of the following result due to Bachmair and Dershowitz [11], where *quasi-commutation* of R over S means that the inclusion $S \cdot R \subseteq R \cdot (R \cup S)^*$ holds.

Lemma 7.2.2. *Let R and S be binary relations.*

1. *If R quasi-commutes over S then well-foundedness of R / S and R coincide.* ✓
2. *If R / S and S are well-founded then $R \cup S$ is well-founded.* ✓

7. Abstract Completion, Formalized

Lemma 7.2.3. *If R is a well-founded rewrite relation then $(R \cup \triangleright) / \triangleright$ is well-founded.* ✔

Proof. First we show the inclusion $\triangleright \cdot R \subseteq R \cdot \triangleright$. Suppose $s \triangleright t R u$. So $s = C[t\sigma]$ for some context C and substitution σ . Because R is closed under contexts and substitutions, $s R C[u\sigma]$. Moreover, $C[u\sigma] \triangleright u$. This establishes the inclusion, and we conclude that R (quasi-)commutes over \triangleright . Because R is well-founded, it follows from Lemma 7.2.2(1) that the relation R / \triangleright is well-founded too. Then R / \triangleright is well-founded since it is contained in R / \triangleright . As \triangleright is well-founded, it follows from Lemma 7.2.2(2) that $R \cup \triangleright$ is well-founded. We have $\triangleright \cdot \triangleright \subseteq \triangleright$ and thus $R \cup \triangleright$ quasi-commutes over \triangleright . Another application of Lemma 7.2.2(1) yields the well-foundedness of $(R \cup \triangleright) / \triangleright$. \square

7.2.2. Abstract Confluence Criteria

We use the following simple confluence criterion for ARSs to replace Newman's Lemma in the correctness proof of abstract completion. In the sequel, we will refer to a conversion of the form $\mathcal{A} \leftarrow \cdot \rightarrow \mathcal{A}$ as a *peak*.

Definition 7.2.4 (Peak Decreasingness ✔). *An ARS $\mathcal{A} = \langle A, \{\rightarrow_\alpha\}_{\alpha \in I} \rangle$ is peak decreasing if there exists a well-founded order $>$ on I such that for all $\alpha, \beta \in I$ the inclusion*

$$\alpha \leftarrow \cdot \rightarrow \beta \subseteq \xleftrightarrow[\vee \alpha \beta]{*}$$

holds. Here $\vee \alpha \beta$ denotes the set $\{\gamma \in I \mid \alpha > \gamma \text{ or } \beta > \gamma\}$ and if $J \subseteq I$ then \xleftrightarrow_J^ denotes a conversion consisting of $\rightarrow_J = \bigcup \{\rightarrow_\gamma \mid \gamma \in J\}$ steps.*

Peak decreasingness is a special case of decreasing diagrams [106], which is known as a very powerful confluence criterion. For the sake of completeness, we present an easy direct (and formalized) proof of the sufficiency of peak decreasingness for confluence. We denote by $\mathcal{M}(J)$ the set of all multisets over a set J .

Lemma 7.2.5. *Every peak decreasing ARS is confluent.* ✔

Proof. Let $>$ be a well-founded order on I which shows that the ARS $\mathcal{A} = \langle A, \{\rightarrow_\alpha\}_{\alpha \in I} \rangle$ is peak decreasing. With every conversion C in \mathcal{A} we associate the multiset M_C consisting of the labels of its steps. These multisets are compared by the multiset extension $>_{\text{mul}}$ of $>$, which is a well-founded order on $\mathcal{M}(I)$. We prove $\leftrightarrow^* \subseteq \downarrow$ by well-founded induction on $>_{\text{mul}}$. Consider a conversion C between a and b . We either have $a \downarrow b$ or $a \leftrightarrow^* \cdot \leftarrow \cdot \rightarrow \cdot \leftrightarrow^* b$. In the former case we are done. In the latter case there exist labels $\alpha, \beta \in I$ and multisets $\Gamma_1, \Gamma_2 \in \mathcal{M}(A)$ such that $M_C = \Gamma_1 \uplus \{\alpha, \beta\} \uplus \Gamma_2$. By the peak decreasingness assumption there exists a conversion C' between a and b such that $M_{C'} = \Gamma_1 \uplus \Gamma \uplus \Gamma_2$ with $\Gamma \in \mathcal{M}(\vee \alpha \beta)$. We obviously have $\{\alpha, \beta\} >_{\text{mul}} \Gamma$ and hence $M_C >_{\text{mul}} M_{C'}$. Finally, we obtain $a \downarrow b$ from the induction hypothesis. \square

A similar criterion to show the Church-Rosser modulo property will be used in Section 7.8. Here an ARS \mathcal{A} is called *Church-Rosser modulo* an ARS \mathcal{B} if the inclusion

$$\xleftrightarrow[\mathcal{A} \cup \mathcal{B}]{} \subseteq \xrightarrow[\mathcal{A}]{} \cdot \xleftrightarrow[\mathcal{B}]{} \cdot \xleftrightarrow[\mathcal{A}]{} \cdot$$

holds.

Definition 7.2.6 (Peak Decreasingness Modulo \checkmark). *Consider the two ARSs $\mathcal{A} = \langle A, \{\rightarrow_\alpha\}_{\alpha \in I} \rangle$ and $\mathcal{B} = \langle B, \{\rightarrow_\beta\}_{\beta \in J} \rangle$. Then \mathcal{A} is peak decreasing modulo \mathcal{B} if there exists a well-founded order $>$ on $I \cup J$ such that for all $\alpha \in I$ and $\gamma \in I \cup J$ the inclusion*

$$\alpha \leftarrow \cdot \rightarrow_\gamma \subseteq \xleftrightarrow[\forall \alpha \gamma]{*}$$

holds. Here $\forall \alpha \gamma$ denotes the set $\{\delta \in I \cup J \mid \alpha > \delta \text{ or } \gamma > \delta\}$.

Lemma 7.2.7. *If \mathcal{A} is peak decreasing modulo \mathcal{B} then \mathcal{A} is Church-Rosser modulo \mathcal{B} .* \checkmark

Proof. Let $x_1 \leftrightarrow_{\alpha_1} \cdots \leftrightarrow_{\alpha_n} x_{n+1}$ and $M = \{\alpha_1, \dots, \alpha_n\}$. We use induction on M with respect to $>_{\text{mul}}$ to show $x_1 \rightarrow_{\mathcal{A}}^* \cdot \leftrightarrow_{\mathcal{B}}^* \cdot \xleftarrow{\mathcal{A}}^* x_{n+1}$. If the given conversion is not of the desired shape, there is an index $1 \leq i < n$ such that $x_i \xleftarrow{\alpha} x_{i+1} \rightarrow_\gamma x_{i+2}$ or $x_i \xrightarrow{\gamma} x_{i+1} \rightarrow_\alpha x_{i+2}$ for some $\alpha \in I$ and $\gamma \in I \cup J$. As the reasoning is similar, we only consider the former case. By peak decreasingness there are labels β_1, \dots, β_m with $x_i \leftrightarrow_{\beta_1} \cdots \leftrightarrow_{\beta_m} x_{i+2}$ such that $\beta_j \in \forall \alpha \gamma$ for all $1 \leq j \leq m$. Writing N for the multiset $\{\beta_1, \dots, \beta_m\}$, we obtain $M >_{\text{mul}} (M - \{\alpha, \gamma\}) \uplus N$ from $\alpha, \gamma \in M$ and $\{\alpha, \gamma\} >_{\text{mul}} N$. Therefore, the claim follows from the induction hypothesis. \square

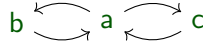
For the correctness proof in Section 7.6 we use a simpler notion than peak decreasingness.

Definition 7.2.8 (Source Decreasingness \checkmark). *Let $\mathcal{A} = \langle A, \rightarrow \rangle$ be an ARS equipped with a well-founded relation $>$ on A , and we write $b \xrightarrow{a} c$ if $b \rightarrow c$ and $a = b$. We say that \mathcal{A} is source decreasing if the inclusion*

$$\leftarrow a \rightarrow \subseteq \xleftrightarrow[\forall a]{*}$$

holds for all $a \in A$. Here $\leftarrow a \rightarrow$ denotes the binary relation consisting of all pairs (b, c) such that $a \rightarrow b$ and $a \rightarrow c$. Moreover, $\xleftrightarrow[\forall a]{}$ denotes the binary relation consisting of all pairs of elements that are connected by a conversion in which all steps are labeled with an element smaller than a .*

Source decreasingness is the specialization of peak decreasingness to source labeling [108, Example 6]. It is closely related to the *connectedness-below* criterion of Winkler and Buchberger [173]. Unlike the latter, source decreasingness does not entail termination. For instance, for $\mathbf{a} > \mathbf{b}$ and $\mathbf{a} > \mathbf{c}$ the non-terminating ARS




is source decreasing but the connectedness-below criterion does not apply.

Lemma 7.2.9. *Every source decreasing ARS is peak decreasing.* \checkmark

Peak decreasingness as a special case of decreasing diagrams was first considered in our ITP publication [58] (the modulo version in Definition 7.2.6 is new). Source decreasingness originates from our later FSCD contribution [59].



7.2.3. Critical Peaks

Completion is based on critical pair analysis. In this subsection we present a version of the critical pair lemma that incorporates primality (cf. Definition 7.2.13 below).

Definition 7.2.10 (Overlaps ) *An overlap of a TRS \mathcal{R} is a triple $\langle \ell_1 \rightarrow r_1, p, \ell_2 \rightarrow r_2 \rangle$, consisting of two rewrite rules and a position, satisfying the following properties:*

- *there are renamings π_1 and π_2 such that $\pi_1(\ell_1 \rightarrow r_1), \pi_2(\ell_2 \rightarrow r_2) \in \mathcal{R}$ (that is, the rules are variants of rules in \mathcal{R}),*
- *$\text{Var}(\ell_1 \rightarrow r_1) \cap \text{Var}(\ell_2 \rightarrow r_2) = \emptyset$ (that is, the rules have no common variables),*
- *$p \in \text{Pos}_{\mathcal{F}}(\ell_2)$,*
- *ℓ_1 and $\ell_2|_p$ are unifiable,*
- *if $p = \epsilon$ then $\ell_1 \rightarrow r_1$ and $\ell_2 \rightarrow r_2$ are not variants of each other.*

In general this definition may lead to an infinite set of overlaps, since there are infinitely many possibilities of taking variable disjoint variants of rules. Fortunately it can be shown that overlaps that originate from the same two rules are variants of each other. Overlaps give rise to critical peaks and pairs.

Definition 7.2.11 (Critical Peaks  and Pairs ) *Suppose $\langle \ell_1 \rightarrow r_1, p, \ell_2 \rightarrow r_2 \rangle$ is an overlap of a TRS \mathcal{R} . Let σ be a most general unifier of ℓ_1 and $\ell_2|_p$. The term $\ell_2\sigma[\ell_1\sigma]_p = \ell_2\sigma$ can be reduced in two different ways:*

$$\begin{array}{ccc} & \ell_2\sigma[\ell_1\sigma]_p = \ell_2\sigma & \\ \ell_1 \rightarrow r_1 \swarrow & & \searrow \ell_2 \rightarrow r_2 \\ & \swarrow p & \searrow \epsilon \\ \ell_2\sigma[r_1\sigma]_p & & r_2\sigma \end{array}$$

We call the quadruple $(\ell_2\sigma[r_1\sigma]_p, p, \ell_2\sigma, r_2\sigma)$ a critical peak and the equation $\ell_2\sigma[r_1\sigma]_p \approx r_2\sigma$ a critical pair of \mathcal{R} , obtained from the overlap. The set of all critical pairs of \mathcal{R} is denoted by $\text{CP}(\mathcal{R})$.

In our formalization of the above definition, instead of an arbitrary most general unifier, we use *the* most general unifier computed by the formalized unification algorithm that is part of `IsaFoR` (thereby removing one degree of freedom and making it easier to show that only finitely many critical pairs have to be considered for finite TRSs).

A critical peak (t, p, s, u) is usually denoted by $t \xrightarrow{p} s \xrightarrow{\epsilon} u$. It can be shown that different critical peaks and pairs obtained from two variants of the same overlap are variants of each other. Since rewriting is equivariant under permutations, it is enough to consult finitely many critical pairs or peaks for finite TRSs (one for each pair of rules and each appropriate position) in order to conclude rewriting related properties (like joinability or fairness, see below) for all of them.

We present a variation of the well-known critical pair lemma for critical peaks and its formalized proof. The slightly cumbersome statement is essential to avoid gaps in the proof of Lemma 7.2.15 below.

Lemma 7.2.12. *Let \mathcal{R} be a TRS. If $t \mathcal{R} \stackrel{p_1}{\leftarrow} s \stackrel{p_2}{\rightarrow} u$ then one of the following holds: \checkmark*

1. $t \downarrow_{\mathcal{R}} u$,
2. $p_2 \leq p_1$ and $t|_{p_2} \stackrel{p_1 \setminus p_2}{\leftarrow} s|_{p_2} \stackrel{\epsilon}{\rightarrow} u|_{p_2}$ is an instance of a critical peak, or
3. $p_1 \leq p_2$ and $u|_{p_1} \stackrel{p_2 \setminus p_1}{\leftarrow} s|_{p_1} \stackrel{\epsilon}{\rightarrow} t|_{p_1}$ is an instance of a critical peak.

Proof. Consider an arbitrary peak $t \xrightarrow{p_1, \ell'_1 \rightarrow r'_1, \sigma'_1} s \xrightarrow{p_2, \ell_2 \rightarrow r_2, \sigma_2} u$. If $p_1 \parallel p_2$ then

$$t \xrightarrow{p_2, \ell_2 \rightarrow r_2, \sigma_2} t[r_2 \sigma_2]_{p_2} = u[r'_1 \sigma'_1]_{p_1} \xrightarrow{p_1, \ell'_1 \rightarrow r'_1, \sigma'_1} u$$

If the positions of the contracted redexes are not parallel then one of them is above the other. Without loss of generality we assume that $p_1 \geq p_2$. Let $p = p_1 \setminus p_2$. Moreover, let π be a permutation such that $\ell_1 \rightarrow r_1 = \pi(\ell'_1 \rightarrow r'_1)$ and $\ell_2 \rightarrow r_2$ have no variables in common. Such a permutation exists since we only have to avoid the finitely many variables of $\ell_2 \rightarrow r_2$ and assume an infinite set of variables. Furthermore, let $\sigma_1 = \pi^{-1} \cdot \sigma'_1$. We have $t = s[r_1 \sigma_1]_{p_1} = s[\ell_2 \sigma_2[r_1 \sigma_1]_p]_{p_2}$ and $u = s[r_2 \sigma_2]_{p_2}$. We consider two cases depending on whether $p \in \mathcal{Pos}_{\mathcal{F}}(\ell_2)$ in conjunction with the fact that whenever $p = \epsilon$ then $\ell_1 \rightarrow r_1$ and $\ell_2 \rightarrow r_2$ are not variants, is true or not.

- Suppose $p \in \mathcal{Pos}_{\mathcal{F}}(\ell_2)$ and $p = \epsilon$ implies that $\ell_1 \rightarrow r_1$ and $\ell_2 \rightarrow r_2$ are not variants. Let $\sigma'(x) = \sigma_1(x)$ for $x \in \mathcal{Var}(\ell_1 \rightarrow r_1)$ and $\sigma'(x) = \sigma_2(x)$, otherwise. The substitution σ' is a unifier of $\ell_2|_p$ and ℓ_1 : $(\ell_2|_p)\sigma' = (\ell_2 \sigma_2)|_p = \ell_1 \sigma_1 = \ell_1 \sigma'$. Then $\langle \ell_1 \rightarrow r_1, p, \ell_2 \rightarrow r_2 \rangle$ is an overlap. Let σ be a most general unifier of $\ell_2|_p$ and ℓ_1 . Hence $\ell_2 \sigma[r_1 \sigma]_p \stackrel{p}{\leftarrow} \ell_2 \sigma \stackrel{\epsilon}{\rightarrow} r_2 \sigma$ is a critical peak and there exists a substitution τ such that $\sigma' = \sigma \tau$. Therefore

$$\ell_2 \sigma_2[r_1 \sigma_1]_p = (\ell_2 \sigma[r_1 \sigma]_p) \tau \stackrel{p}{\leftarrow} (\ell_2 \sigma) \tau \stackrel{\epsilon}{\rightarrow} (r_2 \sigma) \tau = r_2 \sigma_2$$

and thus (2) is obtained.

- Otherwise, either $p = \epsilon$ and $\ell_1 \rightarrow r_1$, $\ell_2 \rightarrow r_2$ are variants, or $p \notin \mathcal{Pos}_{\mathcal{F}}(\ell_2)$. In the former case it is easy to show that $r_1 \sigma_1 = r_2 \sigma_2$ and hence $t = u$. In the latter case, there exist positions q_1, q_2 such that $p = q_1 q_2$ and $q_1 \in \mathcal{Pos}_{\mathcal{V}}(\ell_2)$. Let $\ell_2|_{q_1}$ be the variable x . We have $\sigma_2(x)|_{q_2} = \ell_1 \sigma_1$. Define the substitution σ'_2 as follows:

$$\sigma'_2(y) = \begin{cases} \sigma_2(y)[r_1 \sigma_1]_{q_2} & \text{if } y = x \\ \sigma_2(y) & \text{if } y \neq x \end{cases}$$


Clearly $\sigma_2(x) \rightarrow_{\mathcal{R}} \sigma'_2(x)$, and thus $r_2 \sigma_2 \rightarrow^* r_2 \sigma'_2$. We also have


$$\ell_2 \sigma_2[r_1 \sigma_1]_p = \ell_2 \sigma_2[\sigma'_2(x)]_{q_1} \rightarrow^* \ell_2 \sigma'_2 \rightarrow r_2 \sigma'_2$$


Consequently, $t \rightarrow^* s[r_2 \sigma'_2]_{p_2} \stackrel{*}{\leftarrow} u$. Hence, (1) is concluded. \square

7. Abstract Completion, Formalized


An easy consequence of the above lemma is that for every peak $t \mathcal{R} \leftarrow s \rightarrow_{\mathcal{R}} u$ we have $t \downarrow_{\mathcal{R}} u$ or $t \leftrightarrow_{\text{PCP}(\mathcal{R})} u$. It might be interesting to note that in our formalization of the above proof we do actually not need the fact that left-hand sides of rules are not variables.

Definition 7.2.13 (Prime Critical Peaks and Pairs ) *A critical peak $t \xleftarrow{p} s \xrightarrow{\epsilon} u$ is prime if all proper subterms of $s|_p$ are normal forms. A critical pair is called prime if it is derived from a prime critical peak. We write $\text{PCP}(\mathcal{R})$ to denote the set of all prime critical pairs of a TRS \mathcal{R} .*


Definition 7.2.14. *Given a TRS \mathcal{R} and terms s , t , and u , we write $t \nabla_s u$ if $s \rightarrow_{\mathcal{R}}^+ t$, $s \rightarrow_{\mathcal{R}}^+ u$, and $t \downarrow_{\mathcal{R}} u$ or $t \leftrightarrow_{\text{PCP}(\mathcal{R})} u$. *

Lemma 7.2.15. *Let \mathcal{R} be a TRS. If $t \xleftarrow{p} s \xrightarrow{\epsilon} u$ is a critical peak then $t \nabla_s^2 u$. *


Proof. First suppose that all proper subterms of $s|_p$ are normal forms. Then $t \approx u \in \text{PCP}(\mathcal{R})$ and thus $t \nabla_s u$. Since also $u \nabla_s u$, we obtain the desired $t \nabla_s^2 u$. This leaves us with the case that there is a proper subterm of $s|_p$ that is not a normal form. By considering an innermost redex in $s|_p$ we obtain a position $q > p$ and a term v such that $s \xrightarrow{q} v$ and all proper subterms of $s|_q$ are normal forms. Now, if $v \xleftarrow{q} s \xrightarrow{\epsilon} u$ is an instance of a critical peak then $v \rightarrow_{\text{PCP}(\mathcal{R})} u$. Otherwise, $v \downarrow_{\mathcal{R}} u$ by Lemma 7.2.12, since $q \not\leq \epsilon$. In both cases we obtain $v \nabla_s u$. Finally, we analyze the peak $t \xleftarrow{p} s \xrightarrow{q} v$ by another application of Lemma 7.2.12.


1. If $t \downarrow_{\mathcal{R}} v$, we obtain $t \nabla_s v$ and thus $t \nabla_s^2 u$, since also $v \nabla_s u$.
2. Since $p < q$, only the case that $v|_p \xleftarrow{q \setminus p} s|_p \xrightarrow{\epsilon} t|_p$ is an instance of a critical peak remains. Moreover, all proper subterms of $s|_q$ are normal forms and thus we have an instance of a prime critical peak. Hence $t \leftrightarrow_{\text{PCP}(\mathcal{R})} v$ and together with $v \nabla_s u$ we conclude $t \nabla_s^2 u$. 

Lemma 7.2.16. *Let \mathcal{R} be a TRS. If $t \mathcal{R} \leftarrow s \rightarrow_{\mathcal{R}} u$ then $t \nabla_s^2 u$. *

Proof. From Lemma 7.2.12, either $t \downarrow_{\mathcal{R}} u$ and we are done, or $t \mathcal{R} \leftarrow s \rightarrow_{\mathcal{R}} u$ contains a (possibly reversed) instance of a critical peak. By Lemma 7.2.15 we conclude the proof, since rewriting is closed under substitutions and contexts. 

The following result is due to Kapur et al. [67, Corollary 4].

Corollary 7.2.17. *A terminating TRS is confluent if and only if all its prime critical pairs are joinable. *

Proof. Let \mathcal{R} be a terminating TRS such that $\text{PCP}(\mathcal{R}) \subseteq \downarrow_{\mathcal{R}}$. We claim that \mathcal{R} is source decreasing. As well-founded order we take $> = \rightarrow_{\mathcal{R}}^+$. Consider an arbitrary peak $t \mathcal{R} \leftarrow s \rightarrow_{\mathcal{R}} u$. Lemma 7.2.16 yields a term v such that $t \nabla_s v \nabla_s u$. From the assumption $\text{PCP}(\mathcal{R}) \subseteq \downarrow_{\mathcal{R}}$ we obtain $t \downarrow_{\mathcal{R}} v \downarrow_{\mathcal{R}} u$. Since $s \rightarrow_{\mathcal{R}}^+ v$, all steps in the conversion $t \downarrow_{\mathcal{R}} v \downarrow_{\mathcal{R}} u$ are labeled with a term that is smaller than s . Since the two steps in the peak receive the same label s , source decreasingness is established and hence we obtain the confluence of \mathcal{R} from Lemma 7.2.5. The reverse direction is trivial. 

Note that unlike for ordinary critical pairs, joinability of prime critical pairs does not imply local confluence.

Example 7.2.18. Consider the TRS \mathcal{R} given by the three rules:


$$\text{f(a)} \rightarrow \text{b} \qquad \text{f(a)} \rightarrow \text{c} \qquad \text{a} \rightarrow \text{a}$$

The set $\text{PCP}(\mathcal{R})$ consists of the two pairs $\text{f(a)} \approx \text{b}$ and $\text{f(a)} \approx \text{c}$, which are trivially joinable. But \mathcal{R} is not locally confluent because the peak $\text{b} \xleftarrow{\mathcal{R}} \text{f(a)} \rightarrow_{\mathcal{R}} \text{c}$ is not joinable.

The critical pair lemma (Lemma 7.2.12) in this section is due to Knuth and Bendix [71] and Huet [61]. The primality critical pair criterion was first presented by Kapur, Musser, and Narendran [67]. Our presentation is based on the simpler correctness arguments from our earlier work [58, 59].

7.3. Correctness for Finite Runs

The original completion procedure by Knuth and Bendix [71] was presented as a concrete algorithm. Later on, Bachmair, Dershowitz, and Hsiang [13] presented an inference system for completion and showed that all *fair* implementations thereof (in particular the original procedure) are correct. Abstracting from a concrete strategy, their approach thus has the advantage to cover a variety of implementations. Below, we recall the inference system, which constitutes the basis of the results presented in this section.

Definition 7.3.1 (Knuth-Bendix Completion ) . The inference system KB_f of abstract (Knuth-Bendix) completion operates on pairs $(\mathcal{E}, \mathcal{R})$ of sets of equations \mathcal{E} and rules \mathcal{R} over a common signature \mathcal{F} . It consists of the following inference rules, where we write \mathcal{E}, \mathcal{R} for a pair $(\mathcal{E}, \mathcal{R})$ and \uplus denotes disjoint set union:

$$\begin{array}{ll}
\text{deduce} & \frac{\mathcal{E}, \mathcal{R}}{\mathcal{E} \cup \{s \approx t\}, \mathcal{R}} \quad \text{if } s \xleftarrow{\mathcal{R}} \cdot \rightarrow_{\mathcal{R}} t \quad \text{compose} \quad \frac{\mathcal{E}, \mathcal{R} \uplus \{s \rightarrow t\}}{\mathcal{E}, \mathcal{R} \cup \{s \rightarrow u\}} \quad \text{if } t \rightarrow_{\mathcal{R}} u \\
\text{orient} & \frac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E}, \mathcal{R} \cup \{s \rightarrow t\}} \quad \text{if } s > t \quad \text{simplify} \quad \frac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E} \cup \{u \approx t\}, \mathcal{R}} \quad \text{if } s \rightarrow_{\mathcal{R}} u \\
& \frac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E}, \mathcal{R} \cup \{t \rightarrow s\}} \quad \text{if } t > s \quad \frac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E} \cup \{s \approx u\}, \mathcal{R}} \quad \text{if } t \rightarrow_{\mathcal{R}} u \\
\text{delete} & \frac{\mathcal{E} \uplus \{s \approx s\}, \mathcal{R}}{\mathcal{E}, \mathcal{R}} \quad \text{collapse} \quad \frac{\mathcal{E}, \mathcal{R} \uplus \{t \rightarrow s\}}{\mathcal{E} \cup \{u \approx s\}, \mathcal{R}} \quad \text{if } t \rightarrow_{\mathcal{R}} u
\end{array}$$

Here $>$ is a fixed reduction order on $\mathcal{T}(\mathcal{F}, \mathcal{V})$.

Definition 7.3.1 differs from most of the inference systems in the literature (like those devised by Bachmair and Dershowitz [10, 12]) in that we do not impose an encompassment

7. Abstract Completion, Formalized

condition on collapse. As long as we only consider *finite* runs (see Definition 7.3.5 below)—like in Sections 7.3 to 7.5—this change is valid (as shown by me and Thiemann [142]).

Concerning notation, we write $(\mathcal{E}, \mathcal{R}) \vdash_f (\mathcal{E}', \mathcal{R}')$ whenever we can obtain $(\mathcal{E}', \mathcal{R}')$ from $(\mathcal{E}, \mathcal{R})$ by applying one of the inference rules of Definition 7.3.1. While it is well-known that applying the inference rules of KB_f does not affect the equational theory induced by $\mathcal{E} \cup \mathcal{R}$, our formulation is new and paves the way for a simple correctness proof.

Lemma 7.3.2. *Suppose $(\mathcal{E}, \mathcal{R}) \vdash_f (\mathcal{E}', \mathcal{R}')$. Then, the following two inclusions hold:*

1. If $s \xrightarrow{\mathcal{E} \cup \mathcal{R}} t$ then $s \xrightarrow{\mathcal{R}'} \cdot \xrightarrow{\mathcal{E}' \cup \mathcal{R}'} \cdot \xrightarrow{\mathcal{R}'} t$. ✓
2. If $s \xrightarrow{\mathcal{E}' \cup \mathcal{R}'} t$ then $s \xrightarrow{\mathcal{E} \cup \mathcal{R}}^* t$. ✓

Proof. By inspecting the inference rules of KB_f we easily obtain the following inclusions:

deduce

$$\mathcal{E} \cup \mathcal{R} \subseteq \mathcal{E}' \cup \mathcal{R}' \qquad \mathcal{E}' \cup \mathcal{R}' \subseteq \mathcal{E} \cup \mathcal{R} \cup \xleftarrow{\mathcal{R}} \cdot \xrightarrow{\mathcal{R}}$$

orient

$$\mathcal{E} \cup \mathcal{R} \subseteq \mathcal{E}' \cup \mathcal{R}' \cup (\mathcal{R}')^{-1} \qquad \mathcal{E}' \cup \mathcal{R}' \subseteq \mathcal{E} \cup \mathcal{R} \cup \mathcal{E}^{-1}$$

delete

$$\mathcal{E} \cup \mathcal{R} \subseteq \mathcal{E}' \cup \mathcal{R}' \cup = \qquad \mathcal{E}' \cup \mathcal{R}' \subseteq \mathcal{E} \cup \mathcal{R}$$

compose

$$\mathcal{E} \cup \mathcal{R} \subseteq \mathcal{E}' \cup \mathcal{R}' \cup \xrightarrow{\mathcal{R}'} \cdot \xleftarrow{\mathcal{R}'} \qquad \mathcal{E}' \cup \mathcal{R}' \subseteq \mathcal{E} \cup \mathcal{R} \cup \xrightarrow{\mathcal{R}} \cdot \xleftarrow{\mathcal{R}}$$

simplify

$$\mathcal{E} \cup \mathcal{R} \subseteq \mathcal{E}' \cup \mathcal{R}' \cup \xrightarrow{\mathcal{R}'} \cdot \xrightarrow{\mathcal{E}'} \cup \xrightarrow{\mathcal{E}'} \cdot \xleftarrow{\mathcal{R}'} \qquad \mathcal{E}' \cup \mathcal{R}' \subseteq \mathcal{E} \cup \mathcal{R} \cup \xleftarrow{\mathcal{R}} \cdot \xrightarrow{\mathcal{E}} \cup \xrightarrow{\mathcal{E}} \cdot \xrightarrow{\mathcal{R}}$$

collapse

$$\mathcal{E} \cup \mathcal{R} \subseteq \mathcal{E}' \cup \mathcal{R}' \cup \xrightarrow{\mathcal{R}'} \cdot \xrightarrow{\mathcal{E}'} \qquad \mathcal{E}' \cup \mathcal{R}' \subseteq \mathcal{E} \cup \mathcal{R} \cup \xleftarrow{\mathcal{R}} \cdot \xrightarrow{\mathcal{R}}$$

Consider for instance the **collapse** rule and suppose that $s \approx t \in \mathcal{E} \cup \mathcal{R}$. If $s \approx t \in \mathcal{E}$ then $s \approx t \in \mathcal{E}'$ because $\mathcal{E} \subseteq \mathcal{E}'$. If $s \approx t \in \mathcal{R}$ then either $s \approx t \in \mathcal{R}'$ or $s \rightarrow_{\mathcal{R}} u$ with $u \approx t \in \mathcal{E}'$ and thus $s \rightarrow_{\mathcal{R}'} \cdot \rightarrow_{\mathcal{E}'} t$. This proves the inclusion on the left. For the inclusion on the right the reasoning is similar. Suppose that $s \approx t \in \mathcal{E}' \cup \mathcal{R}'$. If $s \approx t \in \mathcal{R}'$ then $s \approx t \in \mathcal{R}$ because $\mathcal{R}' \subseteq \mathcal{R}$. If $s \approx t \in \mathcal{E}'$ then either $s \approx t \in \mathcal{E}$ or there exists a rule $u \rightarrow t \in \mathcal{R}$ with $u \rightarrow_{\mathcal{R}} s$ and thus $s \mathcal{R} \leftarrow \cdot \rightarrow_{\mathcal{R}} t$.

Since rewrite relations are closed under contexts and substitutions, the inclusions in the right column prove statement (2). Moreover note that each inclusion in the left column is a special case of

$$\mathcal{E} \cup \mathcal{R} \subseteq \xrightarrow[\mathcal{R}']{=} \cdot \xrightarrow[\mathcal{E}' \cup \mathcal{R}']{=} \cdot \xleftarrow[\mathcal{R}']{=}$$

and thus also statement (1) follows from closure under contexts and substitutions of rewrite relations. \square

Corollary 7.3.3. *If $(\mathcal{E}, \mathcal{R}) \vdash_{\mathbf{f}}^* (\mathcal{E}', \mathcal{R}')$ then the relations $\xrightarrow[\mathcal{E} \cup \mathcal{R}]{*}$ and $\xrightarrow[\mathcal{E}' \cup \mathcal{R}']{*}$ coincide.* \checkmark

The next lemma states that termination of \mathcal{R} is preserved by applications of the inference rules of $\text{KB}_{\mathbf{f}}$. It is the final result in this section whose proof refers to the inference rules.

Lemma 7.3.4. *If $(\mathcal{E}, \mathcal{R}) \vdash_{\mathbf{f}}^* (\mathcal{E}', \mathcal{R}')$ and $\mathcal{R} \subseteq >$ then $\mathcal{R}' \subseteq >$.* \checkmark

Proof. We consider a single step $(\mathcal{E}, \mathcal{R}) \vdash_{\mathbf{f}} (\mathcal{E}', \mathcal{R}')$. The statement of the lemma follows by a straightforward induction proof. Observe that deduce, delete, and simplify do not change the set of rewrite rules and hence $\mathcal{R}' = \mathcal{R} \subseteq >$. For collapse we have $\mathcal{R}' \subsetneq \mathcal{R} \subseteq >$. In the case of orient we have $\mathcal{R}' = \mathcal{R} \cup \{s \rightarrow t\}$ with $s > t$ and hence $\mathcal{R}' \subseteq >$ follows from the assumption $\mathcal{R} \subseteq >$. Finally, consider an application of compose. So $\mathcal{R} = \mathcal{R}'' \uplus \{s \rightarrow t\}$ and $\mathcal{R}' = \mathcal{R}'' \cup \{s \rightarrow u\}$ with $t \rightarrow_{\mathcal{R}} u$. We obtain $s > t$ from the assumption $\mathcal{R} \subseteq >$. Since $>$ is a reduction order, $t > u$ follows from $t \rightarrow_{\mathcal{R}} u$. Transitivity of $>$ yields $s > u$ and hence $\mathcal{R}' \subseteq >$ as desired. \square

To guarantee that the result of a finite $\text{KB}_{\mathbf{f}}$ derivation is a complete TRS equivalent to the initial \mathcal{E} , $\text{KB}_{\mathbf{f}}$ derivations must satisfy the *fairness condition* that prime critical pairs of the final TRS \mathcal{R}_n which were not considered during the derivation are joinable in \mathcal{R}_n .

Definition 7.3.5 (Finite Runs and Fairness). *A finite run for a given ES \mathcal{E} is a finite sequence*

$$\mathcal{E}_0, \mathcal{R}_0 \vdash_{\mathbf{f}} \mathcal{E}_1, \mathcal{R}_1 \vdash_{\mathbf{f}} \cdots \vdash_{\mathbf{f}} \mathcal{E}_n, \mathcal{R}_n$$


such that $\mathcal{E}_0 = \mathcal{E}$ and $\mathcal{R}_0 = \emptyset$. The run is fair if $\mathcal{E}_n = \emptyset$ and

$$\text{PCP}(\mathcal{R}_n) \subseteq \downarrow_{\mathcal{R}_n} \cup \bigcup_{i=0}^n \leftrightarrow_{\mathcal{E}_i}$$


The reason for writing $\leftrightarrow_{\mathcal{E}_i}$ instead of \mathcal{E}_i in the definition of fairness is that critical pairs are ordered, so in a fair run a (prime) critical pair $s \approx t$ of \mathcal{R}_n may be ignored by deduce if $t \approx s$ was generated, or more generally, if $s \leftrightarrow_{\mathcal{E}_i} t$ holds at some point in the run. Non-prime critical pairs can always be ignored. Note that our fairness condition differs from earlier notions by permitting that (prime) critical pairs may be joinable in \mathcal{R}_n . This was done to allow for more flexibility in implementations. Our proofs smoothly extend to the relaxed condition.

7. Abstract Completion, Formalized

According to the main result of this section (Theorem 7.3.8), a completion procedure that produces fair runs is correct. The challenge is the confluence proof of \mathcal{R}_n . We show that \mathcal{R}_n is peak decreasing by labeling rewrite steps (not only in \mathcal{R}_n) with multisets of terms. As well-founded order on these multisets we take the multiset extension $>_{\text{mul}}$ of the given reduction order $>$.

Definition 7.3.6 (Labeled Rewriting ). *Let \rightarrow be a rewrite relation and M a finite multiset of terms. We write $s \xrightarrow{M} t$ if $s \rightarrow t$ and there exist terms $s', t' \in M$ such that $s' \geq s$ and $t' \geq t$. Here \geq denotes the reflexive closure of the given reduction order $>$.*

Since both \rightarrow and \geq are closed under contexts and substitutions, $C[t\sigma] \xrightarrow{M'} C[u\sigma]$ whenever $t \xrightarrow{M} u$ and $M' = \{C[s\sigma] \mid s \in M\}$, for all contexts C and substitutions σ .


Lemma 7.3.7. *Let $(\mathcal{E}, \mathcal{R}) \vdash_f (\mathcal{E}', \mathcal{R}')$. If $t \xrightarrow[\mathcal{E} \cup \mathcal{R}]{M}^* u$ and $\mathcal{R}' \subseteq >$ then $t \xrightarrow[\mathcal{E}' \cup \mathcal{R}']{M}^* u$. *


Proof. We consider a single $(\mathcal{E} \cup \mathcal{R})$ -step from t to u . The lemma follows then by induction on the length of the conversion between t and u . According to Lemma 7.3.2(1) there exist terms v and w such that

$$t \xrightarrow[\mathcal{R}']{=} v \xrightarrow[\mathcal{E}' \cup \mathcal{R}']{=} w \xleftarrow[\mathcal{R}']{=} u$$

We claim that the (non-empty) steps can be labeled by M . There exist terms $t', u' \in M$ with $t' \geq t$ and $u' \geq u$. Since $\mathcal{R}' \subseteq >$, we have $t \geq v$ and $u \geq w$ and thus also $t' \geq v$ and $u' \geq w$. Hence

$$t \xrightarrow[\mathcal{R}']{M} v \xrightarrow[\mathcal{E}' \cup \mathcal{R}']{M} w \xleftarrow[\mathcal{R}']{M} u$$

and thus also $t \xrightarrow[\mathcal{E}' \cup \mathcal{R}']{M}^* u$. 

Theorem 7.3.8. *For every fair run Γ *

$$\mathcal{E}_0, \mathcal{R}_0 \vdash_f \mathcal{E}_1, \mathcal{R}_1 \vdash_f \cdots \vdash_f \mathcal{E}_n, \mathcal{R}_n$$

the TRS \mathcal{R}_n is a complete presentation of \mathcal{E} .

Proof. We have $\mathcal{E}_n = \emptyset$. From Corollary 7.3.3 we know that $\leftrightarrow_{\mathcal{E}}^* = \leftrightarrow_{\mathcal{R}_n}^*$. Lemma 7.3.4 yields $\mathcal{R}_n \subseteq >$ and hence \mathcal{R}_n is terminating. It remains to prove that \mathcal{R}_n is confluent. Let

$$t \xleftarrow[\mathcal{R}_n]{M_1} s \xrightarrow[\mathcal{R}_n]{M_2} u$$

be a labeled local peak in \mathcal{R}_n . From Lemma 7.2.16 we obtain $t \nabla_s^2 u$. Let $v \nabla_s w$ appear in this sequence (so $t = v$ or $w = u$). We obtain

$$(v, w) \in \downarrow_{\mathcal{R}_n} \cup \bigcup_{i=0}^n \leftrightarrow_{\mathcal{E}_i}$$

from the definition of ∇_s and fairness of Γ . We label all steps between v and w with the multiset $\{v, w\}$. Because $s > v$ and $s > w$ we have $M_1 >_{\text{mul}} \{v, w\}$ and $M_2 >_{\text{mul}} \{v, w\}$. Hence by repeated applications of Lemma 7.3.7 we obtain a conversion in \mathcal{R}_n between v and w in which each step is labeled with a multiset that is smaller than both M_1 and M_2 . It follows that \mathcal{R}_n is peak decreasing and thus confluent by Lemma 7.2.5. \square

A completion procedure is a program that generates KB_f runs. In order to ensure that the final outcome \mathcal{R}_n is a complete presentation of the initial ES, fair runs should be produced. Fairness requires that prime critical pairs of \mathcal{R}_n are considered during the run. Of course, \mathcal{R}_n is not known during the run, so to be on the safe side, prime critical pairs of any \mathcal{R} that appears during the run should be generated by *deduce*. In particular, there is no need to deduce equations that are not prime critical pairs. So we may strengthen the condition $s \mathcal{R} \leftarrow \cdot \rightarrow \mathcal{R} t$ of *deduce* to $s \approx t \in \text{PCP}(\mathcal{R})$ without affecting Theorem 7.3.8.

The following example shows that the success of a run may depend on the order in which inference rules are applied [14].

Example 7.3.9. Consider the ES \mathcal{E} consisting of the four equations

$$\mathbf{a} \approx \mathbf{b} \qquad \mathbf{a} \approx \mathbf{c} \qquad \mathbf{f}(\mathbf{b}) \approx \mathbf{b} \qquad \mathbf{f}(\mathbf{a}) \approx \mathbf{d}$$

and the reduction order $>_{\text{lpo}}$ with the partial precedence $\mathbf{a} > \mathbf{b} > \mathbf{d}$ and $\mathbf{a} > \mathbf{c} > \mathbf{d}$ but where \mathbf{b} and \mathbf{c} are incomparable. One possible run is

$$\begin{aligned} (\mathcal{E}, \emptyset) &\vdash_{\mathbf{f}}^{\text{orient}^+} (\{\mathbf{a} \approx \mathbf{c}, \mathbf{f}(\mathbf{a}) \approx \mathbf{d}\}, \{\mathbf{a} \rightarrow \mathbf{b}, \mathbf{f}(\mathbf{b}) \rightarrow \mathbf{b}\}) \\ &\vdash_{\mathbf{f}}^{\text{simplify}^+} (\{\mathbf{b} \approx \mathbf{c}, \mathbf{f}(\mathbf{b}) \approx \mathbf{d}\}, \{\mathbf{a} \rightarrow \mathbf{b}, \mathbf{f}(\mathbf{b}) \rightarrow \mathbf{b}\}) \\ &\vdash_{\mathbf{f}}^{\text{simplify}} (\{\mathbf{b} \approx \mathbf{c}, \mathbf{b} \approx \mathbf{d}\}, \{\mathbf{a} \rightarrow \mathbf{b}, \mathbf{f}(\mathbf{b}) \rightarrow \mathbf{b}\}) \\ &\vdash_{\mathbf{f}}^{\text{orient}} (\{\mathbf{b} \approx \mathbf{c}\}, \{\mathbf{a} \rightarrow \mathbf{b}, \mathbf{f}(\mathbf{b}) \rightarrow \mathbf{b}, \mathbf{b} \rightarrow \mathbf{d}\}) \\ &\vdash_{\mathbf{f}}^{\text{collapse}} (\{\mathbf{b} \approx \mathbf{c}, \mathbf{f}(\mathbf{d}) \approx \mathbf{b}\}, \{\mathbf{a} \rightarrow \mathbf{b}, \mathbf{b} \rightarrow \mathbf{d}\}) \\ &\vdash_{\mathbf{f}}^{\text{simplify}^+} (\{\mathbf{d} \approx \mathbf{c}, \mathbf{f}(\mathbf{d}) \approx \mathbf{d}\}, \{\mathbf{a} \rightarrow \mathbf{b}, \mathbf{b} \rightarrow \mathbf{d}\}) \\ &\vdash_{\mathbf{f}}^{\text{orient}^+} (\emptyset, \{\mathbf{a} \rightarrow \mathbf{b}, \mathbf{b} \rightarrow \mathbf{d}, \mathbf{c} \rightarrow \mathbf{d}, \mathbf{f}(\mathbf{d}) \rightarrow \mathbf{d}\}) \end{aligned}$$

which derives a complete presentation of \mathcal{E} . However, the run

$$\begin{aligned} (\mathcal{E}, \emptyset) &\vdash_{\mathbf{f}}^{\text{orient}} (\{\mathbf{a} \approx \mathbf{b}, \mathbf{f}(\mathbf{b}) \approx \mathbf{b}, \mathbf{f}(\mathbf{a}) \approx \mathbf{d}\}, \{\mathbf{a} \rightarrow \mathbf{c}\}) \\ &\vdash_{\mathbf{f}}^{\text{simplify}^+} (\{\mathbf{c} \approx \mathbf{b}, \mathbf{f}(\mathbf{b}) \approx \mathbf{b}, \mathbf{f}(\mathbf{c}) \approx \mathbf{d}\}, \{\mathbf{a} \rightarrow \mathbf{c}\}) \\ &\vdash_{\mathbf{f}}^{\text{orient}^+} (\{\mathbf{c} \approx \mathbf{b}\}, \{\mathbf{a} \rightarrow \mathbf{c}, \mathbf{f}(\mathbf{b}) \rightarrow \mathbf{b}, \mathbf{f}(\mathbf{c}) \rightarrow \mathbf{d}\}) \end{aligned}$$

cannot be extended to a successful one because the equation $\mathbf{c} \approx \mathbf{b}$ cannot be oriented.

The following example shows that even after a KB_f run derived a complete system, exponentially many steps might be performed to obtain a canonical TRS.

Example 7.3.10. Consider the ESs $\mathcal{E}_n = \{\mathbf{f}(\mathbf{g}^i(\mathbf{c})) \approx \mathbf{g}(\mathbf{f}^i(\mathbf{c})) \mid 0 \leq i \leq n\}$ for $n \geq 1$. By taking the Knuth-Bendix order $>_{\text{kbo}}$ with precedence $\mathbf{f} > \mathbf{g}$ and where $w(\mathbf{f}) = w(\mathbf{g})$, all

7. Abstract Completion, Formalized

equations can be oriented from left to right. Since there are no critical pairs, the resulting TRSs $\mathcal{R}_n = \{f(g^i(c)) \rightarrow g(f^i(c)) \mid 0 \leq i \leq n\}$ are complete by Theorem 7.3.8. However, it is not canonical since right-hand sides are not normal forms. When applying compose steps in a naive way by simplifying the rules in descending order, exponentially many steps are required to obtain a canonical system [118]. However, when processing the rules in reverse order only a polynomial number of steps is necessary.

This section resumes our results on finite runs [58]. The presented correctness proof differs substantially from all earlier proofs in that it does not rely on a proof order [13] but is instead based on peak decreasingness. It supports a relaxed side condition of the collapse rule as first used in [142], but in contrast to the latter demands only prime critical pairs to be considered.

7.4. Canonicity and Normalization Equivalence

A natural question arising in the context of completion concerns uniqueness of resulting systems: Is there a single complete presentation of a given equational theory conforming to a certain reduction order? Métivier [87] showed that for reduced and hence canonical systems this is indeed the case, up to renaming variables. In this section we revisit his work, aiming at generalizing his uniqueness result for canonical TRSs and at establishing a transformation to simplify ground-complete TRSs. A key notion to that end is normalization equivalence.

Definition 7.4.1 (Conversion/Normalization Equivalence). *Two ARSs \mathcal{A} and \mathcal{B} are said to be (conversion) equivalent if $\leftrightarrow_{\mathcal{A}}^* = \leftrightarrow_{\mathcal{B}}^*$ and normalization equivalent if $\rightarrow_{\mathcal{A}}^! = \rightarrow_{\mathcal{B}}^!$.*

The following example shows that these two equivalence notions do not coincide.

Example 7.4.2. *Consider the four ARSs:*

$$\begin{array}{ll} \mathcal{A}_1: & a \longrightarrow b \\ \mathcal{A}_2: & a \longrightarrow b \\ & \quad \uparrow \quad \uparrow \\ & \quad \text{ } \quad \text{ } \end{array} \qquad \begin{array}{ll} \mathcal{B}_1: & a \longleftarrow b \\ \mathcal{B}_2: & a \quad b \\ & \uparrow \quad \uparrow \\ & \text{ } \quad \text{ } \end{array}$$

While \mathcal{A}_1 and \mathcal{B}_1 are conversion equivalent but not normalization equivalent, the ARSs \mathcal{A}_2 and \mathcal{B}_2 are normalization equivalent but not conversion equivalent.

The easy proof (by induction on the length of conversions) of the following result is omitted.

Lemma 7.4.3. *Normalization equivalent terminating ARSs are equivalent.* ✓

Note that the termination assumption can be weakened to weak normalization. However, the present version suffices to prove the following lemma that we employ in our proof of Métivier's transformation result [87] (Theorem 7.4.7 below).

Lemma 7.4.4. *Let \mathcal{A} and \mathcal{B} be ARSs such that $\text{NF}(\mathcal{B}) \subseteq \text{NF}(\mathcal{A})$ and either*

1. $\rightarrow_{\mathcal{B}} \subseteq \rightarrow_{\mathcal{A}}^+$ or ✓
2. $\rightarrow_{\mathcal{B}} \subseteq \leftrightarrow_{\mathcal{A}}^*$ and \mathcal{B} is terminating. ✓

If \mathcal{A} is complete then \mathcal{B} is complete and normalization equivalent to \mathcal{A} .

Proof. We first show $\rightarrow_{\mathcal{B}}^! \subseteq \rightarrow_{\mathcal{A}}^!$. In case (1), from the inclusion $\rightarrow_{\mathcal{B}} \subseteq \rightarrow_{\mathcal{A}}^+$ we infer that \mathcal{B} is terminating. Moreover, $\rightarrow_{\mathcal{B}}^* \subseteq \rightarrow_{\mathcal{A}}^*$ and, since $\text{NF}(\mathcal{B}) \subseteq \text{NF}(\mathcal{A})$, also $\rightarrow_{\mathcal{B}}^! \subseteq \rightarrow_{\mathcal{A}}^!$. For case (2), $\rightarrow_{\mathcal{B}}^! \subseteq \rightarrow_{\mathcal{A}}^!$ holds because $\rightarrow_{\mathcal{B}}^! \subseteq \leftrightarrow_{\mathcal{A}}^*$, so by confluence of \mathcal{A} and $\text{NF}(\mathcal{B}) \subseteq \text{NF}(\mathcal{A})$ we obtain $\rightarrow_{\mathcal{B}}^! \subseteq \rightarrow_{\mathcal{A}}^!$. Next we show that the reverse inclusion $\rightarrow_{\mathcal{A}}^! \subseteq \rightarrow_{\mathcal{B}}^!$ holds in both cases. Let $a \rightarrow_{\mathcal{A}}^! b$. Because \mathcal{B} is terminating, $a \rightarrow_{\mathcal{B}}^! c$ for some $c \in \text{NF}(\mathcal{B})$. So $a \rightarrow_{\mathcal{A}}^! c$ and thus $b = c$ from the confluence of \mathcal{A} . It follows that \mathcal{A} and \mathcal{B} are normalization equivalent. It remains to show that \mathcal{B} is locally confluent. This follows from the sequence of inclusions

$$\mathcal{B} \leftarrow \cdot \rightarrow_{\mathcal{B}} \subseteq \leftrightarrow_{\mathcal{A}}^* \subseteq \rightarrow_{\mathcal{A}}^! \cdot \mathcal{A} \leftarrow \subseteq \rightarrow_{\mathcal{B}}^! \cdot \mathcal{B} \leftarrow$$

where we obtain the inclusions from $\rightarrow_{\mathcal{B}} \subseteq \leftrightarrow_{\mathcal{A}}^*$, confluence of \mathcal{A} , termination of \mathcal{A} , and normalization equivalence of \mathcal{A} and \mathcal{B} , respectively. \square

In the above lemma, completeness can be weakened to semi-completeness (that is, the combination of confluence and weak normalization), which is not true for Theorem 7.4.7 as shown by Gramlich [51]. Again, the present version suffices for our purposes. Condition (2) of the lemma can be regarded as a specialization of an abstract result of Toyama [167, Corollary 3.2] to complete systems and will be used in Section 7.7.

Theorem 7.4.7 below shows that we can always eliminate redundancy in a complete TRS. This is achieved by the following two-stage transformation, where, given a TRS \mathcal{R} , we write \mathcal{R}_{\perp} for a set of representatives of the equivalence classes of rules in \mathcal{R} with respect to \doteq (that is, \mathcal{R}_{\perp} is a variant-free version of \mathcal{R}).

Definition 7.4.5. *Given a terminating TRS \mathcal{R} , the TRSs $\dot{\mathcal{R}}$ and $\ddot{\mathcal{R}}$ are defined as follows:*

$$\begin{aligned} \dot{\mathcal{R}} &= \{\ell \rightarrow r \downarrow_{\mathcal{R}} \mid \ell \rightarrow r \in \mathcal{R}\}_{\perp} & \checkmark \\ \ddot{\mathcal{R}} &= \{\ell \rightarrow r \in \dot{\mathcal{R}} \mid \ell \in \text{NF}(\dot{\mathcal{R}} \setminus \{\ell \rightarrow r\})\} & \checkmark \end{aligned}$$

Here $t \downarrow_{\mathcal{R}}$ stands for an arbitrary but fixed normal form of t .

The TRS $\dot{\mathcal{R}}$ is obtained from \mathcal{R} by first normalizing the right-hand sides and then taking representatives of variants of the resulting rules, thereby making sure that the result does not contain several variants of the same rule. To obtain $\ddot{\mathcal{R}}$ we remove the rules of $\dot{\mathcal{R}}$ whose left-hand sides are reducible with another rule of $\dot{\mathcal{R}}$. (This is the only place in the paper where variant-freeness of TRSs is important.)

The following example shows why the result of $\ddot{\mathcal{R}}$ has to be variant-free.

7. Abstract Completion, Formalized

Example 7.4.6. Consider the TRS \mathcal{R} consisting of the four rules

$$f(x) \rightarrow a \qquad f(y) \rightarrow b \qquad a \rightarrow c \qquad b \rightarrow c$$

Then the first transformation without taking representatives of rules would yield $\dot{\mathcal{R}}$


$$f(x) \rightarrow c \qquad f(y) \rightarrow c \qquad a \rightarrow c \qquad b \rightarrow c$$

and the second one $\ddot{\mathcal{R}}$

$$a \rightarrow c \qquad b \rightarrow c$$

Note that $\ddot{\mathcal{R}}$ is not equivalent to \mathcal{R} . This is caused by the fact that the result of the first transformation was no longer variant-free.

The following result, due to Métivier [87, Theorem 7], allows us to obtain a canonical representation of any complete TRS.³ Our proof below proceeds by induction on the well-founded encompassment order \triangleright .

Theorem 7.4.7. *If \mathcal{R} is a complete TRS then $\ddot{\mathcal{R}}$ is a normalization and conversion equivalent canonical TRS.* 

Proof. Let \mathcal{R} be a complete TRS. The inclusions $\ddot{\mathcal{R}} \subseteq \dot{\mathcal{R}} \subseteq \rightarrow_{\mathcal{R}}^+$ are obvious from the definitions. Since \mathcal{R} and $\dot{\mathcal{R}}$ have the same left-hand sides, their normal forms coincide. We show that $\text{NF}(\ddot{\mathcal{R}}) \subseteq \text{NF}(\dot{\mathcal{R}})$. To this end we show that $\ell \notin \text{NF}(\ddot{\mathcal{R}})$ whenever $\ell \rightarrow r \in \dot{\mathcal{R}}$ by induction on ℓ with respect to the well-founded order \triangleright . If $\ell \rightarrow r \in \ddot{\mathcal{R}}$ then $\ell \notin \text{NF}(\ddot{\mathcal{R}})$ holds. So suppose $\ell \rightarrow r \notin \ddot{\mathcal{R}}$. By definition of $\ddot{\mathcal{R}}$, $\ell \notin \text{NF}(\dot{\mathcal{R}} \setminus \{\ell \rightarrow r\})$. So there exists a rewrite rule $\ell' \rightarrow r' \in \dot{\mathcal{R}}$ different from $\ell \rightarrow r$ such that $\ell \triangleright \ell'$. We distinguish two cases.

- If $\ell \triangleright \ell'$ then we obtain $\ell' \notin \text{NF}(\ddot{\mathcal{R}})$ from the induction hypothesis and hence $\ell \notin \text{NF}(\ddot{\mathcal{R}})$ as desired.
- If $\ell \doteq \ell'$ then by Lemma 7.2.1 there exists a renaming σ such that $\ell = \ell'\sigma$. Since $\dot{\mathcal{R}}$ is right-reduced by construction, r and r' are normal forms of $\dot{\mathcal{R}}$. The same holds for $r'\sigma$ because normal forms are closed under renaming. We have $r \dot{\mathcal{R}} \leftarrow \ell = \ell'\sigma \rightarrow_{\dot{\mathcal{R}}} r'\sigma$. Since $\dot{\mathcal{R}}$ is confluent as a consequence of Lemma 7.4.4(1), $r = r'\sigma$. Hence $\ell' \rightarrow r'$ is a variant of $\ell \rightarrow r$, contradicting the fact that $\dot{\mathcal{R}}$ is variant-free (by construction).

From Lemma 7.4.4(1) we infer that the TRSs $\dot{\mathcal{R}}$ and $\ddot{\mathcal{R}}$ are complete and normalization equivalent to \mathcal{R} . The TRS $\ddot{\mathcal{R}}$ is right-reduced because $\ddot{\mathcal{R}} \subseteq \dot{\mathcal{R}}$ and $\dot{\mathcal{R}}$ is right-reduced. From $\text{NF}(\ddot{\mathcal{R}}) = \text{NF}(\dot{\mathcal{R}})$ we easily infer that $\ddot{\mathcal{R}}$ is left-reduced. It follows that $\ddot{\mathcal{R}}$ is canonical. It remains to show that $\ddot{\mathcal{R}}$ is not only normalization equivalent but also (conversion) equivalent to \mathcal{R} . This is an immediate consequence of Lemma 7.4.3. \square

³We were not able to reconstruct enough detail for an Isabelle/HOL formalization from its original proof. Another textbook proof [163, Exercise 7.4.7] involves 13 steps with lots of redundancy.

7.4. Canonicity and Normalization Equivalence

Before we proceed to show uniqueness of normalization equivalent TRSs, we need the following technical lemma.

Lemma 7.4.8. *Let \mathcal{R} be a right-reduced TRS and let s be a reducible term which is minimal with respect to \triangleright . If $s \rightarrow_{\mathcal{R}}^+ t$ then $s \rightarrow t$ is a variant of a rule in \mathcal{R} .* ✓

Proof. Let $\ell \rightarrow r$ be the rewrite rule that is used in the first step from s to t . So $s \triangleright \ell$. By assumption, $s \triangleright \ell$ does not hold and thus $s \dot{=} \ell$ because $\triangleright = \triangleright \cup \dot{=}$. According to Lemma 7.2.1 there exists a renaming σ such that $s = \ell\sigma$. We have $s \rightarrow_{\mathcal{R}} r\sigma \rightarrow_{\mathcal{R}}^* t$. Because \mathcal{R} is right-reduced, $r \in \text{NF}(\mathcal{R})$. Since normal forms are closed under renaming, also $r\sigma \in \text{NF}(\mathcal{R})$ and thus $r\sigma = t$. It follows that $s \rightarrow t$ is a variant of $\ell \rightarrow r$. □

In our formalization, the above proof is the first spot of this section where we actually need that \mathcal{R} satisfies the variable condition (more precisely, only the part of it that right-hand sides of rules do not introduce fresh variables). We are now in a position to present the main result of this section.

Theorem 7.4.9. *Normalization equivalent reduced TRSs are unique up to literal similarity.* ✓

Proof. Let \mathcal{R} and \mathcal{S} be normalization equivalent reduced TRSs. Suppose $\ell \rightarrow r \in \mathcal{R}$. Because \mathcal{R} is right-reduced, $r \in \text{NF}(\mathcal{R})$ and thus $\ell \neq r$. Hence $\ell \rightarrow_{\mathcal{S}}^+ r$ by normalization equivalence. Because \mathcal{R} is left-reduced, ℓ is a minimal (with respect to \triangleright) \mathcal{R} -reducible term. Another application of normalization equivalence yields that ℓ is minimal \mathcal{S} -reducible. Hence $\ell \rightarrow r$ is a variant of a rule in \mathcal{S} by Lemma 7.4.8. □

Example 7.4.10. *Consider the rewrite system \mathcal{R} of combinatory logic with equality test, studied by Klop [70]:*

$$\begin{array}{ll} @(@(@(\mathbf{S}, x), y), z) \rightarrow @(\mathbf{x}, @(\mathbf{z}, @(\mathbf{y}, \mathbf{z}))) & @(@(\mathbf{K}, x), y) \rightarrow \mathbf{x} \\ @(\mathbf{I}, x) \rightarrow x & @(@(\mathbf{D}, x), x) \rightarrow \mathbf{E} \end{array}$$

The rewrite system \mathcal{R} is reduced, but neither terminating nor confluent. One might ask whether there is another reduced rewrite system that computes the same normal forms for every starting term. Theorem 7.4.9 shows that \mathcal{R} is unique up to variable renaming.

As the final result of this section, we prove this result of Métivier [87, Theorem 8] to be an easy consequence of Theorem 7.4.9. Here a TRS \mathcal{R} is said to be compatible with a reduction order $>$ if $\ell > r$ for every rewrite rule $\ell \rightarrow r$ of \mathcal{R} .

Theorem 7.4.11. *Let \mathcal{R} and \mathcal{S} be equivalent canonical TRSs. If \mathcal{R} and \mathcal{S} are compatible with the same reduction order then $\mathcal{R} \dot{=} \mathcal{S}$.* ✓

Proof. Suppose \mathcal{R} and \mathcal{S} are compatible with the reduction order $>$. We show that $\rightarrow_{\mathcal{R}}^! \subseteq \rightarrow_{\mathcal{S}}^!$. Let $s \rightarrow_{\mathcal{R}}^! t$. We show that $t \in \text{NF}(\mathcal{S})$. Let u be the unique \mathcal{S} -normal form of t . We have $t \rightarrow_{\mathcal{S}}^! u$ and thus $t \leftrightarrow_{\mathcal{R}}^* u$ because \mathcal{R} and \mathcal{S} are equivalent. Since $t \in \text{NF}(\mathcal{R})$, we have $u \rightarrow_{\mathcal{R}}^! t$. If $t \neq u$ then both $t > u$ (as $t \rightarrow_{\mathcal{S}}^! u$) and $u > t$ (as $u \rightarrow_{\mathcal{R}}^! t$), which

7. Abstract Completion, Formalized

is impossible. Hence $t = u$ and thus $t \in \text{NF}(\mathcal{S})$. Together with $s \leftrightarrow_{\mathcal{S}}^* t$, which follows from the equivalence of \mathcal{R} and \mathcal{S} , we conclude that $s \rightarrow_{\mathcal{S}}^! t$. We obtain $\rightarrow_{\mathcal{S}}^! \subseteq \rightarrow_{\mathcal{R}}^!$ by symmetry. Hence \mathcal{R} and \mathcal{S} are normalization equivalent and the result follows from Theorem 7.4.9. \square

This section resumes our results on canonicity [59]. While the results of Theorem 7.4.7 and Theorem 7.4.11 are due to Métivier[87], we present novel and simpler proofs based on the (new) auxiliary results Lemma 7.4.4 and Theorem 7.4.9.

7.5. Ground Completion

In this section we focus on the special case of ground equations, that is, equations where both sides are ground terms.

Definition 7.5.1 (Ground Completion \checkmark). *The inference system KB_{g} consists of the inference rules of KB_{f} except for deduce.*

Snyder [128] proved that sets of ground equations can always be completed by KB_{g} , provided a *ground-total* reduction order $>$ is used, that is, for all ground terms $s, t \in \mathcal{T}(\mathcal{F})$ either $s > t$, $t > s$, or $s = t$. He further proved that every reduced ground rewrite system is canonical and can be obtained by completion from any equivalent set of ground equations. Below, we present the proofs of these results that we formalized in Isabelle/HOL.

The following example illustrates the inference system KB_{g} on a set of ground equations.

Example 7.5.2. *Consider the ES \mathcal{E} consisting of the ground equations*

$$f(f(f(a))) \approx f(b) \quad f(f(b)) \approx c \quad f(c) \approx a \quad f(a) \approx f(f(b))$$

As reduction order we take LPO induced by the total precedence $a > b > c > f$. We start by applying orient to the last two equations:

$$f(f(f(a))) \approx f(b) \quad f(f(b)) \approx c \quad f(c) \leftarrow a \quad f(a) \rightarrow f(f(b))$$

An application of collapse produces

$$f(f(f(a))) \approx f(b) \quad f(f(b)) \approx c \quad f(c) \leftarrow a \quad f(f(c)) \approx f(f(b))$$

Next we orient the second equation:

$$f(f(f(a))) \approx f(b) \quad f(f(b)) \rightarrow c \quad f(c) \leftarrow a \quad f(f(c)) \approx f(f(b))$$

Two applications of simplify produce

$$f(f(f(f(c)))) \approx f(b) \quad f(f(b)) \rightarrow c \quad f(c) \leftarrow a \quad f(f(c)) \approx c$$

We continue by orienting the last equation:

$$f(f(f(f(c)))) \approx f(b) \quad f(f(b)) \rightarrow c \quad f(c) \leftarrow a \quad f(f(c)) \rightarrow c$$

Two applications of *simplify* produce

$$c \approx f(b) \quad f(f(b)) \rightarrow c \quad f(c) \leftarrow a \quad f(f(c)) \rightarrow c$$

Orienting the remaining equation followed by a collapse step produces

$$c \leftarrow f(b) \quad f(c) \approx c \quad f(c) \leftarrow a \quad f(f(c)) \rightarrow c$$

Finally, we orient the only remaining equation and collapse, compose, simplify, and delete exhaustively, thereby obtaining the TRS \mathcal{R}

$$c \leftarrow f(b) \quad f(c) \rightarrow c \quad c \leftarrow a$$

which constitutes a canonical presentation of \mathcal{E} .

The absence of *deduce* from KB_g does not hurt for ground systems. If $s \leftarrow \cdot \rightarrow t$ and the two contracted redexes are at parallel positions then trivially $s \rightarrow \cdot \leftarrow t$. If the steps are identical then $s = t$. In the remaining case one of the contracted redexes is a subterm of the other contracted redex, and the effect of *deduce* is achieved by the *collapse* inference rule. On the contrary, the absence of *deduce* is crucial to conclude that KB_g derivations are always finite, as illustrated by the following simple example.

Example 7.5.3. Consider the ground ES \mathcal{E} consisting of the single equation $a \approx b$ and LPO induced by the precedence $a > b$. Using KB_f (that is, KB_g with *deduce*) the following infinite run is possible:

$$\begin{aligned} (\mathcal{E}, \emptyset) &\vdash_f^{\text{orient}} (\emptyset, \{a \rightarrow b\}) \\ &\vdash_f^{\text{deduce}} (\{b \approx b\}, \{a \rightarrow b\}) \\ &\vdash_f^{\text{delete}} (\emptyset, \{a \rightarrow b\}) \\ &\vdash_f^{\text{deduce}} \dots \end{aligned}$$

Lemma 7.5.4. There are no infinite runs $\mathcal{E}_0, \emptyset \vdash_g \mathcal{E}_1, \mathcal{R}_1 \vdash_g \dots$ for finite ground ES \mathcal{E}_0 . ✓

Proof. Let \succ denote the lexicographic combination of the multiset extension $>_{\text{mul}}$ of the reduction order $>$ with the standard order on natural numbers $>_{\mathbb{N}}$. Furthermore let $M(\mathcal{E}, \mathcal{R})$ denote the (finite) multiset of left-hand sides and right-hand sides occurring in \mathcal{E} and \mathcal{R}

$$M(\mathcal{E}, \mathcal{R}) = \bigcup \{\{s, t\} \mid (s, t) \in \mathcal{E}\} \cup \bigcup \{\{s, t\} \mid (s, t) \in \mathcal{R}\}$$

and consider the function P that maps the pair $(\mathcal{E}, \mathcal{R})$ to $(M(\mathcal{E}, \mathcal{R}), |\mathcal{E}|)$. Now it is straightforward to verify that any infinite \vdash_g -sequence would give rise to an infinite sequence $P(\mathcal{E}_0, \emptyset) \succ P(\mathcal{E}_1, \mathcal{R}_1) \succ \dots$, contradicting the well-foundedness of \succ . □

Theorem 7.5.5. If $>$ is total on \mathcal{E} -equivalent ground terms then every maximal KB_g run produces an equivalent canonical presentation for every ground ES \mathcal{E} . ✓

7. Abstract Completion, Formalized

Proof. Consider a maximal KB_g run $\mathcal{E}_0, \emptyset \vdash_g \mathcal{E}_1, \mathcal{R}_1 \vdash_g \dots \vdash_g \mathcal{E}_n, \mathcal{R}_n$ where $\mathcal{E}_0 = \mathcal{E}$ is a ground ES. Because the run is maximal, no inference rule of KB_g is applicable to the final pair $(\mathcal{E}_n, \mathcal{R}_n)$. In particular, *compose* and *collapse* are not applicable and hence the final TRS \mathcal{R}_n is reduced. Since \mathcal{R}_n is ground, this means in particular that there are no critical pairs. Moreover, termination of \mathcal{R}_n follows from Lemma 7.3.4 (since any KB_g run is also a KB_f run), so \mathcal{R}_n is canonical. From Corollary 7.3.3 and the inclusion $\text{KB}_g \subseteq \text{KB}_f$ we infer that \mathcal{E} and $\mathcal{E}_n \cup \mathcal{R}_n$ are equivalent. It follows that $>$ is total on \mathcal{E}_n -equivalent ground terms and thus $\mathcal{E}_n = \emptyset$, for otherwise the run could be extended with an application of *delete* or *orient*. Hence \mathcal{R}_n and \mathcal{E} are equivalent. \square

The restriction on the reduction order $>$ in the above correctness theorem is easy to satisfy. In particular, it holds for any LPO or KBO based on a total precedence.

Next we consider completeness of ground completion. Our proof makes use of the following concept.

Definition 7.5.6 (Random Descent \checkmark). *An ARS \mathcal{A} has random descent if for every conversion $a \leftrightarrow^* b$ with normal form b we have $a \rightarrow^n b$ with $n + l = r$. Here l (r) denotes the number of \leftarrow (\rightarrow) steps in the conversion $a \leftrightarrow^* b$.*

Random descent is useful in the analysis of rewrite strategies [109]. It generalizes a number of earlier concepts, including the property $\leftarrow \cdot \rightarrow \subseteq (\rightarrow \cdot \leftarrow) \cup =$ which is known as WCR1 and holds for left-reduced ground TRSs. We formalized a new, short and direct proof of the following result due to van Oostrom [107]. Here an element a is said to be complete if it is both terminating (there are no infinite rewrite sequences starting at a) and confluent (if $b \xrightarrow{*} a \rightarrow^* c$ then $b \downarrow c$).

Theorem 7.5.7. *Let \mathcal{A} be an ARS with random descent. If $a \leftrightarrow^* b$ with normal form b then a is complete and all rewrite sequences from a to b have the same length.* \checkmark

Proof. Let l (r) be the number of \leftarrow (\rightarrow) steps in the conversion from a to b . We have $l \leq r$ since $n + l = r$ for some n by random descent. First we prove termination of a . For a proof by contradiction, suppose the existence of an infinite rewrite sequence

$$a = a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow \dots$$

Clearly, $a \rightarrow^{r-l} a_{r-l}$ and thus there exists a conversion $a_{r-l} \xrightarrow{*} a \leftrightarrow^* b$ with r backwards and r forwards steps. Hence $a_{r-l} = b$ by another application of random descent and therefore $b \rightarrow a_{r-l+1}$, contradicting the fact that b is a normal form. Next we prove confluence of a . Suppose $c \xrightarrow{*} a \rightarrow^* d$. We obtain the two conversions $c \leftrightarrow^* b$ and $d \leftrightarrow^* b$, which are transformed into $c \downarrow d$ by two applications of random descent. Finally, assume there are two rewrite sequences $a \rightarrow^m b$ and $a \rightarrow^n b$ from a to b of length m and n . Reversing the first sequence and appending the second one yields a conversion $b \leftrightarrow^* b$ with m backwards and n forwards steps. A final application of random descent yields $b \rightarrow^k b$ for some k with $k + m = n$. Since b is a normal form, $k = 0$ and thus $m = n$ as desired. \square

In the series of lemmas below, we establish that reduced ground TRSs are canonical and have random descent.

Lemma 7.5.8. *Left-reduced TRSs enjoy the WCR1 property.* ✓

Proof. This follows from a straightforward case analysis on the relative positions of the two redexes that are part of a peak together with the fact that for left-reduced TRSs the left-hand side alone uniquely determines the employed rewrite rule. □

Lemma 7.5.9. *Left-reduced ground TRSs have random descent.* ✓

Proof. Let \mathcal{R} be a left-reduced TRS and $s \leftrightarrow^* t$ a conversion between two arbitrary but fixed terms s and t such that t is a normal form. We proceed by induction on the length of this conversion. If it is empty or the first step is to the right, we are done. Otherwise, we have $s \leftarrow u \leftrightarrow^* t$ where the conversion between u and t has $l(r) \leftarrow (\rightarrow)$ steps and obtain $u \rightarrow^k t$ with $k + l = r$ by the induction hypothesis. The remainder of the proof proceeds by induction on k together with Lemma 7.5.8. □

Lemma 7.5.10. *Right-reduced ground TRSs are terminating.* ✓

Proof. Let \mathcal{R} be a right-reduced ground TRS. For the sake of a contradiction, assume that \mathcal{R} is non-terminating. Then there is a minimal non-terminating term t (that is, all its proper subterms are terminating). This means that after a finite number of non-root steps $t \rightarrow^* u$ there will be a root step $u \rightarrow v$ such that v is non-terminating. But since \mathcal{R} is right-reduced and ground, v is a ground normal form, deriving the desired contradiction. □

Corollary 7.5.11. *Reduced ground TRSs are canonical and have random descent.* ✓

Proof. Let \mathcal{R} be a reduced ground TRS. Then, by Lemma 7.5.9, \mathcal{R} has random descent. Moreover, by Lemma 7.5.10, \mathcal{R} is terminating. Finally, since all terms are \mathcal{R} -terminating, confluence of \mathcal{R} is an immediate consequence of the definition of random descent. □

Theorem 7.5.12. *For every ground ES \mathcal{E} and every equivalent reduced ground TRS \mathcal{R} there exist a reduction order $>$ and a derivation $\mathcal{E}, \emptyset \vdash_{\mathbf{g}} \dots \vdash_{\mathbf{g}} \emptyset, \mathcal{R}$.* ✓

Proof. Let $>$ be a reduction order that contains \mathcal{R} and is total on \mathcal{E} -equivalent ground terms. Consider a maximal $\text{KB}_{\mathbf{g}}$ run starting from \mathcal{E} and using $>$. According to Theorem 7.5.5, the run produces an equivalent reduced TRS \mathcal{R}' . Since $\mathcal{R} \subseteq >$ and $\mathcal{R}' \subseteq >$, we obtain $\mathcal{R} = \mathcal{R}'$ from Theorem 7.4.11. It remains to show that $>$ exists. Let \sqsupset be a total precedence and define $s > t$ if and only if $s \leftrightarrow_{\mathcal{E}}^* t$ and either $d_{\mathcal{R}}(s) > d_{\mathcal{R}}(t)$ or both $d_{\mathcal{R}}(s) = d_{\mathcal{R}}(t)$ and $s \sqsupset_{\text{lpo}} t$.⁴ Here $d_{\mathcal{R}}(u)$ is the number of rewrite steps in \mathcal{R} to normalize the term u , which is well-defined since all normalizing sequences in a reduced ground TRS have the same length as a consequence of Corollary 7.5.11 and Theorem 7.5.7. It is easy to show that $>$ has the required properties. The only interesting

⁴In the formalization we actually use \sqsupset_{kbo} with all weights set to 1, since in contrast to LPO, for KBO ground-totally for total precedences has already been formalized before.

7. Abstract Completion, Formalized

cases are closure under contexts and substitutions. Both are basically handled by the following observation: $d_{\mathcal{R}}(C[t\sigma]) = d_{\mathcal{R}}(C[t\downarrow\sigma]) + d_{\mathcal{R}}(t)$ for any term t (which holds due to random descent together with termination). This allows us to lift $d_{\mathcal{R}}(s) = d_{\mathcal{R}}(t)$ and $d_{\mathcal{R}}(s) > d_{\mathcal{R}}(t)$ into arbitrary contexts and substitutions. \square

The above result cannot be generalized to left-linear right-ground systems, as shown in the following example due to Dominik Klein (personal communication).

Example 7.5.13. Consider the ES \mathcal{E} consisting of the two equations $\mathbf{f}(x) \approx \mathbf{f}(a)$ and $\mathbf{f}(b) \approx b$. Let $>$ be a reduction order. If $\mathbf{f}(b) > b$ does not hold, no inference rule of KB_g is applicable to (\mathcal{E}, \emptyset) . If $\mathbf{f}(b) > b$ then the second equation can be oriented

$$(\mathcal{E}, \emptyset) \vdash_g (\{\mathbf{f}(x) \approx \mathbf{f}(a)\}, \{\mathbf{f}(b) \rightarrow b\})$$

but no further inference steps of KB_g are possible. Hence completion will fail on \mathcal{E} . Nevertheless, the TRS \mathcal{R} consisting of the rewrite rule $\mathbf{f}(x) \rightarrow b$ constitutes a canonical presentation of \mathcal{E} .

The correctness result of ground completion (Theorem 7.5.5) is due to Snyder [128], and our formalized proof basically follows his approach. In addition, we present a new completeness proof based on random descent (Theorem 7.5.12).

7.6. Correctness for Infinite Runs

Completion as presented in the preceding sections does not always succeed in producing a finite complete presentation. It may fail because an unorientable equation is encountered or it may run forever. In the latter case it is possible that in the limit a possibly infinite complete presentation is obtained. In this case, completion can serve as a semi-decision procedure for the validity problem of the initial equations [62]. In this section we give a new proof that fair infinite runs produce complete presentations of the initial equations, provided the collapse rule is restored to its original formulation (cf. Definition 7.6.2 below).

The reason why this restriction is necessary is provided by the following example (due to Baader and Nipkow [9]), which shows that the correctness result (Theorem 7.3.8) of Section 7.3 does not extend to infinite runs without further ado.

Example 7.6.1. Consider the ES \mathcal{E} consisting of the equations

$$aba \approx ab$$

$$bb \approx b$$

and LPO with precedence $a > b$ as reduction order. After two orient steps, we apply deduce to generate the two critical pairs:

$$aba \rightarrow ab$$

$$bb \rightarrow b$$

$$abab \approx abba$$

$$bb \approx bb$$

The second one is immediately deleted and the first one is simplified:

$$aba \rightarrow ab$$

$$bb \rightarrow b$$

$$abb \approx aba$$

and subsequently oriented:

$$\text{aba} \rightarrow \text{ab} \quad \text{bb} \rightarrow \text{b} \quad \text{aba} \rightarrow \text{abb}$$

At this point we use the third rule to collapse the first rule:

$$\text{abb} \approx \text{ab} \quad \text{bb} \rightarrow \text{b} \quad \text{aba} \rightarrow \text{abb}$$

An application of simplify followed by delete results in:

$$\text{bb} \rightarrow \text{b} \quad \text{aba} \rightarrow \text{abb}$$

Repeating the above process produces

$$\text{bb} \rightarrow \text{b} \quad \text{aba} \rightarrow \text{abbb}$$

and then

$$\text{bb} \rightarrow \text{b} \quad \text{aba} \rightarrow \text{abbbb}$$

ad infinitum. Since none of the rules $\text{aba} \rightarrow \text{ab}^n$ survives, in the limit we obtain the TRS consisting of the single rule $\text{bb} \rightarrow \text{b}$. This TRS is complete but not equivalent to \mathcal{E} as witnessed by non-joinability of aba and ab .

Definition 7.6.2 (Knuth-Bendix Completion). *The inference system KB_i consists of the inference rules deduce, orient, delete, compose, and simplify of KB_f together with the following modified collapse rule:*

$$\text{collapse}_{\triangleright} \quad \frac{\mathcal{E}, \mathcal{R} \uplus \{t \rightarrow s\}}{\mathcal{E} \cup \{u \approx s\}, \mathcal{R}} \quad \text{if } t \xrightarrow{\triangleright}_{\mathcal{R}} u$$

Here the condition $t \xrightarrow{\triangleright}_{\mathcal{R}} u$ is defined as $t \rightarrow u$ using some rule $\ell \rightarrow r \in \mathcal{R}$ such that $t \triangleright \ell$.

Note that the collapse step in Example 7.6.1 does not satisfy the encompassment condition from the previous definition.

We write $(\mathcal{E}, \mathcal{R}) \vdash_i (\mathcal{E}', \mathcal{R}')$ if $(\mathcal{E}', \mathcal{R}')$ can be reached from $(\mathcal{E}, \mathcal{R})$ by employing one of the inference rules of Definition 7.6.2.

Definition 7.6.3. *An infinite run is a maximal sequence of the form*

$$\Gamma: (\mathcal{E}_0, \mathcal{R}_0) \vdash_i (\mathcal{E}_1, \mathcal{R}_1) \vdash_i (\mathcal{E}_2, \mathcal{R}_2) \vdash_i \dots$$

with $\mathcal{R}_0 = \emptyset$. We define

$$\mathcal{E}_\infty = \bigcup_{i \geq 0} \mathcal{E}_i \quad \mathcal{R}_\infty = \bigcup_{i \geq 0} \mathcal{R}_i \quad \mathcal{E}_\omega = \bigcup_{i \geq 0} \bigcap_{j \geq i} \mathcal{E}_j \quad \mathcal{R}_\omega = \bigcup_{i \geq 0} \bigcap_{j \geq i} \mathcal{R}_j$$

Equations in \mathcal{E}_ω and rules in \mathcal{R}_ω are called persistent. The run Γ is called fair if $\mathcal{E}_\omega = \emptyset$ and the inclusion $\text{PCP}(\mathcal{R}_\omega) \subseteq \downarrow_{\mathcal{R}_\omega} \cup \leftrightarrow_{\mathcal{E}_\omega}$ holds.

7. Abstract Completion, Formalized

Bachmair et al. [13] proved that for every fair run satisfying $\mathcal{E}_\omega = \emptyset$ the TRS \mathcal{R}_ω constitutes a complete presentation of \mathcal{E}_0 . The remainder of this section is dedicated to establish the same result, but on a different route without encountering proof orders.

Compared to our proofs for finite runs from Section 7.3, in the following we will disentangle our reasoning about rules from our reasoning about equations and furthermore replace peak decreasingness by the slightly simpler concept of source decreasingness. So why not use this more modular and simpler approach also in our earlier proofs for finite runs? The main difference between the two situations is the encompassment condition of deduce. Unfortunately, without the encompassment condition the equivalent of Lemma 7.6.10 below for finite runs breaks down and we are forced to reason about rules and equations simultaneously (Lemma 7.3.7). Nevertheless, it seems useful to also have a correctness proof for KB_f (lacking the encompassment condition), since out of the four completion tools we are aware of (CiME3 [26], KBCV [159], mkbTT [178], Slothrop [170]), only CiME3 actually implements the encompassment condition.

Lemma 7.6.4. *If $(\mathcal{E}, \mathcal{R}) \vdash_i (\mathcal{E}', \mathcal{R}')$ then the following inclusions hold:*

1. $\mathcal{E}' \cup \mathcal{R}' \subseteq \xleftrightarrow[\mathcal{E} \cup \mathcal{R}]{*}$ ✓
2. $\mathcal{E} \setminus \mathcal{E}' \subseteq (\xrightarrow[\mathcal{R}']{} \cdot \mathcal{E}') \cup (\mathcal{E}' \cdot \xleftarrow[\mathcal{R}']{}) \cup \mathcal{R}' \cup \mathcal{R}'^{-1} \cup =$ ✓
3. $\mathcal{R} \setminus \mathcal{R}' \subseteq (\xrightarrow[\mathcal{R}']{} \cdot \mathcal{E}') \cup (\mathcal{R}' \cdot \xleftarrow[\mathcal{R}']{})$ ✓

Together these properties reveal that inference steps do not change the conversion relation.

Corollary 7.6.5. *If $(\mathcal{E}, \mathcal{R}) \vdash_i^* (\mathcal{E}', \mathcal{R}')$ then the relations $\xleftrightarrow[\mathcal{E} \cup \mathcal{R}]{*}$ and $\xleftrightarrow[\mathcal{E}' \cup \mathcal{R}']{*}$ coincide.* ✓

Below, we consider an infinite run $\Gamma: (\mathcal{E}_0, \mathcal{R}_0) \vdash_i (\mathcal{E}_1, \mathcal{R}_1) \vdash_i (\mathcal{E}_2, \mathcal{R}_2) \vdash_i \dots$ such that $\mathcal{E}_\omega = \emptyset$. First we show that all rewrite rules are compatible with the reduction order $>$.

Lemma 7.6.6. *The inclusions $\mathcal{R}_\omega \subseteq \mathcal{R}_\infty \subseteq >$ hold.* ✓✓

Next, we verify that every equality in \mathcal{E}_i can be turned into a valley in \mathcal{R}_∞ . Note that in contrast to the proof order approach [13] and to the correctness proof for finite runs given in Section 7.3 we reason separately about equations and rules. This more local rationale simplifies the analysis as we can use different well-founded induction arguments for the two cases, rather than synthesizing an order that covers both.

Lemma 7.6.7. *The inclusion $\mathcal{E}_\infty \subseteq \downarrow_{\mathcal{R}_\infty}$ holds.* ✓

Proof. Let $s \approx t \in \mathcal{E}_i$ for some $i \geq 0$. By induction on $\{s, t\}$ with respect to $>_{\text{mul}}$ we show $s \downarrow_{\mathcal{R}_\infty} t$. Because $\mathcal{E}_\omega = \emptyset$, $s \approx t \in \mathcal{E}_{j-1} \setminus \mathcal{E}_j$ for some $j > i$. Following Lemma 7.6.4(2), we distinguish three cases.

- If $s \approx t \in \mathcal{R}_j \cup \mathcal{R}_j^{-1} \cup =$ then the claim trivially holds.

- If $s \rightarrow_{\mathcal{R}_j} u$ and $u \approx t \in \mathcal{E}_j$ for some term u then $\{s, t\} >_{\text{mul}} \{u, t\}$ and thus $u \downarrow_{\mathcal{R}_\infty} t$ by the induction hypothesis. Hence also $s \downarrow_{\mathcal{R}_\infty} t$.
- Similarly, if $s \approx u \in \mathcal{E}_j$ and $u \mathcal{R}_j \leftarrow t$ for some term u then $\{s, t\} >_{\text{mul}} \{s, u\}$ and we obtain $s \downarrow_{\mathcal{R}_\infty} t$ as in the preceding case. \square

Corollary 7.6.8. *The inclusion $\xrightarrow{\mathcal{E}_\infty} \subseteq \xleftarrow{\mathcal{R}_\infty}^*$ holds.* \checkmark

In order to show confluence of \mathcal{R}_ω we use source decreasingness as defined in Section 7.2, employing the following extension of the reduction order $>$.

Definition 7.6.9. *We define $\succ = ((> \cup \triangleright) / \triangleright)^+$.*

According to Lemma 7.2.3, \succ is a well-founded order. The next lemma allows us to transform every non-persistent rule $\ell \rightarrow r$ into an \mathcal{R}_ω -conversion below ℓ .

Lemma 7.6.10. *The inclusion $\xrightarrow{\mathcal{R}_\infty}^s \subseteq \xleftarrow{\mathcal{R}_\omega}^{s*}$ holds for all terms s .* \checkmark

Proof. Let $s \xrightarrow{\mathcal{R}_\infty}^s t$ by employing the rewrite rule $\ell \rightarrow r$. We prove $s \xleftarrow{\mathcal{R}_\omega}^{s*} t$ by induction on (ℓ, r) with respect to \succ_{lex} . If $\ell \rightarrow r \in \mathcal{R}_\omega$ then the claim trivially holds. Otherwise, $\ell \rightarrow r \in \mathcal{R}_{i-1} \setminus \mathcal{R}_i$ for some $i > 0$. Using Lemma 7.6.4(3), we distinguish two cases.

- Suppose $\ell \xrightarrow{\triangleright} \ell' \rightarrow r'$ and $u \approx r \in \mathcal{E}_i$ for some term u and rule $\ell' \rightarrow r' \in \mathcal{R}_i$. We obtain $\ell \xrightarrow{\triangleright} \ell' \rightarrow r' u \downarrow_{\mathcal{R}_\infty} r$ from Lemma 7.6.7. We have $\ell \triangleright \ell'$ and both $\ell > u$ and $\ell > r$. It follows that all rewrite rules $\ell'' \rightarrow r''$ employed in $\ell \xrightarrow{\triangleright} u \downarrow_{\mathcal{R}_\infty} r$ satisfy $(\ell, r) \succ_{\text{lex}} (\ell'', r'')$. Moreover, all steps in $\ell \downarrow_{\mathcal{R}_\infty} r$ are labeled with a term $\leq \ell$. Hence we obtain $\ell \xleftarrow{\mathcal{R}_\omega}^{s*} r$ from the induction hypothesis.
- Suppose $\ell \rightarrow u \in \mathcal{R}_i$ and $u \leftarrow \ell' \rightarrow r'$ for some term u and rewrite rule $\ell' \rightarrow r' \in \mathcal{R}_i$. We have $(\ell, r) \succ_{\text{lex}} (\ell, u)$ and $(\ell, r) \succ_{\text{lex}} (\ell', r')$ because $r > u$ and $\ell > r \triangleright \ell' > r'$. Moreover, both steps are labeled with a term $\leq \ell$ and thus we obtain $\ell \xleftarrow{\mathcal{R}_\omega}^{s*} r$ from the induction hypothesis.

So in both cases we have $\ell \xleftarrow{\mathcal{R}_\omega}^{s*} r$ and thus also $s \xleftarrow{\mathcal{R}_\omega}^{s*} t$. \square

Corollary 7.6.11. *The relations $\xleftarrow{\mathcal{R}_\infty}^*$ and $\xleftarrow{\mathcal{R}_\omega}^*$ coincide.* \checkmark

We arrive at the main theorem of this section. Note that Bachmair's correctness proof [10] uses induction with respect to a well-founded order on conversions to directly show that any conversion of $\mathcal{E}_\infty \cup \mathcal{R}_\infty$ can be transformed into a joining sequence of \mathcal{R}_ω . In contrast, we prove confluence via source decreasingness. This allows us to concentrate on *local* peaks.

Theorem 7.6.12. *If Γ is fair then \mathcal{R}_ω is a complete presentation of \mathcal{E}_0 .* \checkmark

7. Abstract Completion, Formalized

Proof. We have $\mathcal{E}_\omega = \emptyset$ because Γ is non-failing. The TRS \mathcal{R}_ω is terminating by Lemma 7.6.6. We show source decreasingness of labeled \mathcal{R}_ω reduction with respect to the reduction order $>$. So let $t \xrightarrow{\mathcal{R}_\omega}^s s \xrightarrow{\mathcal{R}_\omega} u$. From Lemma 7.2.16 we obtain $t \nabla_s^2 u$. Let $v \nabla_s w$ appear in this sequence (so $t = v$ or $w = u$). We have $s > v$, $s > w$, and $(v, w) \in \downarrow_{\mathcal{R}_\omega} \cup \leftrightarrow_{\mathcal{E}_\infty}$ by the definition of ∇_s and fairness of Γ .

- If $v \downarrow_{\mathcal{R}_\omega} w$ then $v \xrightarrow{\mathbb{W}v}_{\mathcal{R}_\omega}^* \cdot \mathcal{R}_\omega^* \xrightarrow{\mathbb{W}w} w$ and thus $v \xrightarrow{\nabla_s}_{\mathcal{R}_\omega}^* w$.
- If $v \leftrightarrow_{\mathcal{E}_\infty} w$ then $v \leftrightarrow_{\mathcal{E}_i} w$ for some $i \geq 0$ then $v \downarrow_{\mathcal{R}_{\nabla_s^i}} w$ by Lemma 7.6.7. We obtain $v \xrightarrow{\nabla_s}_{\mathcal{R}_\infty}^* w$ as in the previous case and thus $v \xrightarrow{\nabla_s}_{\mathcal{R}_\omega}^* w$ by Lemma 7.6.10.

Hence $t \xrightarrow{\nabla_s}_{\mathcal{R}_\omega}^* u$. Confluence of \mathcal{R}_ω now follows from Lemmata 7.2.9 and 7.2.5. It remains to show $\leftrightarrow_{\mathcal{E}_0}^* = \leftrightarrow_{\mathcal{R}_\omega}^*$. Using Corollary 7.6.5 we obtain $\rightarrow_{\mathcal{E}_i \cup \mathcal{R}_i} \subseteq \leftrightarrow_{\mathcal{E}_0}^*$ by a straightforward induction on i . This in turn yields $\leftrightarrow_{\mathcal{E}_0}^* = \leftrightarrow_{\mathcal{E}_\infty \cup \mathcal{R}_\infty}^*$. From Corollary 7.6.8 we infer $\leftrightarrow_{\mathcal{E}_\infty \cup \mathcal{R}_\infty}^* = \leftrightarrow_{\mathcal{R}_\infty}^*$ and we conclude by an appeal to Corollary 7.6.11. \square

Example 7.6.13. Consider the ES \mathcal{E} and the KBO $>$ from Example 7.1.1. Let \mathcal{P}_n for $n \geq 1$ denote the TRS $\{\mathbf{ab}^{i+1}\mathbf{ab} \rightarrow \mathbf{babba}^i \mid 1 \leq i \leq n\}$. One possible infinite completion run is the following:

$$\begin{aligned} (\mathcal{E}, \emptyset) &\vdash_i^{\text{orient}} (\emptyset, \{\mathbf{aba} \rightarrow \mathbf{bab}\}) && \vdash_i^{\text{deduce}} (\{\mathbf{abbab} \approx \mathbf{babba}\}, \{\mathbf{aba} \rightarrow \mathbf{bab}\}) \\ &\vdash_i^{\text{orient}} (\emptyset, \{\mathbf{aba} \rightarrow \mathbf{bab}\} \cup \mathcal{P}_1) && \vdash_i^{\text{deduce}} (\{\mathbf{abbbab} \approx \mathbf{babbaa}\}, \{\mathbf{aba} \rightarrow \mathbf{bab}\} \cup \mathcal{P}_1) \\ &\vdash_i^{\text{orient}} (\emptyset, \{\mathbf{aba} \rightarrow \mathbf{bab}\} \cup \mathcal{P}_2) && \vdash_i \dots \end{aligned}$$

If this run is continued in a fair way we subsequently construct the TRSs \mathcal{P}_n and can in the limit obtain the result $\mathcal{R}_\omega = \{\mathbf{aba} \rightarrow \mathbf{bab}\} \cup \{\mathbf{ab}^{i+1}\mathbf{ab} \rightarrow \mathbf{babba}^i \mid i \geq 1\}$, which is complete according to Theorem 7.6.12.

This section recapitulates our results on infinite runs [59]. Our correctness proof (Theorem 7.6.12) differs substantially from earlier proofs in the literature. Due to a less monolithic structure we consider this proof to be more formalization friendly: Instead of lexicographically combining several orders into a single proof reduction relation, we use source decreasingness together with different orders as necessary to prove auxiliary results. In particular, our approach naturally supports prime critical pairs.

7.7. Ordered Completion

Completion may fail to construct a complete system if unorientable equations are encountered. For example, the ES \mathcal{E} consisting of the two equations $\mathbf{0} + \mathbf{x} \approx \mathbf{x}$ and $\mathbf{x} + \mathbf{y} \approx \mathbf{y} + \mathbf{x}$ admits no complete presentation. (We will prove it in Section 7.8.) This can happen even if a finite complete system exists, as illustrated by the following example.

Example 7.7.1. Consider the ES \mathcal{E} [12] consisting of the three equations

$$\mathbf{1} \cdot (-\mathbf{x} + \mathbf{x}) \approx \mathbf{0} \qquad \mathbf{1} \cdot (\mathbf{x} + -\mathbf{x}) \approx \mathbf{x} + -\mathbf{x} \qquad -\mathbf{x} + \mathbf{x} \approx \mathbf{y} + -\mathbf{y}$$

7.7. Ordered Completion

Any run of standard Knuth-Bendix completion will fail on this input system; the first two equations may be oriented from left to right if a suitable order is employed but no further steps are possible. However, the TRS \mathcal{R} consisting of the rules


$$1 \cdot 0 \rightarrow 0 \qquad x + -x \rightarrow 0 \qquad -x + x \rightarrow 0$$

constitutes a canonical presentation of \mathcal{E} .

Ordered completion was developed to remedy this shortcoming. In contrast to completion as presented in the preceding section it never fails, though the resulting system is in general only ground complete.

For an ES \mathcal{E} , an ordered rewrite step is a rewrite step using a rule from $\mathcal{E}^>$, which is the infinite set of rewrite rules $\ell\sigma \rightarrow r\sigma$ such that $\ell \approx r \in \mathcal{E}^\pm$ and $\ell\sigma > r\sigma$ for some substitution σ .

The following inference rules for ordered completion are due to Bachmair, Dershowitz, and Plaisted [14]. In order to simplify the notation, we abbreviate $\mathcal{E}_\omega^> \cup \mathcal{R}_\omega$ to \mathcal{S} , and use the following shorthands. We write $t \xrightarrow{\mathbb{P}}_{\mathcal{E}^>} u$ if there exist an equation $\ell \approx r \in \mathcal{E}^\pm$, a context C , and a substitution σ such that $t = C[\ell\sigma]$, $u = C[r\sigma]$, $\ell\sigma > r\sigma$, and $t \triangleright \ell$. The union of $\rightarrow_{\mathcal{R}}$ and $\xrightarrow{\mathbb{P}}_{\mathcal{E}^>}$ is denoted by $\xrightarrow{\mathbb{P}_1}_{\mathcal{S}}$ and we write $\xrightarrow{\mathbb{P}_2}_{\mathcal{S}}$ for the union of $\xrightarrow{\mathbb{P}_1}_{\mathcal{S}}$ and $\xrightarrow{\mathbb{P}}_{\mathcal{E}^>}$.

Definition 7.7.2 (Ordered Completion ). *The inference system KB_\circ of ordered completion operates on pairs $(\mathcal{E}, \mathcal{R})$ of equations \mathcal{E} and rules \mathcal{R} over a common signature \mathcal{F} . It consists of the following inference rules:*

$$\begin{array}{ll}
\text{deduce} \quad \frac{\mathcal{E}, \mathcal{R}}{\mathcal{E} \cup \{s \approx t\}, \mathcal{R}} & \text{if } s \xleftarrow{\mathcal{R} \cup \mathcal{E}^\pm} \cdot \xrightarrow{\mathcal{R} \cup \mathcal{E}^\pm} t \quad \text{compose} \quad \frac{\mathcal{E}, \mathcal{R} \uplus \{s \rightarrow t\}}{\mathcal{E}, \mathcal{R} \cup \{s \rightarrow u\}} \quad \text{if } t \rightarrow_{\mathcal{S}} u \\
\text{orient} \quad \frac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E}, \mathcal{R} \cup \{s \rightarrow t\}} & \text{if } s > t \quad \text{simplify} \quad \frac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E} \cup \{u \approx t\}, \mathcal{R}} \quad \text{if } s \xrightarrow{\mathbb{P}_1}_{\mathcal{S}} u \\
& \text{if } t > s \quad \frac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E}, \mathcal{R} \cup \{t \rightarrow s\}} \quad \text{if } t \xrightarrow{\mathbb{P}_1}_{\mathcal{S}} u \\
\text{delete} \quad \frac{\mathcal{E} \uplus \{s \approx s\}, \mathcal{R}}{\mathcal{E}, \mathcal{R}} & \text{collapse} \quad \frac{\mathcal{E}, \mathcal{R} \uplus \{t \rightarrow s\}}{\mathcal{E} \cup \{u \approx s\}, \mathcal{R}} \quad \text{if } t \xrightarrow{\mathbb{P}_2}_{\mathcal{S}} u
\end{array}$$

The deduce rule may be applied to any peak, though in practice it is typically limited to the addition of extended critical pairs (which are defined in Definition 7.7.11 below). We write $(\mathcal{E}, \mathcal{R}) \vdash_\circ (\mathcal{E}', \mathcal{R}')$ if $(\mathcal{E}', \mathcal{R}')$ can be reached from $(\mathcal{E}, \mathcal{R})$ by employing one of the inference rules of Definition 7.7.2. We start by stating the equivalents of Lemma 7.6.4 and Corollary 7.6.5 for ordered completion.

Lemma 7.7.3. *If $(\mathcal{E}, \mathcal{R}) \vdash_\circ (\mathcal{E}', \mathcal{R}')$ then the following inclusions hold:*

$$1. \mathcal{E}' \cup \mathcal{R}' \subseteq \xleftarrow[\mathcal{E} \cup \mathcal{R}]{*} \quad \quad \quad \checkmark$$

7. Abstract Completion, Formalized

$$2. \mathcal{E} \setminus \mathcal{E}' \subseteq (\xrightarrow[S']{\mathbb{P}_1} \cdot \mathcal{E}'^\pm)^\pm \cup \mathcal{R}'^\pm \cup =$$

✓

$$3. \mathcal{R} \setminus \mathcal{R}' \subseteq (\xrightarrow[S']{\mathbb{P}_2} \cdot \mathcal{E}') \cup (\mathcal{R}' \cdot \xleftarrow[S']{ })$$

✓

Corollary 7.7.4. *If $(\mathcal{E}, \mathcal{R}) \vdash_o (\mathcal{E}', \mathcal{R}')$ then the relations $\xleftarrow[\mathcal{E} \cup \mathcal{R}]{}^*$ and $\xleftarrow[\mathcal{E}' \cup \mathcal{R}']{}^*$ coincide.* ✓

We illustrate KB_o by means of an example.

Example 7.7.5. *Consider the ES \mathcal{E} consisting of the following three equations:*

$$f(x) \approx f(a) \qquad f(b) \approx b \qquad g(f(b), x) \approx g(x, b)$$

By taking the Knuth-Bendix order $>_{\text{kbo}}$ with precedence $f > b$ and where all function symbols are assigned weight 1, the following KB_o inference sequence can be obtained:

$$\begin{aligned} (\mathcal{E}, \emptyset) &\vdash_o^{\text{orient}^+} (\{f(x) \approx f(a)\}, \{f(b) \rightarrow b, g(f(b), x) \rightarrow g(x, b)\}) \\ &\vdash_o^{\text{deduce}} (\{f(x) \approx f(a), f(b) \approx f(a)\}, \{f(b) \rightarrow b, g(f(b), x) \rightarrow g(x, b)\}) \\ &\vdash_o^{\text{simplify}} (\{f(x) \approx f(a), b \approx f(a)\}, \{f(b) \rightarrow b, g(f(b), x) \rightarrow g(x, b)\}) \\ &\vdash_o^{\text{orient}} (\{f(x) \approx f(a)\}, \{f(b) \rightarrow b, g(f(b), x) \rightarrow g(x, b), f(a) \rightarrow b\}) \\ &\vdash_o^{\text{simplify}} (\{f(x) \approx b\}, \{f(b) \rightarrow b, g(f(b), x) \rightarrow g(x, b), f(a) \rightarrow b\}) \\ &\vdash_o^{\text{collapse}^+} (\{f(x) \approx b, b \approx b, g(b, x) \approx g(x, b)\}, \emptyset) \\ &\vdash_o^{\text{orient}} (\{b \approx b, g(b, x) \approx g(x, b)\}, \{f(x) \rightarrow b\}) \\ &\vdash_o^{\text{delete}} (\{g(b, x) \approx g(x, b)\}, \{f(x) \rightarrow b\}) \\ &\vdash_o^{\text{deduce}} (\{g(b, x) \approx g(x, b), b \approx b\}, \{f(x) \rightarrow b\}) \end{aligned}$$

This sequence can be extended to an (infinite) run by repeating the last two steps. Then we have $\mathcal{R}_\omega = \{f(x) \rightarrow b\}$ and $\mathcal{E}_\omega = \{g(b, x) \approx g(x, b)\}$.

Below, we consider an arbitrary run $\Gamma: (\mathcal{E}_0, \mathcal{R}_0) \vdash_o (\mathcal{E}_1, \mathcal{R}_1) \vdash_o (\mathcal{E}_2, \mathcal{R}_2) \vdash_o \dots$. In general $\mathcal{E}_\infty \subseteq \downarrow_{\mathcal{R}_\infty}$ does not hold, as Example 7.7.5 illustrates. So unlike in the preceding section we now omit the condition $\mathcal{E}_\omega = \emptyset$. However, this comes at the price of weaker properties of the resulting system, as the remainder of this section shows.

Lemma 7.7.6. *The inclusions $\mathcal{R}_\omega \subseteq \mathcal{R}_\infty \subseteq >$ and $\mathcal{E}_\omega \subseteq \mathcal{E}_\infty$ hold.* ✓✓✓

We use the relation \xrightarrow{M} from Definition 7.3.6 to show that any equation step below a term set M eventually turns into a conversion over $\mathcal{E}_\omega \cup \mathcal{R}_\infty$ that is still below M . Note that just like in Section 7.6 we avoid the use of a synthesized termination argument by handling equations and rules separately.

Lemma 7.7.7. *The inclusion $\xrightarrow[\mathcal{E}_\infty]{S} \subseteq \xleftarrow[\mathcal{E}_\omega \cup \mathcal{R}_\infty]{S}^*$ holds for all sets S of terms.* ✓

Proof. Let $t \approx u \in \mathcal{E}_\infty$. We prove

$$\frac{M}{t \approx u} \subseteq \frac{M}{\mathcal{E}_\omega \cup \mathcal{R}_\infty}^*$$

by induction on $\{t, u\}$ with respect to the well-founded order \succ_{mul} . If $t \approx u \in \mathcal{E}_\omega^\pm$ then the claim follows trivially. Otherwise, $t \approx u \in (\mathcal{E}_{i-1} \setminus \mathcal{E}_i)^\pm$ for some $i > 0$. Using Lemma 7.7.3(2), we distinguish two subcases.

- Suppose $t \approx u \in (\xrightarrow{\mathbb{P}_1}_{\mathcal{S}_i} \cdot \mathcal{E}_i^\pm)^\pm$. There exist a term t' and an equation $v' \approx u' \in \mathcal{E}_i^\pm$ such that $\{t, u\} = \{t', u'\}$ and $t' \xrightarrow{\mathbb{P}_1}_{\mathcal{S}_i} v'$. It is sufficient to show

$$t' \xleftarrow[\mathcal{E}_\omega \cup \mathcal{R}_\infty]{\{t', u'\}^*} v' \quad \text{and} \quad v' \xleftarrow[\mathcal{E}_\omega \cup \mathcal{R}_\infty]{\{t', u'\}^*} u'$$

The second conversion follows from $t' > v'$ and the induction hypothesis for $v' \approx u' \in \mathcal{E}_i^\pm$, which is applicable as $\{t, u\} = \{t', u'\} \succ_{\text{mul}} \{v', u'\}$. The first conversion is obtained as follows. Because of $t' \xrightarrow{\mathbb{P}_1}_{\mathcal{S}_i} v'$, we have $t' \rightarrow_{\mathcal{R}_i} v'$ or $t' \xrightarrow{\mathbb{P}_i}_{\mathcal{E}_i^>} v'$. If $t' \rightarrow_{\mathcal{R}_i} v'$ then this step can be labeled with $\{t', u'\}$ as $t' > v'$. Otherwise, there exist an equation $\ell \approx r \in \mathcal{E}_i^\pm$, a context C , and a substitution σ such that $t' = C[\ell\sigma]$, $v' = C[r\sigma]$, $\ell\sigma > r\sigma$, and $t' \triangleright \ell$. We have $t' \succ \ell$ and $t' \succ r$ as $t' \triangleright \ell\sigma > r\sigma \triangleright r$. Therefore $\{t', u'\} \succ_{\text{mul}} \{\ell, r\}$ holds, so

$$\ell \xleftarrow[\mathcal{E}_\omega \cup \mathcal{R}_\infty]{\{\ell, r\}^*} r$$

follows from the induction hypothesis. Closure under contexts and substitutions now yields $t \xleftarrow[\mathcal{E}_\omega \cup \mathcal{R}_\infty]{\{t, u\}^*} u$.

- If $t \approx u \in \mathcal{R}_i^\pm \cup =$ then $t \xleftarrow[\mathcal{R}_\infty]{\{t, u\}^*} u$.

In both cases $t \xleftarrow[\mathcal{E}_\omega \cup \mathcal{R}_\infty]{\{t, u\}^*} u$ holds. Since M contains upper bounds of t and u with respect to \geq , the desired inclusion follows from the closure under contexts and substitutions of $\rightarrow_{\mathcal{E}_\omega \cup \mathcal{R}_\infty}$ and \geq . \square

Next, we show that a rewrite step that uses a rule in \mathcal{R}_∞ and is below a multiset of terms M eventually turns into a conversion over persistent rules and equations that is still below M . To this end we write γt for the set $\{u \in \mathcal{T}(\mathcal{F}, \mathcal{V}) \mid t \succ u\}$.

Lemma 7.7.8. *The inclusion $\frac{M}{\mathcal{R}_\infty} \subseteq \frac{M}{\mathcal{E}_\omega \cup \mathcal{R}_\omega}^*$ holds for all multisets M of terms.* \square

Proof. Let $\ell \approx r \in \mathcal{R}_\infty$. We prove

$$\frac{M}{\ell \rightarrow r} \subseteq \frac{M}{\mathcal{E}_\omega \cup \mathcal{R}_\omega}^*$$

by induction on (ℓ, r) with respect to the well-founded order \succ_{lex} . If $\ell \rightarrow r \in \mathcal{R}_\omega$ then the claim trivially holds. Otherwise, there is some $i > 0$ such that $\ell \rightarrow r \in \mathcal{R}_{i-1} \setminus \mathcal{R}_i$. From Lemma 7.7.7 and the induction hypothesis the inclusions

$$\frac{N}{\mathcal{E}_\infty \cup \mathcal{R}_\infty} \subseteq \frac{N}{\mathcal{E}_\omega \cup \mathcal{R}_\infty}^* \subseteq \frac{N}{\mathcal{E}_\omega \cup \mathcal{R}_\omega}^* \quad (7.1)$$

are obtained for every set $N \subseteq \gamma \ell$. Using Lemma 7.7.3, we distinguish two cases.

7. Abstract Completion, Formalized

- Suppose $\ell \xrightarrow{\triangleright^2}_{\mathcal{S}_i} u$ and $u \approx r \in \mathcal{E}_i$ for some term u . There exist an equation $\ell' \approx r' \in \mathcal{E}_\infty^\pm \cup \mathcal{R}_\infty$, a context C and a substitution σ such that $\ell = C[\ell'\sigma]$, $u = C[r'\sigma]$, $\ell\sigma > r\sigma$, and $\ell \triangleright \ell'$. We have $\ell \succ \ell', r'$ as $\ell \triangleright \ell'$ and $\ell \triangleright \ell'\sigma > r'\sigma \triangleright r'$ and thus

$$\ell' \xleftarrow[\mathcal{E}_\infty \cup \mathcal{R}_\infty]{\{\ell'\}} r'$$

Since $\{\ell', r'\} \subseteq \nabla \ell$ we obtain $\ell' \xleftarrow[\mathcal{E}_\omega \cup \mathcal{R}_\omega]{\{\ell', r'\}^*} r'$ from (7.1). Therefore, $\ell \xleftarrow[\mathcal{E}_\omega \cup \mathcal{R}_\omega]{\{\ell\}^*} u$ follows from closure under contexts and substitutions and $\ell > u$. Again from $\ell > u, r$ we obtain $u \xleftarrow[\mathcal{E}_\infty \cup \mathcal{R}_\infty]{\nabla \ell} r$ and thus $u \xleftarrow[\mathcal{E}_\omega \cup \mathcal{R}_\omega]{\nabla \ell} r$ follows from (7.1).

- Suppose $\ell \rightarrow u \in \mathcal{R}_i$ and $u \mathcal{S}_i \leftarrow r$ for some term u . We have $r > u$ and thus $(\ell, r) \succ_{\text{lex}} (\ell, u)$. Hence we can apply the induction hypothesis to $\ell \xrightarrow[\ell \rightarrow u]{\{\ell\}} u$, yielding $\ell \xleftarrow[\mathcal{E}_\omega \cup \mathcal{R}_\omega]{\{\ell\}^*} u$. From $\ell > r > u$ we obtain $u \xleftarrow[\mathcal{E}_\infty \cup \mathcal{R}_\infty]{\nabla \ell} r$ and thus $u \xleftarrow[\mathcal{E}_\omega \cup \mathcal{R}_\omega]{\nabla \ell} r$ follows by (7.1).

In both cases $\ell \xleftarrow[\mathcal{E}_\omega \cup \mathcal{R}_\omega]{\{\ell\}^*} r$ holds. Since $\rightarrow_{\mathcal{E}_\omega \cup \mathcal{R}_\omega}$ and \triangleright are closed under contexts and substitutions, the desired inclusion on steps using $\ell \rightarrow r$ follows. \square

We can combine the preceding lemmata to obtain an inclusion in conversions over persistent equations and rules.

Corollary 7.7.9. *The inclusion $\frac{M}{\mathcal{E}_\infty \cup \mathcal{R}_\infty} \subseteq \frac{M}{\mathcal{E}_\omega \cup \mathcal{R}_\omega}^*$ holds for all multisets of terms M .* \checkmark

For instance, in Example 7.7.5 we have $\mathbf{f}(x) \xleftarrow[\mathcal{E}_\infty]{M} \mathbf{f}(a)$ for $M = \{\mathbf{f}(x), \mathbf{f}(a)\}$ and the conversion $\mathbf{f}(x) \leftrightarrow \mathbf{b} \leftrightarrow \mathbf{f}(a)$ in $\mathcal{E}_\omega \cup \mathcal{R}_\omega$ clearly satisfies $\mathbf{f}(x) \xleftarrow[\mathcal{E}_\omega \cup \mathcal{R}_\omega]{M^*} \mathbf{f}(a)$.

The results obtained so far are sufficient to show that ordered completion can produce a complete system.


Theorem 7.7.10. *If Γ satisfies $\text{PCP}(\mathcal{R}_\omega) \subseteq \downarrow_{\mathcal{R}_\omega} \cup \leftrightarrow_{\mathcal{E}_\infty}$ and $\mathcal{E}_\omega = \emptyset$ then \mathcal{R}_ω is a complete presentation of \mathcal{E}_0 .* \checkmark

Proof. We prove that \mathcal{R}_ω is confluent by showing that labeled \mathcal{R}_ω reduction on arbitrary terms is source decreasing. Consider $t \mathcal{R}_\omega \xleftarrow{s} s \xrightarrow{s}_{\mathcal{R}_\omega} u$. From Lemma 7.2.16 we obtain $t \nabla_s^2 u$ (where \mathcal{R}_ω takes the place of \mathcal{R} in the definition of ∇_s). Let $v \nabla_s w$ appear in this sequence (so $t = v$ or $w = u$). We have $s > v$, $s > w$, and $v \downarrow_{\mathcal{R}_\omega} w$ or $v \leftrightarrow_{\mathcal{E}_\infty} w$ by the definition of ∇_s and the assumption $\text{PCP}(\mathcal{R}_\omega) \subseteq \leftrightarrow_{\mathcal{E}_\infty}$.

- If $v \downarrow_{\mathcal{R}_\omega} w$ then $v \xrightarrow[\mathcal{R}_\omega]{\nabla v} \cdot \mathcal{R}_\omega \xleftarrow[\mathcal{R}_\omega]{\nabla w} w$ and thus $v \xleftarrow[\mathcal{R}_\omega]{\nabla s} w$.
- If $v \leftrightarrow_{\mathcal{E}_\infty} w$ then $v \xleftarrow[\mathcal{R}_\omega]{\nabla s} w$ by Corollary 7.7.9.

Hence $t \xleftarrow[\mathcal{R}_\omega]{\nabla s} u$. Confluence of \mathcal{R}_ω follows from Lemmata 7.2.9 and 7.2.5. Termination of \mathcal{R}_ω holds by Lemma 7.7.6. We have $\leftrightarrow_{\mathcal{E}_0}^* = \leftrightarrow_{\mathcal{R}_\omega}^*$ by an easy induction argument using Corollary 7.7.4, so \mathcal{R}_ω is a complete presentation of \mathcal{E}_0 . \square

From now on we specialize our results to ground terms. In the remainder of this section we therefore assume that $>$ is a ground-total reduction order. Before continuing with results on ordered completion, we define extended critical pairs.


Definition 7.7.11 (Extended Overlaps ) . An extended overlap of a given ES \mathcal{E} is a triple $\langle \ell_1 \approx r_1, p, \ell_2 \approx r_2 \rangle$ satisfying the following properties:

- there are renamings π_1 and π_2 such that $\pi_1(\ell_1 \approx r_1), \pi_2(\ell_2 \approx r_2) \in \mathcal{E}^\pm$ (that is, the equations are variants of equations in \mathcal{E}^\pm),
- $\text{Var}(\ell_1 \approx r_1) \cap \text{Var}(\ell_2 \approx r_2) = \emptyset$,
- $p \in \text{Pos}_{\mathcal{F}}(\ell_2)$,
- ℓ_1 and $\ell_2|_p$ are unifiable with some mgu μ , and
- $r_1\mu \not\approx \ell_1\mu$ and $r_2\mu \not\approx \ell_2\mu$.

An extended overlap gives rise to the extended critical pair $\ell_2[r_1]_p\mu \approx r_2\mu$. An extended critical pair is called *prime* if all proper subterms of $\ell_1\mu$ are $\mathcal{E}^>$ -normal forms. The set of extended prime critical pairs among equations in \mathcal{E} is denoted by $\text{PCP}_{>}(\mathcal{E})$.

For example, the equations $1 \cdot (x + -x) \approx x + -x$ and $y + -y \approx -z + z$ are variable-disjoint variants of equations in Example 7.7.1. Neither of them can be oriented from right to left (independent of the choice of $>$). Because of the peak $1 \cdot (-z + z) \leftrightarrow 1 \cdot (x + -x) \leftrightarrow x + -x$ they admit the extended overlap $\langle y + -y \approx -z + z, 1, 1 \cdot (x + -x) \approx x + -x \rangle$ which gives rise to the extended critical pair $1 \cdot (-z + z) \approx x + -x$. Note that since the second equation is unorientable, a run of a standard completion procedure will not encounter this critical pair.


Extended critical pairs are important due to the Extended Critical Pair Lemma [14], according to which these are the only peaks relevant for ground confluence. In our formalization we use the following variant. The proof employs a similar peak analysis as in Lemma 7.2.12.


Lemma 7.7.12. Let \mathcal{E} be an ES and consider a peak 

$$t \xleftarrow[r_1 \approx \ell_1]{pq, \sigma_1} s \xrightarrow[\ell_2 \approx r_2]{p, \sigma_2} u$$

involving ground terms s , t , and u such that $\ell_1\sigma_1 > r_1\sigma_1$ and $\ell_2\sigma_2 > r_2\sigma_2$. If $\ell_1 \approx r_1, \ell_2 \approx r_2 \in \mathcal{E}$ do not form an extended overlap at position q then $t \downarrow_{\mathcal{E}} u$.

In the sequel, we write \mathcal{S}_ω for the TRS $\mathcal{E}_\omega^> \cup \mathcal{R}_\omega$.

Corollary 7.7.13. If $s \xleftrightarrow{\mathcal{E}_\omega^>} t$ for ground terms s and t then $s \xleftrightarrow[\mathcal{S}_\omega]{\{s, t\}}^* t$. 

Proof. We obtain $s \xleftrightarrow[\mathcal{E}_\omega^>]{\{s, t\}}^* t$ from Corollary 7.7.9. Since $>$ is ground-total, all \mathcal{E}_ω steps in this conversion are $(\mathcal{E}_\omega^>)^\pm$ steps or trivial steps between identical terms. Hence $s \xleftrightarrow[\mathcal{S}_\omega]{\{s, t\}}^* t$ as desired. 

7. Abstract Completion, Formalized

Definition 7.7.14. A run $(\mathcal{E}_0, \mathcal{R}_0) \vdash_o (\mathcal{E}_1, \mathcal{R}_1) \vdash_o (\mathcal{E}_2, \mathcal{R}_2) \vdash_o \dots$ is called fair if the inclusion $\text{PCP}_{>}(\mathcal{E}_\omega \cup \mathcal{R}_\omega) \subseteq \downarrow_{\mathcal{S}_\omega} \cup \leftrightarrow_{\mathcal{E}_\infty}$ holds.

The following lemma links extended prime critical pairs to standard critical pairs and hence allows us to reuse results from Section 7.2.3 for our main correctness result (Theorem 7.7.16 below).

Lemma 7.7.15. For a TRS \mathcal{R} and an ES \mathcal{E} , the inclusion $\xrightarrow{\text{PCP}(\mathcal{S})} \subseteq \xrightarrow{\text{PCP}_{>}(\mathcal{E} \cup \mathcal{R})} \cup \downarrow_{\mathcal{S}}$ holds on ground terms. ✓

Proof. Suppose $s \leftrightarrow_e t$ for ground terms s and t and a prime critical pair $e: \ell_2\sigma[r_1\sigma]_p \approx r_2\sigma$ generated from the overlap $\langle \ell_1 \rightarrow r_1, p, \ell_2 \rightarrow r_2 \rangle$ in \mathcal{S} . Let $u_i \approx v_i$ be the equation $\ell_i \approx r_i$ if $\ell_i \rightarrow r_i \in \mathcal{R}$ and the equation in \mathcal{E}^\pm such that $\ell_i = u_i\tau_i$ and $r_i = v_i\tau_i$ for some substitution τ_i if $\ell_i \rightarrow r_i \in \mathcal{E}^>$. In the former case we let τ_i be the empty substitution. Since the equations $u_1 \approx v_1$ and $u_2 \approx v_2$ are assumed to be variable-disjoint, the substitution $\tau = \tau_1 \cup \tau_2$ is well-defined. We distinguish two cases.

- If $p \notin \text{Pos}_{\mathcal{F}}(u_2)$ then $\langle u_1 \approx v_1, p, u_2 \approx v_2 \rangle$ is not an overlap and hence $s \downarrow_{\mathcal{S}} t$ by Lemma 7.7.12.
- Suppose $p \in \text{Pos}_{\mathcal{F}}(u_2)$. Since $u_2|_p\tau\sigma = \ell_2|_p\sigma = \ell_1\sigma = u_1\tau\sigma$ there exist an mgu μ of $u_2|_p$ and u_1 , and a substitution ρ such that $\mu\rho = \tau\sigma$. Because $u_i\mu\rho = \ell_i\sigma > r_i\sigma = v_i\mu\rho$, $v_i\mu > u_i\mu$ is impossible. Hence $e': u_2\mu[v_1\mu]_p \approx v_2\mu \in \text{CP}_{>}(\mathcal{E} \cup \mathcal{R})$ and

$$\ell_2\sigma[r_1\sigma]_p = u_2\mu\rho[v_1\mu\rho]_p = u_2\mu[v_1\mu]_p\rho \xrightarrow[e']{} v_2\mu\rho = r_2\sigma$$

Since e is prime, proper subterms of $\ell_2\sigma|_p = u_2\mu\rho|_p$ are irreducible with respect to \mathcal{S} , and hence the same holds for proper subterms of $u_2\mu$. It follows that $e' \in \text{PCP}_{>}(\mathcal{E} \cup \mathcal{R})$ and thus $\ell_2\sigma[r_1\sigma]_p \xrightarrow{\text{PCP}_{>}(\mathcal{E} \cup \mathcal{R})} r_2\sigma$. Hence also $s \xrightarrow{\text{PCP}_{>}(\mathcal{E} \cup \mathcal{R})} t$. □

This relationship between extended critical pairs among $\mathcal{E} \cup \mathcal{R}$ and critical pairs among \mathcal{S} is the final ingredient for the main result of this section. As in the preceding section, we establish correctness of ordered completion via source decreasingness.

Theorem 7.7.16. If Γ is fair then \mathcal{S}_ω is ground-complete and $\xrightarrow[\mathcal{E}_0]{*}$ and $\xrightarrow[\mathcal{E}_\omega \cup \mathcal{R}_\omega]{*}$ coincide. ✓

Proof. Termination of \mathcal{S}_ω is a consequence of Lemma 7.7.6 and the definition of $\mathcal{E}_\omega^>$. Next we show that \mathcal{S}_ω is ground-confluent. To this end, we show that labeled \mathcal{S}_ω reduction is source decreasing on ground terms. So let s, t , and u be ground terms such that

$$t \xrightarrow[\mathcal{S}_\omega]{s} s \xrightarrow[\mathcal{S}_\omega]{s} u$$

Lemma 7.2.16 yields $t \nabla_s^2 u$ (where \mathcal{S}_ω takes the place of \mathcal{R} in the definition of ∇_s). Let $v \nabla_s w$ appear in this sequence (so $t = v$ or $w = u$ and both terms are ground). We have $s > v$, $s > w$, and $(v, w) \in \downarrow_{\mathcal{S}_\omega} \cup \leftrightarrow_{\mathcal{E}_\infty}$ by the definition of ∇_s , Lemma 7.7.15, and fairness of Γ .

- If $v \downarrow_{\mathcal{S}_\omega} w$ then $v \xrightarrow{\mathbb{W}v}_{\mathcal{S}_\omega}^* \cdot \mathcal{S}_\omega^* \xleftarrow{\mathbb{W}w} w$ and thus $v \xleftrightarrow{\mathbb{S}}_{\mathcal{S}_\omega}^* w$.
- If $v \leftrightarrow_{\mathcal{E}_\infty} w$ then $v \leftrightarrow_{\mathcal{E}_i} w$ for some $i \geq 0$ and thus $v \xleftrightarrow{\mathbb{S}}_{\mathcal{S}_\omega}^* w$ by Corollary 7.7.13.

Hence $t \xleftrightarrow{\mathbb{S}}_{\mathcal{S}_\omega}^* u$. Confluence of the ARS that is obtained by restricting \mathcal{S}_ω to ground terms now follows from Lemmata 7.2.9 and 7.2.5. It remains to show $\leftrightarrow_{\mathcal{E}_0}^* = \leftrightarrow_{\mathcal{E}_\omega \cup \mathcal{R}_\omega}^*$. Using Corollary 7.7.4 we obtain $\rightarrow_{\mathcal{E}_i \cup \mathcal{R}_i} \subseteq \leftrightarrow_{\mathcal{E}_0}^*$ for all $i \geq 0$ by a straightforward induction argument. This in turn yields $\leftrightarrow_{\mathcal{E}_\infty \cup \mathcal{R}_\infty}^* \subseteq \leftrightarrow_{\mathcal{E}_0}^*$ and in particular $\leftrightarrow_{\mathcal{E}_\omega \cup \mathcal{R}_\omega}^* \subseteq \leftrightarrow_{\mathcal{E}_0}^*$. The reverse inclusion follows from Corollary 7.7.9 and the inclusion $\leftrightarrow_{\mathcal{E}_0}^* \subseteq \leftrightarrow_{\mathcal{E}_\infty \cup \mathcal{R}_\infty}^*$. \square

If \mathcal{E}_ω is empty, the TRS \mathcal{R}_ω is not only ground-confluent but actually confluent on all terms. Even though this result is not surprising, we did not find it explicitly stated in the literature.

Theorem 7.7.17. *If Γ is fair and $\mathcal{E}_\omega = \emptyset$ then \mathcal{R}_ω is a complete presentation of \mathcal{E}_0 .* \checkmark

Proof. We have $\text{PCP}(\mathcal{R}_\omega) \subseteq \text{PCP}_>(\mathcal{E}_\omega \cup \mathcal{R}_\omega)$ since $\mathcal{R}_\omega \subseteq >$. Hence the result follows from fairness and Theorem 7.7.10. \square

Example 7.7.18. *Consider the ES \mathcal{E} from Example 7.7.1 and $>_{\text{ipo}}$ with precedence $+ > 0$. After two orient steps, we apply deduce:*

$$1 \cdot (-x + x) \rightarrow 0 \quad 1 \cdot (x + -x) \rightarrow x + -x \quad -x + x \approx y + -y \quad 1 \cdot (-z + z) \approx x + -x$$

The newly added equation is simplified and then oriented:

$$1 \cdot (-x + x) \rightarrow 0 \quad 1 \cdot (x + -x) \rightarrow x + -x \quad -x + x \approx y + -y \quad x + -x \rightarrow 0$$

Using the new rewrite rule, the remaining equation is simplified, the second rule is subjected to compose and subsequently to collapse:

$$1 \cdot (-x + x) \rightarrow 0 \quad 1 \cdot 0 \approx 0 \quad -x + x \approx 0 \quad x + -x \rightarrow 0$$

Orienting both equations results in:

$$1 \cdot (-x + x) \rightarrow 0 \quad 1 \cdot 0 \rightarrow 0 \quad -x + x \rightarrow 0 \quad x + -x \rightarrow 0$$

At this point the first rule is collapsed using the third rule, and subsequently oriented (into an existing rule):

$$1 \cdot 0 \rightarrow 0 \quad -x + x \rightarrow 0 \quad x + -x \rightarrow 0$$

This sequence can be extended to an infinite run by repeatedly adding (using deduce) and deleting the trivial equation $0 \approx 0$. Then the set of persistent rules \mathcal{R}_ω coincides with the TRS \mathcal{R} from Example 7.7.1, and $\mathcal{E}_\omega = \emptyset$.

The final result in this section is in the spirit of Theorem 7.4.7 but for ordered completion, showing that a ground-complete system can be interreduced to some extent.

7. Abstract Completion, Formalized

Definition 7.7.19. Given a ground-complete system $\mathcal{S} = \mathcal{E}^> \cup \mathcal{R}$, we define

$$\begin{aligned}\mathcal{R}' &= \{\ell \rightarrow r \mid \ell \rightarrow r \in \dot{\mathcal{Q}} \text{ and } \ell \in \text{NF}(\xrightarrow{\mathcal{S}})\} \\ \mathcal{E}' &= \{s \downarrow_{\mathcal{R}'} \approx t \downarrow_{\mathcal{R}'} \mid s \approx t \in \mathcal{E}\} \setminus =\end{aligned}$$

where $\mathcal{Q} = \mathcal{R} \cup (\mathcal{E}^\pm \cap >)$ and $\dot{\mathcal{Q}}$ is defined in Definition 7.4.5.

Here we write $t \xrightarrow{\mathcal{S}} u$ if there are a rule $\ell \rightarrow r \in \mathcal{S}$, a context C , and a substitution σ such that $t = C[\ell\sigma]$, $u = C[r\sigma]$, and $t \triangleright \ell$. For example, if \mathcal{E} is empty and \mathcal{R} consists of the single rule $\mathbf{f}(x, y) \rightarrow \mathbf{g}(x)$ we have $\mathbf{f}(y, z) \in \text{NF}(\xrightarrow{\mathcal{S}})$, but $\mathbf{f}(\mathbf{g}(x), y) \notin \text{NF}(\xrightarrow{\mathcal{S}})$ and $\mathbf{f}(x, x) \notin \text{NF}(\xrightarrow{\mathcal{S}})$.

Theorem 7.7.20. If $\mathcal{S} = \mathcal{E}^> \cup \mathcal{R}$ is ground-complete then $\mathcal{S}' = \mathcal{E}'^> \cup \mathcal{R}'$ is ground-complete and normalization and conversion equivalent on ground terms. \square

Proof. We first show $\text{NF}(\mathcal{S}') \subseteq \text{NF}(\mathcal{S})$. For a rule $\ell \rightarrow r \in \mathcal{S}$, let $b_{\ell \rightarrow r}$ be \perp if $\ell \rightarrow r \in \mathcal{Q}$ and \top otherwise. We prove $\ell \notin \text{NF}(\mathcal{S}')$ for every rule $\ell \rightarrow r \in \mathcal{S}$, by induction on $(\ell, b_{\ell \rightarrow r})$ with respect to the lexicographic combination of \triangleright and the order where $\top > \perp$.

- If $\ell \rightarrow r \in \mathcal{Q}$ two cases can be distinguished. If $\ell \notin \text{NF}(\xrightarrow{\mathcal{S}})$ then $\ell \triangleright \ell'$ for some rule $\ell' \rightarrow r' \in \mathcal{S}$ and thus $\ell' \notin \text{NF}(\mathcal{S}')$ by the induction hypothesis. Hence also $\ell \notin \text{NF}(\mathcal{S}')$. If $\ell \in \text{NF}(\xrightarrow{\mathcal{S}})$ then, by construction of \mathcal{R}' , there is some rule $\ell \rightarrow r' \in \mathcal{R}'$ (modulo renaming), so $\ell \notin \text{NF}(\mathcal{S}')$.
- If $\ell \rightarrow r \notin \mathcal{Q}$ then $\ell = u\sigma$ and $r = v\sigma$ for some equation $u \approx v \in \mathcal{E}^\pm$ and substitution σ such that $\ell > r$. We distinguish two cases. First, if $u \in \text{NF}(\mathcal{R}')$ then $u = u \downarrow_{\mathcal{R}'}$. We have $\ell > r \geq v \downarrow_{\mathcal{R}'} \sigma$ because $\mathcal{R}' \subseteq >$ and hence $u \neq v \downarrow_{\mathcal{R}'}$. It follows that $u \approx v \downarrow_{\mathcal{R}'} \in \mathcal{E}'^\pm$ and thus $\ell \rightarrow v \downarrow_{\mathcal{R}'} \sigma \in \mathcal{E}'^>$. Hence $\ell \notin \text{NF}(\mathcal{S}')$. Second, if $u \notin \text{NF}(\mathcal{R}')$ then $u \notin \text{NF}(\dot{\mathcal{Q}})$ since $\mathcal{R}' \subseteq \dot{\mathcal{Q}}$. So there exists a rule $\ell' \rightarrow r' \in \mathcal{Q}$ such that $u \triangleright \ell'$. Clearly $\ell \triangleright \ell'$. Since $\ell \rightarrow r \notin \mathcal{Q}$, the induction hypothesis yields $\ell' \notin \text{NF}(\mathcal{S}')$. Hence also $\ell \notin \text{NF}(\mathcal{S}')$.

We next establish the inclusion $\rightarrow_{\mathcal{S}'} \subseteq \leftrightarrow_{\mathcal{S}}^*$ on ground terms. We have $\mathcal{E}' \cup \mathcal{R}' \subseteq \leftrightarrow_{\mathcal{E} \cup \mathcal{R}}^*$ by construction. For ground terms s and t , a step $s \rightarrow_{\mathcal{S}'} t$ implies $s \leftrightarrow_{\mathcal{E}' \cup \mathcal{R}'} t$ and hence existence of a conversion $s \leftrightarrow_{\mathcal{E} \cup \mathcal{R}}^* t$. We can also obtain such a conversion where all intermediate terms are ground by replacing every variable with some ground term. Since the reduction order $>$ is ground-total, $\rightarrow_{\mathcal{E} \cup \mathcal{R}} \subseteq \leftrightarrow_{\mathcal{S}}^*$ holds on ground terms. Hence there is a conversion $s \leftrightarrow_{\mathcal{S}}^* t$.

Moreover, the system \mathcal{S}' is clearly terminating as it is included in $>$. Thus the result follows from Lemma 7.4.4(2), viewing \mathcal{S} and \mathcal{S}' as ARSs on ground terms. \square

We illustrate the transformation of Definition 7.7.19 on a concrete example.

Example 7.7.21. Consider the following system with \mathcal{R} consisting of one rule and \mathcal{E} consisting of three equations:

$$\begin{aligned}\mathbf{s}(\mathbf{s}(x)) + \mathbf{s}(x) &\rightarrow \mathbf{s}(x) + \mathbf{s}(\mathbf{s}(x)) & x + \mathbf{s}(y) &\approx \mathbf{s}(x + y) & x + y &\approx y + x \\ \mathbf{s}(x) + y &\approx \mathbf{s}(x + y)\end{aligned}$$

7.8. Completeness Results for Ordered Completion

It is ground-complete for the lexicographic path order [65] with $+ > s$ as precedence. We have $\mathcal{Q} = \mathcal{R} \cup \{x + s(y) \rightarrow s(x + y), s(x) + y \rightarrow s(x + y)\}$. Since the term $s(s(x)) + s(x)$ is reducible by the rule $s(x) + x \rightarrow x + s(x) \in \mathcal{S}$ and $s(s(x)) + s(x) \triangleright s(x) + x$, the rule of \mathcal{R} does not remain in \mathcal{R}' . Hence, $\mathcal{R}' = \{x + s(y) \rightarrow s(x + y), s(x) + y \rightarrow s(x + y)\}$ and $\mathcal{E}' = \{x + y \approx y + x\}$.

One may wonder whether \mathcal{R}' can simply be defined as $\ddot{\mathcal{Q}}$ instead of imposing a strict encompassment condition. The following example shows that this destroys reducibility.

Example 7.7.22. Consider the following system where \mathcal{R} consists of two rules and \mathcal{E} consists of one equation:

$$f(x, y) \rightarrow g(x) \qquad f(x, y) \rightarrow g(y) \qquad g(x) \approx g(y)$$

Then $\mathcal{E}^> \cup \mathcal{R}$ is ground-complete if $>$ is the lexicographic path order with $f > g$ as precedence. We have $\mathcal{R}' = \dot{\mathcal{Q}} = \mathcal{Q} = \mathcal{R}$ and $\mathcal{E}' = \mathcal{E}$ but $\ddot{\mathcal{Q}} = \emptyset$.

Note that we obtain an equivalent ground-complete system if we add, for instance, an equation $g(g(x)) \approx g(y)$. This shows that even systems which are simplified with respect to the procedure suggested by Theorem 7.7.20 are not unique.

This section resumes our results on ordered completion [59]. Like in Sections 7.3 and 7.6, our proofs deviate from the standard approach [14] in that we avoid proof orders in favor of different, simpler orderings as required, together with source decreasingness. Again, we also support prime critical pairs. For Theorem 7.7.17 and the interreduction result of Theorem 7.7.20 we are not aware of earlier references in the literature.

7.8. Completeness Results for Ordered Completion

Ordered completion never fails and its limit always constitutes a ground-complete system. On the other hand, if there is a *complete* presentation that is compatible with the employed reduction order, does ordered completion also produce a complete presentation, ending with $\mathcal{E}_\omega = \emptyset$? In this section we revisit two results from the literature which provide sufficient conditions for ordered completion to always derive a complete system, independent of the strategy employed by a completion procedure. In Section 7.8.1 we reprove the result by Bachmair, Dershowitz, and Plaisted for the case where the reduction order is ground total [14]. The corresponding result by Devie [34] for linear systems is considered in Section 7.8.2.

7.8.1. Ground-Total Orders

In this subsection we consider a fair run Γ of ordered completion

$$(\mathcal{E}_0, \mathcal{R}_0) \vdash_o (\mathcal{E}_1, \mathcal{R}_1) \vdash_o (\mathcal{E}_2, \mathcal{R}_2) \vdash_o \dots$$

with respect to a ground-total reduction order $>$. If $\mathcal{E}_\omega = \emptyset$ then the TRS \mathcal{R}_ω is a complete presentation of \mathcal{E}_0 by Theorem 7.7.17. According to Bachmair et al. [14,

7. Abstract Completion, Formalized

Theorem 2], under certain conditions fair runs always conclude with $\mathcal{E}_\omega = \emptyset$ whenever there exists a complete presentation of \mathcal{E}_0 compatible with $>$. In the remainder of this subsection we give a formalized proof of this result. Like the original proof, it is based on the idea that ground-completeness of \mathcal{R}_ω is preserved under signature extension with constants. Let \mathcal{K} be a set of different fresh constants \hat{x} for every variable $x \in \mathcal{V}$. We first show that the reduction order $>$ can be extended to a ground-total order on the signature augmented by \mathcal{K} such that minimum constants are preserved.

Lemma 7.8.1. *There exists a ground-total reduction order $>^\mathcal{K}$ on $\mathcal{T}(\mathcal{F} \cup \mathcal{K}, \mathcal{V})$ such that $> \subseteq >^\mathcal{K}$ and the minimum constant with respect to $>$ is also minimum in $>^\mathcal{K}$. \checkmark*

Proof. Let $\perp \in \mathcal{F}$ be the minimum constant with respect to $>$. We consider the KBO \sqsubset_{kbo} with weights $w_0 = 1$ and $w(f) = 1$ for all $f \in \mathcal{F} \cup \mathcal{K}$ together with a precedence \sqsubset which is total on $\mathcal{F} \cup \mathcal{K}$, has \perp as the minimum element, and satisfies $\hat{x} \sqsubset f$ for all $f \in \mathcal{F}$ and $\hat{x} \in \mathcal{K}$. Given a term $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{K}, \mathcal{V})$, we write t_\perp for the term obtained from t by replacing every constant in \mathcal{K} with \perp . Furthermore, we define $s >^\mathcal{K} t$ as $s_\perp > t_\perp$, or both $s_\perp = t_\perp$ and $s \sqsubset_{\text{kbo}} t$. We show that $>^\mathcal{K}$ is a ground-total reduction order with the stated properties. Ground totality of $>^\mathcal{K}$ follows from ground totality of \sqsubset_{kbo} given the total precedence. Well-foundedness holds by construction as a lexicographic combination of well-founded relations. Closure under substitutions is satisfied because it holds for both $>$ and \sqsubset_{kbo} , and $s_\perp = t_\perp$ implies $s\sigma_\perp = t\sigma_\perp$. Similar arguments apply to closure under contexts and transitivity. By construction of \sqsubset and the definition of $>^\mathcal{K}$, the constant \perp is still minimal. Moreover $>^\mathcal{K}$ extends $>$ because $s > t$ implies $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, so $s_\perp = s > t = t_\perp$ and hence $s >^\mathcal{K} t$. \square

We write \hat{t} for the ground term that is obtained from t by replacing every variable x by the constant \hat{x} . In the next lemma we verify some basic properties related to this *grounding* operation.

Lemma 7.8.2. *Let \mathcal{R} be a TRS over a signature \mathcal{F} and let $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$.*

1. *If $s > t$ then $\hat{s} >^\mathcal{K} \hat{t}$. \checkmark*
2. *Suppose $s \neq t$. Then $s \rightarrow_\mathcal{R} t$ if and only if $\hat{s} \rightarrow_\mathcal{R} \hat{t}$. \checkmark*

Proof.

1. Suppose $s > t$. Lemma 7.8.1 yields $s >^\mathcal{K} t$ and, because $>^\mathcal{K}$ is closed under substitutions, $\hat{s} >^\mathcal{K} \hat{t}$.
2. We consider the two implications separately.
 - If $s \rightarrow_\mathcal{R} t$ then $\text{Var}(t) \subseteq \text{Var}(s)$. Let σ be a substitution such that $\hat{s} = s\sigma$. We have $\hat{t} = t\sigma$ and thus $\hat{s} = s\sigma \rightarrow_\mathcal{R} t\sigma = \hat{t}$.
 - Conversely, if $\hat{s} \rightarrow_\mathcal{R} \hat{t}$ then $\hat{s}|_p = \ell\sigma$ and $\hat{t} = \hat{s}[r\sigma]_p$ for some rule $\ell \rightarrow r \in \mathcal{R}$, position p , and substitution σ . We denote the substitution $\{x \mapsto \phi(\sigma(x)) \mid x \in \mathcal{V}\}$ by σ_ϕ . Here $\phi(u)$ denotes the term obtained from u after replacing every constant \hat{x} of \mathcal{K} by x . Because $s|_p = \phi(\hat{s}|_p) = \phi(\ell\sigma) = \ell\sigma_\phi$ and $t = \phi(\hat{t}) = \phi(\hat{s}[r\sigma]_p) = s[r\sigma_\phi]_p$, we obtain $s \rightarrow_\mathcal{R} t$ as desired. \square

7.8. Completeness Results for Ordered Completion

It is not hard to see that the TRS \mathcal{S}_ω still constitutes a ground-complete presentation of \mathcal{E}_0 when considered over the extended signature, as shown below.

Lemma 7.8.3. *The TRS \mathcal{S}_ω is ground-complete over $\mathcal{F} \cup \mathcal{K}$ and $\leftrightarrow_{\mathcal{E}_\omega \cup \mathcal{R}_\omega}^* = \leftrightarrow_{\mathcal{E}_0}^*$. \checkmark*

Proof. Since $>^\mathcal{K}$ contains $>$ by Lemma 7.8.1, the run Γ is also a valid run with respect to $>^\mathcal{K}$. It is moreover fair since $> \subseteq >^\mathcal{K}$ implies $\text{PCP}_{>^\mathcal{K}}(\mathcal{E}) \subseteq \text{PCP}_{>}(\mathcal{E})$ for any set of equations \mathcal{E} , by Definition 7.7.11. Hence the result follows from Theorem 7.7.16. \square

An important observation for the completeness proof below is that normal forms with respect to the final system \mathcal{S}_ω and with respect to the union \mathcal{S}_∞ of intermediate systems coincide, as shown below.

Lemma 7.8.4. *The inclusion $\text{NF}(\mathcal{S}_\omega) \subseteq \text{NF}(\mathcal{S}_\infty)$ holds. \checkmark*

Proof. The result is an immediate consequence of the following two claims:

- (a) If $\ell \approx r \in \mathcal{E}_\infty^\pm$, $\ell \in \text{NF}(\mathcal{R}_\infty)$, and $\ell\sigma > r\sigma$ then $\ell\sigma \notin \text{NF}(\mathcal{E}_\omega^>)$.
- (b) If $\ell \rightarrow r \in \mathcal{R}_\infty$ then $\ell \notin \text{NF}(\mathcal{S}_\omega)$.

For claim (a) we use induction on $\{\ell, r\}$ with respect to \succ_{mul} . If $\ell \approx r \in \mathcal{E}_\omega^\pm$ the result is immediate. Otherwise, $\ell \approx r \in \mathcal{E}_i \setminus \mathcal{E}_{i+1}$ or $r \approx \ell \in \mathcal{E}_i \setminus \mathcal{E}_{i+1}$ for some $i \geq 0$. Without loss of generality we assume the former since the latter case is similar. From Lemma 7.7.3(2) we obtain $\ell \rightarrow_{\mathcal{S}_{i+1}}^{\mathbb{D}^1} \cdot \mathcal{E}_{i+1}^\pm r$, $\ell \rightarrow r \in \mathcal{R}_{i+1}$, $r \rightarrow \ell \in \mathcal{R}_{i+1}$, or $\ell = r$. The latter two cases are impossible because of the assumption $\ell\sigma > r\sigma$ and the inclusion $\mathcal{R}_{i+1} \subseteq \mathcal{R}_\infty \subseteq >$. Also $\ell \rightarrow r \in \mathcal{R}_{i+1}$ is impossible because of the assumption $\ell \in \text{NF}(\mathcal{R}_\infty)$.

- Suppose $\ell \rightarrow_{\mathcal{S}_{i+1}}^{\mathbb{D}^1} u$ and $u \approx r \in \mathcal{E}_{i+1}^\pm$ for some term u . The step $\ell \rightarrow_{\mathcal{S}_{i+1}} u$ cannot use a rule in \mathcal{R}_{i+1} because $\ell \in \text{NF}(\mathcal{R}_\infty)$. So there must be an equation $\ell' \approx r' \in \mathcal{E}_{i+1}^\pm$, a substitution τ , and a position p in ℓ such that $\ell|_p = \ell'\tau$, $u|_p = r'\tau$, $\ell'\tau > r'\tau$, and $\ell \triangleright \ell'$. Because of $\ell \triangleright \ell'\tau > r'\tau \triangleright r'$ we have $\ell \succ r'$, and therefore $\{\ell, r\} \succ_{\text{mul}} \{\ell', r'\}$. Moreover, $\ell' \in \text{NF}(\mathcal{R}_\infty)$. The induction hypothesis yields $\ell'\tau \notin \text{NF}(\mathcal{E}_\omega^>)$. Since $\ell \triangleright \ell'\tau$, we have $\ell \notin \text{NF}(\mathcal{E}_\omega^>)$ and thus also $\ell\sigma \notin \text{NF}(\mathcal{E}_\omega^>)$.
- In the remaining case we have $r \rightarrow_{\mathcal{S}_{i+1}}^{\mathbb{D}^1} u$ and $u \approx \ell \in \mathcal{E}_{i+1}^\pm$ for some term u . We have $r > u$ and thus also $r \succ u$ and $\{\ell, r\} \succ_{\text{mul}} \{\ell, u\}$. Because $\ell\sigma > r\sigma > u\sigma$, the result follows from the induction hypothesis.

For claim (b) we use induction on (ℓ, r) with respect to \succ_{lex} . If $\ell \rightarrow r \in \mathcal{R}_\omega$ then $\ell \notin \text{NF}(\mathcal{S}_\omega)$ trivially holds. Otherwise, $\ell \rightarrow r \in \mathcal{R}_i \setminus \mathcal{R}_{i+1}$ for some $i \geq 0$. From Lemma 7.7.3(3) we obtain $\ell \rightarrow_{\mathcal{S}_{i+1}}^{\mathbb{D}^2} \cdot \mathcal{E}_{i+1} r$ or $\ell \mathcal{R}_{i+1} \cdot \mathcal{S}_{i+1} \leftarrow r$. In the latter case there is a term u such that $\ell \rightarrow u \in \mathcal{R}_{i+1}$ and $r \rightarrow_{\mathcal{S}_{i+1}} u$. Since this implies $r > u$ and thus $(\ell, r) \succ_{\text{lex}} (\ell, u)$, we obtain $\ell \notin \text{NF}(\mathcal{S}_\omega)$ from the induction hypothesis. In the former case there is a term u such that $\ell \rightarrow_{\mathcal{S}_{i+1}}^{\mathbb{D}^2} u$ and $u \approx r \in \mathcal{E}_{i+1}$. If the step $\ell \rightarrow_{\mathcal{S}_{i+1}} u$ uses a rule $\ell' \rightarrow r' \in \mathcal{R}_{i+1}$ then the result follows from the induction hypothesis because $\ell \triangleright \ell'$ implies $(\ell, r) \succ_{\text{lex}} (\ell', r')$, and $\ell' \notin \text{NF}(\mathcal{S}_\omega)$ implies $\ell \notin \text{NF}(\mathcal{S}_\omega)$. Otherwise, there exist

7. Abstract Completion, Formalized

an equation $\ell' \approx r' \in \mathcal{E}_{i+1}$, a position p in ℓ , and a substitution σ such that $\ell|_p = \ell'\sigma$, $r|_p = r'\sigma$, $\ell'\sigma > r'\sigma$, and $\ell \triangleright \ell'$. If $\ell' \in \text{NF}(\mathcal{R}_\infty)$ then we obtain $\ell'\sigma \notin \text{NF}(\mathcal{E}_\omega^>)$ from claim (a) and thus $\ell \notin \text{NF}(\mathcal{S}_\omega)$ because $\ell \triangleright \ell'\sigma$. If $\ell' \notin \text{NF}(\mathcal{R}_\infty)$ then there exists some rule $\ell'' \rightarrow r'' \in \mathcal{R}_\infty$ such that $\ell' \triangleright \ell''$. In this case we have $\ell > \ell''$ and thus $(\ell, r) \succ_{\text{lex}} (\ell'', r'')$. We obtain $\ell'' \notin \text{NF}(\mathcal{S}_\omega)$ from the induction hypothesis. Hence also $\ell \notin \text{NF}(\mathcal{S}_\omega)$. \square

Corollary 7.8.5. *The identity $\text{NF}(\mathcal{S}_\omega) = \text{NF}(\mathcal{S}_\infty)$ holds.* \checkmark

Proof. We obtain $\text{NF}(\mathcal{S}_\infty) \subseteq \text{NF}(\mathcal{S}_\omega)$ from the inclusion $\rightarrow_{\mathcal{S}_\omega} \subseteq \rightarrow_{\mathcal{S}_\infty}$ and hence the result follows from Lemma 7.8.4. \square

Hereafter we assume that there is a complete presentation \mathcal{R} of \mathcal{E}_0 with $\mathcal{R} \subseteq >$. We next show that grounded terms which are \mathcal{S}_ω -normal forms are also \mathcal{R} -normal forms.

Lemma 7.8.6. *If $\hat{t} \in \text{NF}(\mathcal{S}_\omega)$ then $\hat{t} \in \text{NF}(\mathcal{R})$.* \checkmark

Proof. Suppose $\hat{t} \in \text{NF}(\mathcal{S}_\omega)$ but $\hat{t} \notin \text{NF}(\mathcal{R})$, so $\hat{t} \rightarrow_{\mathcal{R}} u$ for some term u . Since \hat{t} is ground and \mathcal{R} is terminating, also u is ground. We obtain $\hat{t} \downarrow_{\mathcal{S}_\omega} u$ from the ground-completeness of \mathcal{S}_ω (Lemma 7.8.3). Since $\hat{t} >^{\mathcal{K}} u$ by the global assumption $\mathcal{R} \subseteq >$ and Lemma 7.8.2(1), the joining sequence cannot be of the form $\hat{t} \xrightarrow{\mathcal{S}_\omega^*} u$ as this would imply $u \geq \hat{t}$ and thus $u \geq^{\mathcal{K}} \hat{t}$, contradicting the well-foundedness of $>^{\mathcal{K}}$. Therefore we must have $\hat{t} \rightarrow_{\mathcal{S}_\omega}^+ \cdot \mathcal{S}_\omega^* \leftarrow u$ which means that \hat{t} is reducible in \mathcal{S}_ω , contradicting the assumption $\hat{t} \in \text{NF}(\mathcal{S}_\omega)$. \square

The preliminary results collected so far now lead to the following key observation: If a grounded term \hat{s} is reducible then so is its (possibly non-ground) counterpart s . In Lemma 7.8.8 below we can then connect \mathcal{R} -reducibility to \mathcal{S}_ω -reducibility for terms over the original signature.

Lemma 7.8.7. *If $\hat{s} \rightarrow_{\mathcal{S}_\infty} \hat{t}$ then $s \notin \text{NF}(\mathcal{S}_\infty)$.* \checkmark

Proof. There exist an equation $\ell \approx r \in \mathcal{E}_\infty^\pm \cup \mathcal{R}_\infty$, a position p , and a substitution σ such that $\hat{s}|_p = \ell\hat{\sigma}$, $\hat{t} = \hat{s}[r\hat{\sigma}]_p$, and $\ell\hat{\sigma} >^{\mathcal{K}} r\hat{\sigma}$. We perform induction on \hat{t} with respect to $>^{\mathcal{K}}$. If $p \neq \epsilon$ then $\hat{t} \triangleright r\hat{\sigma}$ and thus $\hat{t} >^{\mathcal{K}} r\hat{\sigma}$ because $>^{\mathcal{K}}$ is a ground-total reduction order. The induction hypothesis yields $\ell\sigma \notin \text{NF}(\mathcal{S}_\infty)$, which implies $s \notin \text{NF}(\mathcal{S}_\infty)$. So in the following we assume that the step $\hat{s} \rightarrow_{\mathcal{S}_\infty} \hat{t}$ takes place at the root position. If $s > t$ then $s \rightarrow t \in \mathcal{S}_\infty$, from which the claim is immediate. This covers the case $\ell \approx r \in \mathcal{R}_\infty$, so if $s \not> t$ then $\ell \approx r \in \mathcal{E}_\infty^\pm$. We distinguish two cases, $\hat{t} \in \text{NF}(\mathcal{S}_\infty)$ and $\hat{t} \notin \text{NF}(\mathcal{S}_\infty)$.

- If $\hat{t} \in \text{NF}(\mathcal{S}_\infty)$ then $\hat{t} \in \text{NF}(\mathcal{S}_\omega)$ by Corollary 7.8.5 and thus $\hat{t} \in \text{NF}(\mathcal{R})$ by Lemma 7.8.6. From Lemma 7.8.3 and the fact that \mathcal{R} is a complete presentation of \mathcal{E}_0 we obtain $\hat{s} \rightarrow_{\mathcal{R}}^+ \hat{t}$. The latter implies $s \rightarrow_{\mathcal{R}}^+ t$ by Lemma 7.8.2(2) and thus $s > t$, which is a contradiction.
- Suppose $\hat{t} \notin \text{NF}(\mathcal{S}_\infty)$. We distinguish two further cases, depending on whether or not $\ell \approx r$ belongs to \mathcal{E}_ω^\pm .

If $\ell \approx r \notin \mathcal{E}_\omega^\pm$ then $\ell \approx r \in (\mathcal{E}_i \setminus \mathcal{E}_{i+1})^\pm$ for some $i \geq 0$. From Lemma 7.7.3(2) we obtain $\ell (\rightarrow_{\mathcal{S}_{i+1}} \cdot \mathcal{E}_{i+1}^\pm)^\pm r$, $\ell \rightarrow r \in \mathcal{R}_{i+1}$, $r \rightarrow \ell \in \mathcal{R}_{i+1}$, or $\ell = r$. The last two

7.8. Completeness Results for Ordered Completion

cases contradict $\hat{s} >^{\mathcal{K}} \hat{t}$. If $\ell \rightarrow_{\mathcal{S}_{i+1}} \cdot \mathcal{E}_{i+1}^{\pm} r$ or $\ell \rightarrow r \in \mathcal{R}_{i+1}$ then $\ell \notin \text{NF}(\mathcal{S}_{\infty})$ and thus $s = \ell\sigma \notin \text{NF}(\mathcal{S}_{\infty})$. Otherwise, $r \rightarrow_{\mathcal{S}_{i+1}} u$ for some term u with $u \approx \ell \in \mathcal{E}_{i+1}^{\pm}$. We have $s = \ell\sigma \leftrightarrow_{\mathcal{E}_{\infty}} u\sigma \xrightarrow{\mathcal{S}_{\infty}} r\sigma = t$ and thus $\hat{s} = \ell\hat{\sigma} \rightarrow_{\mathcal{S}_{\infty}} u\hat{\sigma} = \widehat{u\sigma}$ and $\hat{t} = r\hat{\sigma} >^{\mathcal{K}} u\hat{\sigma}$. The induction hypothesis yields $s \notin \text{NF}(\mathcal{S}_{\infty})$.

In the second case we assume $\ell \approx r \in \mathcal{E}_{\omega}^{\pm}$. From the assumption $\hat{t} \notin \text{NF}(\mathcal{S}_{\infty})$ we obtain a term u such that $\hat{t} \rightarrow_{\mathcal{S}_{\infty}} \hat{u}$. We have $\hat{t} >^{\mathcal{K}} \hat{u}$ and thus $t \notin \text{NF}(\mathcal{S}_{\omega})$ by the induction hypothesis. Consider an innermost \mathcal{S}_{ω} -step starting from t , say $t \xrightarrow{i}_{\mathcal{S}_{\omega}} v$, such that there exists a peak

$$s = \ell\sigma \leftrightarrow_{\ell \approx r} r\sigma = t \xrightarrow{q}_{\mathcal{S}_{\omega}} v \quad (\star)$$

with $\ell\sigma = s \not\approx t = r\sigma$. If the two steps form an overlap then $s \leftrightarrow_{\text{PCP}_{>}(\mathcal{E}_{\omega} \cup \mathcal{R}_{\omega})} v$ since $t \xrightarrow{i}_{\mathcal{S}_{\omega}} v$ is innermost, and thus $s \leftrightarrow_{\mathcal{E}_{\infty}} v$ or $s \downarrow_{\mathcal{S}_{\omega}} v$ is obtained from the fairness of the run. In the former case, since $\hat{s} >^{\mathcal{K}} \hat{t} >^{\mathcal{K}} \hat{v}$, we have $\hat{s} \rightarrow_{\mathcal{S}_{\infty}} \hat{v}$ and thus the induction hypothesis applies. If on the other hand $s \downarrow_{\mathcal{S}_{\omega}} v$ then we cannot have $v \rightarrow_{\mathcal{S}_{\omega}}^* s$ as this would imply $v \geq s$, contradicting $\hat{s} >^{\mathcal{K}} \hat{v}$ because $>$ and $>^{\mathcal{K}}$ are compatible by Lemma 7.8.2(1). So s must be \mathcal{S}_{ω} -reducible.

Otherwise, the peak (\star) constitutes a variable overlap, so there is some variable $x \in \text{Var}(r)$ and positions q_1 and q_2 such that $r|_{q_1} = x$ and $q = q_1q_2$. If $x \notin \text{Var}(\ell)$ then $s \leftrightarrow_{\mathcal{E}_{\infty}} v$ and the induction hypothesis applies as before. Otherwise, $s = \ell\sigma$ is reducible in \mathcal{S}_{∞} . \square

Lemma 7.8.8. *The inclusion $\text{NF}(\mathcal{S}_{\omega}) \subseteq \text{NF}(\mathcal{R})$ holds.* \checkmark

Proof. Suppose $t \rightarrow_{\mathcal{R}} u$, so $t > u$ and thus also $\hat{t} >^{\mathcal{K}} \hat{u}$. From Lemma 7.8.3 we obtain $\hat{t} \downarrow_{\mathcal{S}_{\omega}} \hat{u}$. Like in the proof of Lemma 7.8.6, $\hat{u} \rightarrow_{\mathcal{S}_{\omega}}^* \hat{t}$ would imply $\hat{u} \geq^{\mathcal{K}} \hat{t}$, contradicting well-foundedness of $>^{\mathcal{K}}$. Therefore the joining sequence must be of the shape $\hat{t} \rightarrow_{\mathcal{S}_{\omega}}^+ \cdot \mathcal{S}_{\omega}^* \leftarrow \hat{u}$ and thus \hat{t} is reducible in \mathcal{S}_{ω} and also in \mathcal{S}_{∞} because $\mathcal{S}_{\omega} \subseteq \mathcal{S}_{\infty}$. Lemma 7.8.7 yields $t \notin \text{NF}(\mathcal{S}_{\infty})$ and thus $t \notin \text{NF}(\mathcal{S}_{\omega})$ by Corollary 7.8.5. \square

We call $(\mathcal{E}_{\omega}, \mathcal{R}_{\omega})$ *simplified* if \mathcal{R}_{ω} is reduced and all equations in \mathcal{E}_{ω} are irreducible with respect to \mathcal{R}_{ω} , unorientable with respect to $>$, and non-trivial. We use the following auxiliary result before proving the main completeness theorem.

Lemma 7.8.9. *If $(\mathcal{E}_{\omega}, \mathcal{R}_{\omega})$ is simplified then $\text{NF}(\mathcal{R}_{\omega}) \subseteq \text{NF}(\mathcal{S}_{\omega})$.* \checkmark

Proof. Suppose $(\mathcal{E}_{\omega}, \mathcal{R}_{\omega})$ is simplified and let $s \in \text{NF}(\mathcal{R}_{\omega})$. We prove $s \in \text{NF}(\mathcal{S}_{\omega})$ by induction on s with respect to \succ . If $s \notin \text{NF}(\mathcal{S}_{\omega})$ then there exist an equation $\ell \approx r \in \mathcal{E}_{\omega}^{\pm}$, a context C , and a substitution σ such that $s = C[\ell\sigma]$ and $\ell\sigma > r\sigma$. Since $s \triangleright \ell\sigma > r\sigma \triangleright r$ implies $s \succ r$ and $r \in \text{NF}(\mathcal{R}_{\omega})$ by the assumption that $(\mathcal{E}_{\omega}, \mathcal{R}_{\omega})$ is simplified, the induction hypothesis yields $r \in \text{NF}(\mathcal{S}_{\omega})$. Hence $r \in \text{NF}(\mathcal{R})$ by Lemma 7.8.8. Since \mathcal{R} is a complete presentation of \mathcal{E}_0 , we have $\ell \downarrow_{\mathcal{R}} r$ and thus $\ell \rightarrow_{\mathcal{R}}^* r$. From $\mathcal{R} \subseteq >$ we infer $\ell > r$ or $\ell = r$, contradicting the assumption that $(\mathcal{E}_{\omega}, \mathcal{R}_{\omega})$ is simplified. \square

Theorem 7.8.10. *If $(\mathcal{E}_{\omega}, \mathcal{R}_{\omega})$ is simplified then $\mathcal{E}_{\omega} = \emptyset$ and \mathcal{R}_{ω} is literally similar to \mathcal{R} .* \checkmark

7. Abstract Completion, Formalized

Proof. Suppose $\mathcal{E}_\omega \neq \emptyset$ and let $s \approx t$ be an equation in \mathcal{E}_ω . The terms s and t are \mathcal{R} -normal forms by the assumption that $(\mathcal{E}_\omega, \mathcal{R}_\omega)$ is simplified in combination with Lemmata 7.8.8 and 7.8.9. Since \mathcal{R} is a complete presentation of \mathcal{E}_0 , we obtain $s = t$, contradicting the assumption that $(\mathcal{E}_\omega, \mathcal{R}_\omega)$ is simplified. Hence, $\mathcal{E}_\omega = \emptyset$ and therefore \mathcal{R}_ω is a complete presentation of \mathcal{E}_0 by Theorem 7.7.17. Since $(\mathcal{E}_\omega, \mathcal{R}_\omega)$ is simplified, \mathcal{R}_ω is even a canonical presentation of \mathcal{E}_0 . As $\mathcal{R}_\omega \subseteq >$, literal similarity of \mathcal{R}_ω and \mathcal{R} is concluded by Theorem 7.4.11. \square

A run of ordered completion is called *simplifying* if its limit $(\mathcal{E}_\omega, \mathcal{R}_\omega)$ is simplified.

Example 7.8.11. Consider again the ES \mathcal{E} from Example 7.7.1 and its complete presentation \mathcal{R} , which cannot be derived using standard completion. Termination of \mathcal{R} can be shown by a suitable KBO. Thus, by Theorem 7.8.10 any fair and simplifying run of ordered completion on \mathcal{E} using the same order will succeed with a variant of \mathcal{R} , independent of the employed strategy.

The results of this subsection are due to Bachmair, Dershowitz, and Plaisted [14]. However, our proof is structured into many preliminary results, as opposed to the monolithic original version, and we fill in numerous details omitted in the original version.

7.8.2. Linear Systems

The previously presented correctness and completeness results (Theorems 7.7.16 and 7.8.10) do not state any properties of the system obtained when running KB_0 with a reduction order that is not ground-total. The following example from Devie [34] shows that the restriction to ground-total orders can actually be severe.

Example 7.8.12. Consider the ES \mathcal{E} consisting of the following equations:

$$\begin{array}{lll} f_1(g_1(i_1(x))) \approx g_1(i_1(f_1(g_1(i_2(x))))) & h_1(g_1(i_1(x))) \approx g_1(i_1(x)) & f_1(a) \approx a \\ f_2(g_2(i_2(x))) \approx g_2(i_2(f_2(g_2(i_1(x))))) & h_2(g_2(i_2(x))) \approx g_2(i_2(x)) & f_2(a) \approx a \\ g_1(a) \approx a & h_1(a) \approx a & i_1(a) \approx a \\ g_2(a) \approx a & h_2(a) \approx a & i_2(a) \approx a \end{array}$$


When orienting all equations from left to right we obtain a TRS \mathcal{R} which is easily shown to be terminating by automatic tools. As all critical pairs are joinable it is confluent, and thus canonical since it is also reduced. However, \mathcal{R} cannot be oriented by any ground-total reduction order $>$. We have $i_1(a) \leftrightarrow_{\mathcal{E}}^* i_2(a)$ but neither $i_2(a) > i_1(a)$ nor $i_1(a) > i_2(a)$ can hold; using the rule $f_1(g_1(i_1(x))) \rightarrow g_1(i_1(f_1(g_1(i_2(x)))))$, the former would imply

$$f_1(g_1(i_2(a))) > f_1(g_1(i_1(a))) > g_1(i_1(f_1(g_1(i_2(a)))))$$

which contradicts well-foundedness, and for the latter a similar argument applies. As a matter of fact, in [34] it is shown that any KB_0 run starting from \mathcal{E} and using a ground-total reduction order will fail to generate a finite result.


7.8. Completeness Results for Ordered Completion

Devie [34] gives a second sufficient condition for an ordered completion procedure to compute a canonical result whenever such a presentation exists, without imposing any restriction on the reduction order. Instead, the set of input equalities \mathcal{E}_0 is required to be linear, and Devie considers an ordered completion inference system with a modified deduction rule to ensure that linearity is preserved. He moreover shows that under these circumstances a relaxed fairness condition is sufficient. In this section we give a new proof of this result which has been formalized. First we recall Devie's inference system.

Definition 7.8.13 (Linear Ordered Completion ). *The inference system KB_l of linear ordered completion consists of the rules orient, delete, compose, simplify, and collapse_> of KB_i (Definition 7.6.2) together with the following modified deduction rule:*


$$\text{deduce}_l \quad \frac{\mathcal{E}, \mathcal{R}}{\mathcal{E} \cup \{s \approx t\}, \mathcal{R}} \quad \text{if } s \xleftarrow{\mathcal{E} \cup \mathcal{R}} \cdot \xrightarrow{\mathcal{E} \cup \mathcal{R}} t \text{ and } s \approx t \text{ is linear}$$

We write $(\mathcal{E}, \mathcal{R}) \vdash_l (\mathcal{E}', \mathcal{R}')$ if $(\mathcal{E}', \mathcal{R}')$ can be reached from $(\mathcal{E}, \mathcal{R})$ by employing one of the inference rules of Definition 7.8.13.

Lemma 7.8.14. *The inclusion $\text{KB}_l \subseteq \text{KB}_o$ holds.* 

Note that in contrast to the ordered completion system KB_o , ordered rewriting using orientable instances of \mathcal{E} is not permitted in compose, simplify, and collapse_>. This is because ordered rewrite steps need not preserve linearity as stated in Lemma 7.8.15 below. For example, a compose step in KB_o on the linear rule $g(x) \rightarrow f(f(x))$ using the linear equation $f(x) \approx f(y)$ may result in the nonlinear rule $g(x) \rightarrow f(h(x, x))$ when $h(x, x)$ is substituted for the variable y and a reduction order $>$ is used such that $f(f(x)) > f(h(x, x))$.

With these restrictions, it is not hard to prove that inference steps preserve linearity.

Lemma 7.8.15. *If $\mathcal{E} \cup \mathcal{R}$ is linear and $(\mathcal{E}, \mathcal{R}) \vdash_l (\mathcal{E}', \mathcal{R}')$ then $\mathcal{E}' \cup \mathcal{R}'$ is linear.* 


From now on we consider $\mathcal{E}_0 \cup \mathcal{R}_0$ to be linear.


Definition 7.8.16. *An extended overlap (Definition 7.7.11) which satisfies $\ell_1 > r_1$ and $r_2 \not> \ell_2$, or $\ell_2 > r_2$ and $r_1 \not> \ell_1$ gives rise to a linear critical pair [34]. The set of all linear critical pairs originating from equations in \mathcal{E} is denoted $\text{LCP}_{>}(\mathcal{E})$. An infinite run*

$$(\mathcal{E}_0, \mathcal{R}_0) \vdash_l (\mathcal{E}_1, \mathcal{R}_1) \vdash_l (\mathcal{E}_2, \mathcal{R}_2) \vdash_l \dots$$

is fair if the inclusion $\text{LCP}_{>}(\mathcal{E}_\omega \cup \mathcal{R}_\omega) \subseteq \downarrow_{\mathcal{R}_\omega} \cup \leftrightarrow_{\mathcal{E}_\omega}$ holds.

Below, we consider an infinite fair run Γ . We next show that the result of a fair run without persistent equations is indeed complete.

Theorem 7.8.17. *If Γ is fair and $\mathcal{E}_\omega = \emptyset$ then \mathcal{R}_ω is a complete presentation of \mathcal{E}_ω .* 

Proof. The run Γ is also a valid KB_o run by Lemma 7.8.14. We moreover have that $\text{PCP}(\mathcal{R}_\omega) \subseteq \text{LCP}(\mathcal{E}_\omega \cup \mathcal{R}_\omega)$ since $\mathcal{R}_\omega \subseteq >$, and hence $\text{PCP}(\mathcal{R}_\omega) \subseteq \downarrow_{\mathcal{R}_\omega} \cup \leftrightarrow_{\mathcal{E}_\omega}$ by fairness. So the result follows from Theorem 7.7.10. 

7. Abstract Completion, Formalized

The following result relates equations in \mathcal{E}_∞ and rules in \mathcal{R}_∞ to persistent equations and rules, respectively.

Lemma 7.8.18.

$$1. \mathcal{E}_\infty \subseteq \frac{*}{\mathcal{R}_\infty} \cdot \frac{\overline{}}{\mathcal{E}_\omega} \cdot \frac{*}{\mathcal{R}_\infty} \quad \checkmark$$

$$2. \text{ If } \ell \rightarrow r \in \mathcal{R}_\infty \text{ then } \ell \xrightarrow{\mathcal{R}_\omega} \cdot \left(\frac{\leq \ell}{\mathcal{E}_\omega \cup \mathcal{R}_\omega} \right)^* r. \quad \checkmark$$

Proof.

1. For an equation $s \approx t \in \mathcal{E}_\infty$ we prove the desired inclusion by induction on $\{s, t\}$ with respect to $>_{\text{mul}}$.
2. By induction on (ℓ, r) with respect to $>_{\text{lex}}$. □

In order to show that \mathcal{R}_ω is Church-Rosser modulo \mathcal{E}_ω , we need a result about joinability of critical peaks modulo persistent equations.

Lemma 7.8.19. *If there is an equation $\ell \approx r \in \mathcal{E}_\omega^\pm \cup \mathcal{R}_\omega$ with $r \not\approx \ell$ that is involved in a peak $s \xleftarrow[r \approx \ell]{} \cdot \xrightarrow{\mathcal{R}_\omega} t$ then $s \xrightarrow{\mathcal{R}_\infty} \cdot \frac{\overline{}}{\mathcal{E}_\omega} \cdot \frac{*}{\mathcal{R}_\infty} t$.* ✓

Proof. If the two steps occur at parallel positions then they commute and thus $s \rightarrow_{\mathcal{R}_\omega} \cdot r \approx \ell \leftarrow t$. If the peak constitutes an overlap then $s \leftrightarrow_{\text{LCP}(\mathcal{E}_\omega \cup \mathcal{R}_\omega)} t$ since $\mathcal{R}_\omega \subseteq >$ and $r \not\approx \ell$ by assumption. We thus have $s \leftrightarrow_{\mathcal{E}_\infty} t$ or $s \downarrow_{\mathcal{R}_\omega} t$ by fairness such that the claim follows from Lemma 7.8.18(1) and $\mathcal{R}_\omega \subseteq \mathcal{R}_\infty$. Otherwise, we have a variable overlap. By Lemma 7.8.15 both \mathcal{E}_ω and \mathcal{R}_ω are linear. This implies $s \rightarrow_{\mathcal{R}_\omega} \cdot \frac{\overline{}}{\mathcal{E}_\omega} \cdot r \approx \ell \leftarrow t$, so the claim follows from the inclusion $\mathcal{R}_\omega \subseteq \mathcal{R}_\infty$. □

Lemma 7.8.20. *The TRS \mathcal{R}_ω is Church-Rosser modulo \mathcal{E}_ω .* ✓

Proof. Define the ARSs \mathcal{A} and \mathcal{B} with multiset labeling as follows:

- $s \xrightarrow{M}_{\mathcal{A}} t$ if $s \xrightarrow{\{s'\}}_{\mathcal{R}_\omega} t$ and $M = \{s'\}$ for some term $s' \geq s$.
- $s \xrightarrow{M}_{\mathcal{B}} t$ if $s \xleftarrow{\{s', t'\}}_{\mathcal{E}_\omega} t$ and $M = \{s', t'\}$ for some terms $s' \geq s$ and $t' \geq t$.

By equipping them with the well-founded order $>_{\text{mul}}$ Lemmata 7.8.19 and 7.8.18 imply the condition of peak decreasingness modulo. Hence, Lemma 7.2.7 applies. □

For a run of KB_1 we call $(\mathcal{E}_\omega, \mathcal{R}_\omega)$ *simplified* if \mathcal{R}_ω is reduced and \mathcal{E}_ω is irreducible with respect to \mathcal{R}_ω and does not contain trivial equations. From now on we assume that $(\mathcal{E}_\omega, \mathcal{R}_\omega)$ is simplified. This allows us to establish relationships between \mathcal{R} -normal forms and normal forms with respect to the result of the linear completion run.

Lemma 7.8.21. *The inclusion $\text{NF}(\mathcal{R}) \subseteq \text{NF}(\mathcal{E}_\omega^\pm) \cap \text{NF}(\mathcal{R}_\omega)$ holds.* ✓

7.8. Completeness Results for Ordered Completion

Proof. Let $t \in \text{NF}(\mathcal{R})$. Assume to the contrary that $t \rightarrow u$ for some term u by applying an equation $\ell \approx r \in \mathcal{E}_\omega^\pm \cup \mathcal{R}_\omega$ from left to right. Because \mathcal{R} is a complete presentation of $\mathcal{E}_\omega \cup \mathcal{R}_\omega$, we have $\ell \downarrow_{\mathcal{R}} r$. Since $t \in \text{NF}(\mathcal{R})$ implies $\ell \in \text{NF}(\mathcal{R})$, we obtain $r \rightarrow_{\mathcal{R}}^* \ell$. If $\ell \approx r \in \mathcal{R}_\omega$ this contradicts $\mathcal{R}_\omega \subseteq >$, otherwise $\ell \approx r \in \mathcal{E}_\omega^\pm$ and $r \rightarrow_{\mathcal{R}}^* \ell$ contradict unorientability and non-triviality of \mathcal{E}_ω , which hold by the assumption that \mathcal{E}_ω is simplified. \square

Lemma 7.8.22. *The inclusion $\text{NF}(\mathcal{R}_\omega) \subseteq \text{NF}(\mathcal{R})$ holds.* ✓

Proof. We show that $\ell \rightarrow_{\mathcal{R}_\omega}^+ r$ for every $\ell \rightarrow r \in \mathcal{R}$, which is sufficient to prove the claim. Let $\ell \rightarrow r \in \mathcal{R}$. By Lemma 7.8.20 we have

$$\ell \xrightarrow[\mathcal{R}_\omega]{*} u \xleftrightarrow[\mathcal{E}_\omega]{*} v \xleftarrow[\mathcal{R}_\omega]{*} r$$

for some terms u and v . Since \mathcal{R} is reduced, $r \in \text{NF}(\mathcal{R})$. According to Lemma 7.8.21, both $r \in \text{NF}(\mathcal{E}_\omega^\pm)$ and $r \in \text{NF}(\mathcal{R}_\omega)$ hold. Hence $r = v$ follows from $r \rightarrow_{\mathcal{R}_\omega}^* v$ and $u \xleftrightarrow[\mathcal{E}_\omega]{*} v = r$ implies $u = v$. Therefore $\ell \rightarrow_{\mathcal{R}_\omega}^* r$. Since \mathcal{R} is terminating, $\ell = r$ is impossible and thus $\ell \rightarrow_{\mathcal{R}_\omega}^+ r$ as desired. \square

As in the previous section, the last result allows us to establish the main completeness theorem.

Theorem 7.8.23. *If $(\mathcal{E}_\omega, \mathcal{R}_\omega)$ is simplified then $\mathcal{E}_\omega = \emptyset$ and \mathcal{R}_ω is literally similar to \mathcal{R} .* ✓

Proof. The TRS \mathcal{R} is complete, the TRS \mathcal{R}_ω is terminating, and the inclusion $\rightarrow_{\mathcal{R}_\omega} \subseteq \xleftrightarrow[\mathcal{R}]{*}$ holds because \mathcal{R} is a complete presentation. Moreover, $\text{NF}(\mathcal{R}_\omega) \subseteq \text{NF}(\mathcal{R})$ by Lemma 7.8.22. Hence, Lemma 7.4.4(2) applies. Since \mathcal{R}_ω is a complete presentation, $\mathcal{E}_\omega = \emptyset$ by the assumption of a simplified system. \square

Example 7.8.24. *By Theorem 7.8.23 any simplifying KB_1 run on the equational system \mathcal{E} and the reduction order $>_{\text{lpo}}$ from Example 7.3.9 will result in a canonical presentation, independent of the order in which inference steps are applied. Note that Theorem 7.8.10 does not apply since the given order $>_{\text{lpo}}$ is not ground total.*

We conclude the subsection by showing the absence of a complete presentation for the equational system mentioned in the first paragraph of Section 7.7.

Example 7.8.25. *Let \mathcal{E} be the ES consisting of the two equations $0 + x \approx x$ and $x + y \approx y + x$. We show that \mathcal{E} admits no complete presentation. Assume to the contrary that \mathcal{R} is a complete presentation of \mathcal{E} . We use $\rightarrow_{\mathcal{R}}^+$ as the reduction order $>$ for KB_1 . Because $0 + x \xleftrightarrow[\mathcal{E}]{*} x$ implies $0 + x \downarrow_{\mathcal{R}} x$ and $x \in \text{NF}(\mathcal{R})$, we have $0 + x > x$. In the same*

7. Abstract Completion, Formalized

way $x + 0 > x$ is derived. Therefore, the following fair run of KB_1 is constructed:

$$\begin{aligned}
(\mathcal{E}, \emptyset) &\vdash_{\text{orient}}^{} (\{x + y \approx y + x\}, \{0 + x \rightarrow x\}) \\
&\vdash_{\text{deduce}}^{} (\{x + y \approx y + x, x + 0 \approx x\}, \{0 + x \rightarrow x\}) \\
&\vdash_{\text{orient}}^{} (\{x + y \approx y + x\}, \{0 + x \rightarrow x, x + 0 \rightarrow x\}) \\
&\vdash_{\text{deduce}}^{} (\{x + y \approx y + x, 0 \approx 0\}, \{0 + x \rightarrow x, x + 0 \rightarrow x\}) \\
&\vdash_{\text{delete}}^{} (\{x + y \approx y + x\}, \{0 + x \rightarrow x, x + 0 \rightarrow x\}) \\
&\vdash_{\text{deduce}}^{} (\{x + y \approx y + x, 0 \approx 0\}, \{0 + x \rightarrow x, x + 0 \rightarrow x\}) \\
&\vdash_{\text{delete}}^{} \dots
\end{aligned}$$

It is easy to check that this run is fair; the two non-trivial critical pairs $x + 0 \approx x$ and $0 + x \approx x$ belong to \mathcal{E}_∞^\pm . We have $\mathcal{E}_\omega = \{x + y \approx y + x\}$ and $\mathcal{R}_\omega = \{0 + x \rightarrow x, x + 0 \rightarrow x\}$. Note that $(\mathcal{E}_\omega, \mathcal{R}_\omega)$ is simplified. According to Theorem 7.8.23, the persistent set \mathcal{E}_ω must be empty. This is a contradiction and thus \mathcal{R} does not exist.

In summary, our proof of Theorem 7.7.10 resembles the approach by Devie [34], though our version is structured along several preliminary results that are of independent interest, such as Lemmata 7.8.20, 7.8.21, and 7.8.22.

7.9. Conclusion

In this paper we have presented new and formalized proofs for a number of correctness and completeness results for abstract completion, ranging from the decidable case of ground-completion to completeness results for ordered completion. By using modern abstract confluence criteria, we could avoid the use of proof orders, which had a positive effect on the Isabelle/HOL formalization.

We mention some topics for future work. Concerning completion of ground systems, the literature contains other interesting results that we might consider as target for future formalization efforts. Gallier et al. [40] showed that every ground ES \mathcal{E} can be transformed into an equivalent canonical TRS in $O(n^3)$ time, where n is the combined size of the terms appearing in \mathcal{E} . Snyder [128] improved this result to an $O(n \log n)$ time algorithm. Moreover, his algorithm can enumerate all canonical presentations, of which there are at most 2^k [128, Theorem 4.7], where k is the number of equations in \mathcal{E} . Furthermore, all canonical presentations have the same number of rules.

In the context of ordered completion, completeness remains an open problem in the general case: It is unknown whether an ordered completion run can find a complete system \mathcal{R} for a set of input equations \mathcal{E} if neither \mathcal{E} is linear (Theorem 7.8.23) nor \mathcal{R} is compatible with a ground-total reduction order (Theorem 7.8.10).

There are several important extensions of completion that we did not consider in this paper. We mention completion in the presence of associative and commutative (AC) symbols [117], normalized completion [81, 175], as well as maximal completion [69]. They are natural candidates for future formalization efforts.

Acknowledgement

We are grateful for the detailed comments by the anonymous reviewers, which greatly helped us to improve the paper.

8. Certified Equational Reasoning via Ordered Completion

Publication Details

Christian Sternagel and Sarah Winkler. Certified Equational Reasoning via Ordered Completion. In *Proceedings of the 27th International*, volume 11716 of *Lecture Notes in Computer Science*, pages 508–528, Springer, 2019
[doi:10.1007/978-3-030-29436-6_30](https://doi.org/10.1007/978-3-030-29436-6_30)

Abstract

On the one hand, equational reasoning is a fundamental part of automated theorem proving with ordered completion as a key technique. On the other hand, the complexity of corresponding, often highly optimized, automated reasoning tools makes implementations inherently error-prone. As a remedy, we provide a formally verified certifier for ordered completion based techniques. This certifier is code generated from an accompanying Isabelle/HOL formalization of ordered rewriting and ordered completion incorporating an advanced ground joinability criterion. It allows us to rigorously validate generated proof certificates from several domains: ordered completion, satisfiability in equational logic, and confluence of conditional term rewriting.

8.1. Introduction

Equational reasoning constitutes a main area of automated theorem proving in which completion has evolved as a fundamental technique [71]. Completion aims to transform a given set of equations into a terminating and confluent rewrite system that induces the same equational theory. Thus, on success, such a rewrite system can be used to decide equivalence of terms with respect to the initial set of equations. The original completion procedure may fail due to unorientable equations. As a remedy to this problem, ordered completion—also known as unfailing completion—was developed [14]. As the name suggests, unfailing completion always yields a result (which may however be infinite and thus take infinitely many inference steps to compute). This time, the result is an ordered rewrite system (given by a ground total reduction order, a set of rules which are oriented with respect to this order, and a set of equations) that is still terminating, but in general only ground confluent (that is, confluent on ground terms). Thus, the resulting system can be used to decide equivalence of *ground* terms with respect to the initial set of equations. This suffices for many practical purposes: A well-known success

8. Certified Equational Reasoning via Ordered Completion

story of ordered completion is the solution of the long-standing Robbins conjecture [85], followed by applications to other problems from (Boolean) algebra [86]. More recent applications include the use of ordered completion in algebraic data integration [124] and in confluence proofs of conditional term rewrite systems [155].

As an introductory example, let us illustrate ordered completion on the following set of equations describing a group where all elements are self-inverse:

$$f(x, y) \approx f(y, x) \quad f(x, f(y, z)) \approx f(f(x, y), z) \quad f(x, x) \approx 0 \quad f(x, 0) \approx x$$

Using ordered completion, the tool **MædMax** [176] transforms it into the following rules (\rightarrow) and equations (\approx), together with a suitable ground total reduction order $>$ that orients all rules from left to right.

$$\begin{array}{ll} f(x, f(x, y)) \rightarrow f(0, y) & f(x, f(y, x)) \rightarrow f(0, y) \quad f(x, x) \rightarrow 0 \quad f(x, 0) \rightarrow x \\ f(f(x, y), z) \rightarrow f(x, f(y, z)) & f(0, x) \rightarrow x \\ f(x, f(y, z)) \approx f(y, f(x, z)) & f(x, y) \approx f(y, x) \end{array}$$

This ordered rewrite system can be used to decide a given equation between ground terms, by checking whether the unique normal forms (with respect to ordered rewriting using $>$) of both terms coincide.

Automated reasoning tools are highly sophisticated pieces of software, not only because they implement complex calculi, but also due to their high degree optimization. Consequently, their implementation is inherently error-prone.

To improve their trustability we follow a two-staged certification approach and (1) add the relevant concepts and results regarding ordered completion to a formal library using the proof assistant Isabelle/HOL [103] (version Isabelle2019), and from there (2) code generate [52] a trusted certifier that is correct by construction. Our formalization strengthens the originally proposed procedure [14] by using a relaxed version of the inference system, while incorporating a stronger ground joinability criterion [83]. Our certifier allows us to rigorously validate generated proof certificates from several domains: ordered completion, satisfiability in equational logic, and confluence of conditional term rewriting.

More specifically, our contributions are as follows:


- We extend the existing *Isabelle Formalization of Rewriting*¹ (IsaFoR for short) by ordered rewriting and a generalization of the ordered completion calculus oKB [14], and prove the latter correct for finite completion runs with respect to ground total reduction orders (Section 8.3).
- We establish ground totality of the Knuth-Bendix order and the lexicographic path order in IsaFoR (Section 8.3).
- We formalize two criteria for ground joinability [14, 83] known from the literature, that allow us to apply our previous results to concrete completion runs (Section

¹<http://cl-informatik.uibk.ac.at/isafor>

8.4). In fact, we present a slightly more powerful version of the latter, and fix an error in its proof, as described below.

- We apply ordered completion to satisfiability in equational logic and infeasibility of conditions in conditional rewriting (Section 8.5).
- We extend the XML-based *certification problem format* (CPF for short) [144] by certificates for ordered completion and formalize corresponding executable check functions that verify the supplied derivations (Section 8.6).
- Finally, we extend the completion tool **MædMax** [176], as well as the confluence tool **ConCon** [155] by certificate generation and evaluate our approach on existing benchmarks (Section 8.7).

As a result, **CeTA** (the certifier accompanying **IsaFoR**) can now certify (a) ordered completion proofs and (b) satisfiability proofs of equational logic produced by the tool **MædMax**, as well as (c) conditional confluence proofs by **ConCon** where infeasibility of critical pairs is established via equational logic. To the best of our knowledge, **CeTA** constitutes the first proof checker in all of these domains.

In the remainder we provide hyperlinks (marked by ) to an HTML rendering of our formalization.

This work is an extension of an earlier workshop paper [151]. Further note that the **IsaFoR** formalization of the results in this paper is, apart from very basic results on (ordered) rewriting, entirely disjoint from our previous formalization together with Hirokawa and Middeldorp [59]. On the one hand, we consider a relaxed completion inference system where more inferences are allowed. This is possible since we are only interested in finite completion runs. On the other hand, we employ a stronger ground joinability criterion. Another major difference is that our new formalization enables actual certification of ordered completion based techniques, which is not the case for our work with Hirokawa and Middeldorp.

8.2. Preliminaries

In the sequel, we use standard notation from term rewriting [9]. Let $\mathcal{T}(\mathcal{F}, \mathcal{V})$ denote the *set of all terms* over a signature \mathcal{F} and an infinite set of variables \mathcal{V} , and $\mathcal{T}(\mathcal{F})$ the *set of all ground terms* over \mathcal{F} (that is, terms without variables). A *substitution* σ is a mapping from variables to terms. As usual, we write $t\sigma$ for the *application* of σ to the term t . A *variable permutation* (or *renaming*) π is a bijective substitution such that $\pi(x) \in \mathcal{V}$ for all $x \in \mathcal{V}$. Given an equational system (ES) \mathcal{E} , we write $\mathcal{E}^{\leftrightarrow}$ to denote its *symmetric closure* $\mathcal{E} \cup \{t \approx s \mid s \approx t \in \mathcal{E}\}$. A *reduction order* is a proper and well-founded order on terms which is closed under contexts and substitutions. It is *\mathcal{F} -ground total* if it is total on $\mathcal{T}(\mathcal{F})$. In the remainder we often focus on the Knuth-Bendix order (KBO), written $>_{\text{kbo}}$, and the lexicographic path order (LPO), written $>_{\text{lpo}}$. Given a reduction order $>$ and an ES \mathcal{E} , the term rewrite system (TRS) $\mathcal{E}_>$ consists of all rules $s\sigma \rightarrow t\sigma$ such that $s \approx t \in \mathcal{E}^{\leftrightarrow}$ and $s\sigma > t\sigma$.

8. Certified Equational Reasoning via Ordered Completion


Given a reduction order $>$, an *extended overlap* consists of two variable-disjoint variants $\ell_1 \approx r_1$ and $\ell_2 \approx r_2$ of equations in $\mathcal{E}^{\leftrightarrow}$ such that $p \in \mathcal{Pos}_{\mathcal{F}}(\ell_2)$ and ℓ_1 and $\ell_2|_p$ are unifiable with most general unifier μ . An extended overlap which in addition satisfies $r_1\mu \not\approx \ell_1\mu$ and $r_2\mu \not\approx \ell_2\mu$ gives rise to the *extended critical pair* $\ell_2[r_1]_p\mu \approx r_2\mu$. The set $\text{CP}_{>}(\mathcal{E})$ consists of all extended critical pairs between equations in \mathcal{E} . A relation on terms is (*ground*) *complete*, if it is terminating and confluent (on ground terms). A TRS \mathcal{R} is (*ground*) *complete* whenever the induced rewrite relation $\rightarrow_{\mathcal{R}}$ is. Finally, we say that a TRS \mathcal{R} is a presentation of an ES \mathcal{E} , whenever $\leftrightarrow_{\mathcal{E}}^* = \leftrightarrow_{\mathcal{R}}^*$ (that is, their equational theories coincide).

A substitution σ is *grounding* for a term t if $\sigma(x) \in \mathcal{T}(\mathcal{F})$ for all $x \in \text{Var}(t)$. Two terms s and t are called *ground joinable* over a rewrite system \mathcal{R} , denoted $s \downarrow_{\mathcal{R}}^g t$ if $s\sigma \downarrow_{\mathcal{R}} t\sigma$ for all substitutions σ that are grounding for s and t .

For any complete rewrite relation \rightarrow , we denote the (necessarily unique) *normal form* of a term t (that is, the term u such that we have $t \rightarrow^* u$ but $u \not\rightarrow v$ for all terms v) by $t\downarrow$. By an *ordered rewrite system* we mean a pair $(\mathcal{E}, \mathcal{R})$, consisting of an ES \mathcal{E} and a TRS \mathcal{R} , together with a reduction order $>$. Then, *ordered rewriting* is rewriting with respect to the TRS $\mathcal{R} \cup \mathcal{E}_{>}$. Note that ordered rewriting is always terminating if $\mathcal{R} \subseteq >$. Take commutativity $x * y \approx y * x$ for example, which causes nontermination when used as a rule in a TRS. Nevertheless, the ordered rewrite system $(\{x * y \approx y * x\}, \emptyset)$ together with KBO, say with precedence $* > \mathbf{a} > \mathbf{b}$, is terminating and we can for example rewrite $\mathbf{a} * \mathbf{b}$ to $\mathbf{b} * \mathbf{a}$ since applying the substitution $\{x \mapsto \mathbf{a}, y \mapsto \mathbf{b}\}$ to the commutativity equation results in a KBO-oriented instance.

8.3. Formalized Ordered Completion

Ordered completion is commonly presented as a set of inference rules, parameterized by a fixed reduction order $>$. This way of presentation conveniently leaves a lot of freedom to implementations. We use the following inference system, with some differences to the original formulation [14] that we discuss below.

Definition 8.3.1 (Ordered Completion ). *The inference system oKB of ordered completion operates on pairs $(\mathcal{E}, \mathcal{R})$ of equations \mathcal{E} and rules \mathcal{R} over a common signature \mathcal{F} . It consists of the following inference rules, where \mathcal{S} abbreviates $\mathcal{R} \cup \mathcal{E}_{>}$ and π is a renaming.*

$$\begin{array}{ll}
\text{deduce} & \frac{\mathcal{E}, \mathcal{R}}{\mathcal{E} \cup \{s\pi \approx t\pi\}, \mathcal{R}} \quad \text{if } s \xleftarrow{\mathcal{R} \cup \mathcal{E}^{\leftrightarrow}} \cdot \rightarrow t \quad \text{compose} \quad \frac{\mathcal{E}, \mathcal{R} \uplus \{s \rightarrow t\}}{\mathcal{E}, \mathcal{R} \cup \{s\pi \rightarrow u\pi\}} \quad \text{if } t \rightarrow_{\mathcal{S}} u \\
\text{orient} & \frac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E}, \mathcal{R} \cup \{s\pi \rightarrow t\pi\}} \quad \text{if } s > t \quad \text{simplify} \quad \frac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E} \cup \{u\pi \approx t\pi\}, \mathcal{R}} \quad \text{if } s \rightarrow_{\mathcal{S}} u \\
& \frac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E}, \mathcal{R} \cup \{t\pi \rightarrow s\pi\}} \quad \text{if } t > s \quad \frac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E} \cup \{s\pi \approx u\pi\}, \mathcal{R}} \quad \text{if } t \rightarrow_{\mathcal{S}} u \\
\text{delete} & \frac{\mathcal{E} \uplus \{s \approx s\}, \mathcal{R}}{\mathcal{E}, \mathcal{R}} \quad \text{collapse} \quad \frac{\mathcal{E}, \mathcal{R} \uplus \{t \rightarrow s\}}{\mathcal{E} \cup \{u\pi \approx s\pi\}, \mathcal{R}} \quad \text{if } t \rightarrow_{\mathcal{S}} u
\end{array}$$

8.3. Formalized Ordered Completion

We write $(\mathcal{E}, \mathcal{R}) \vdash (\mathcal{E}', \mathcal{R}')$ if $(\mathcal{E}', \mathcal{R}')$ is obtained from $(\mathcal{E}, \mathcal{R})$ by employing one of the above inference rules. A finite sequence of inference steps

$$(\mathcal{E}_0, \emptyset) \vdash (\mathcal{E}_1, \mathcal{R}_1) \vdash \dots \vdash (\mathcal{E}_n, \mathcal{R}_n)$$

is called a *run*. Definition 8.3.1 differs from the original formulation of ordered completion [14] (as well as the formulation in our previous work together with Hirokawa and Middeldorp [59]) in two ways. First, **collapse** and **simplify** do not have an encompassment condition.² This omission is possible since we only consider *finite* runs. Second, we allow variants of rules and equations to be added. This relaxation tremendously simplifies certificate generation in tools, where facts are renamed upon generation to avoid the maintenance and processing of many renamed versions of the same equation or rule. Also note that the **deduce** rule admits the addition of equations that originate from arbitrary peaks. In practice, tools usually limit its application to extended critical pairs.

The following two results establish that the rules resulting from a finite oKBrun are oriented by the reduction order $>$ and that the induced equational theories before and after completion coincide.

Lemma 8.3.2 (✓). *If $(\mathcal{E}, \mathcal{R}) \vdash^* (\mathcal{E}', \mathcal{R}')$ then $\mathcal{R} \subseteq >$ implies $\mathcal{R}' \subseteq >$.* □

Lemma 8.3.3 (✓). *If $(\mathcal{E}, \mathcal{R}) \vdash^* (\mathcal{E}', \mathcal{R}')$ then $\leftrightarrow_{\mathcal{E} \cup \mathcal{R}}^* = \leftrightarrow_{\mathcal{E}' \cup \mathcal{R}'}^*$.* □

If the employed reduction order is \mathcal{F} -ground total then the above two results imply the following conversion equivalence involving *ordered rewriting* with respect to the final system.

Lemma 8.3.4 (✓). *Suppose $>$ is \mathcal{F} -ground total and $\mathcal{R} \subseteq >$. If $(\mathcal{E}, \mathcal{R}) \vdash^* (\mathcal{E}', \mathcal{R}')$ such that $\mathcal{E}', \mathcal{R}'$, and $>$ are over the signature \mathcal{F} , then $\leftrightarrow_{\mathcal{E} \cup \mathcal{R}}^* = \leftrightarrow_{\mathcal{E}' \cup \mathcal{R}'}^*$ holds for conversions between terms in $\mathcal{T}(\mathcal{F})$.* □

This result is a key ingredient to our correctness results in Section 8.4. In order to apply it, however, we need ground total reduction orders. To this end, we formalized the following two results in IsaFoR.

Lemma 8.3.5 (✓). *If $>$ is a total precedence on \mathcal{F} then $>_{\text{kbo}}$ is \mathcal{F} -ground total.* □

Lemma 8.3.6 (✓). *If $>$ is a total precedence on \mathcal{F} then $>_{\text{lpo}}$ is \mathcal{F} -ground total.* □

In addition, we proved that for any given KBO $>_{\text{kbo}}$ (LPO $>_{\text{lpo}}$) defined over a total precedence $>$ there exists a minimal constant, that is, a constant c such that $t \geq_{\text{kbo}} c$ ($t \geq_{\text{lpo}} c$) holds for all $t \in \mathcal{T}(\mathcal{F})$ (which will be needed in Section 8.4). In earlier work by Becker et al. [15] ground totality of a lambda-free higher-order variant of KBO is formalized in Isabelle/HOL. However, for our purposes it makes sense to work with the definition of KBO that is already widely used in IsaFoR.

²The encompassment condition demands that if a rule or equation $\ell \approx r$ is used to rewrite a term $t = C[\ell\sigma]$ then C is non-empty or σ is not a renaming.

8. Certified Equational Reasoning via Ordered Completion

By Lemma 8.3.4, any two ground terms convertible in the initial equational theory are convertible with respect to ordered rewriting in the system obtained from an `oKBrun`. The remaining key issue is to decide when the current ordered rewrite system is ground confluent, such that a tool implementing `oKB` can stop. Instead of defining a fairness criterion as done by Bachmair et al. [14], we use the following criterion for correctness involving ground joinability.

Lemma 8.3.7 (✓). *If for all equations $s \approx t$ in \mathcal{E} we have $s > t$ or $t \approx s$ in \mathcal{E} and $\text{CP}_{>}(\mathcal{E}) \subseteq \downarrow_{\mathcal{E}_{>}}^g$ then \mathcal{E} is ground confluent with respect to $>$. \square*

Note that the symmetry condition on \mathcal{E} above is just a convenient way to express the split of \mathcal{E} into rewrite rules with fixed orientation, and equations applicable in both directions, which allows us to treat an ordered rewrite system as a single set of equations. Lemmas 8.3.4 and 8.3.7 combine to the following correctness result.

Corollary 8.3.8 (✓). *If $>$ is \mathcal{F} -ground total and $(\mathcal{E}_0, \emptyset) \vdash^* (\mathcal{E}, \mathcal{R})$ such that \mathcal{E}' , \mathcal{R}' , and $>$ are over the signature \mathcal{F} and $\text{CP}_{>}(\mathcal{R} \cup \mathcal{E}^{\leftrightarrow}) \subseteq \downarrow_{\mathcal{R} \cup \mathcal{E}_{>}^{\leftrightarrow}}^g$, then $\mathcal{S} = \mathcal{R} \cup \mathcal{E}_{>}^{\leftrightarrow}$ is ground complete and $\leftrightarrow_{\mathcal{E}_0}^* = \leftrightarrow_{\mathcal{S}}^*$ holds for conversions between terms in $\mathcal{T}(\mathcal{F})$.*

Before we can apply this result in order to obtain ground completeness we need to be able to discharge its ground joinability assumption on extended critical pairs. This is the topic of the next section.

8.4. Formalized Ground Joinability Criteria

In general, ground joinability is undecidable even for terminating rewrite systems [68]. Below, we formalize two sufficient criteria.

8.4.1. A Simple Criterion

We start with the criterion that Bachmair et al. [14] proposed when they introduced ordered completion.

Lemma 8.4.1 (✓). *Suppose $>$ is a ground total reduction order over \mathcal{F} with a minimal constant. Then, $\mathcal{E}_{>}$ is \mathcal{F} -ground complete whenever for all $s \approx t \in \text{CP}_{>}(\mathcal{E}^{\leftrightarrow})$ it holds that $s \downarrow_{\mathcal{E}_{>}} t$, or $s \approx t = (s' \approx t')\sigma$ for some $s' \approx t' \in \mathcal{E}^{\leftrightarrow}$. \square*

A minimal constant c is needed to turn arbitrary ordered rewrite steps into ordered rewrite steps over $\mathcal{T}(\mathcal{F})$: when performing an ordered rewrite step using an equation $u \approx v$ with $V = \text{Var}(v) \setminus \text{Var}(u) \neq \emptyset$, a step over $\mathcal{T}(\mathcal{F})$ is obtained by instantiating all variables in V to c . We illustrate the criterion on an example.

Example 8.4.2. *The following equational system \mathcal{E}_0 is derived by `ConCon` while checking infeasibility of a critical pair of the conditional rewrite system `Cops #361`:*

$$\begin{array}{lll}
 x \div y \approx \langle 0, y \rangle & x \div y \approx \langle s(q), r \rangle & x - 0 \approx x \\
 0 - y \approx 0 & s(x) - s(y) \approx x - y & s(x) > s(y) \approx x > y \\
 s(x) > 0 \approx \text{true} & s(x) \leq s(y) \approx x \leq y & 0 \leq x \approx \text{true}
 \end{array}$$

In an ordered completion run, MædMax transforms \mathcal{E}_0 into the following rules \mathcal{R} and equations \mathcal{E} :

$$\begin{array}{lll}
x - 0 \rightarrow x & 0 - x \rightarrow 0 & s(x) - s(y) \rightarrow x - y \\
0 \leq x \rightarrow \text{true} & s(x) \leq s(y) \rightarrow x \leq y & x \div y \rightarrow \langle 0, y \rangle \\
s(x) > 0 \rightarrow \text{true} & s(x) > s(y) \rightarrow x > y & \\
\langle s(x), y \rangle \approx \langle s(q), r \rangle & \langle 0, y \rangle \approx \langle s(q), r \rangle & \langle 0, x \rangle \approx \langle 0, y \rangle
\end{array}$$

Ground confluence of this system can be established by means of Lemma 8.4.1. For example, the extended overlap between the first two equations gives rise to the extended critical pair $\langle 0, y \rangle \approx \langle s(x), y \rangle$, which is just an instance of the second equation (and similarly for the other extended critical pairs).

8.4.2. Ground Joinability via Order Closures

The criterion discussed in Section 8.4.1 is rather weak. For instance, it cannot handle associativity and commutativity, as illustrated next [83, Example 1.1].

Example 8.4.3. Consider the system \mathcal{E} consisting of the three equations

$$(1) \quad (x * y) * z \approx x * (y * z) \quad (2) \quad x * y \approx y * x \quad (3) \quad x * (y * z) \approx y * (x * z)$$

and the reduction order $>_{\text{kbo}}$ with $w_0 = 1$ and $w(*) = 0$. The first equation can be oriented from left to right, whereas the other ones are unorientable.

We obtain the following extended critical peak from equations (2) and (1):

$$z * (x * y) \leftarrow (x * y) * z \rightarrow x * (y * z)$$

The resulting extended critical pair is neither an instance of an equation in \mathcal{E} nor joinable. Thus the criterion of Lemma 8.4.1 does not apply.

However, this extended critical pair is ground joinable, which we show in the following. The reduction order $>_{\text{kbo}}$ is contained in an \mathcal{F}' -ground total one on any extension of the signature $\mathcal{F}' \supseteq \mathcal{F}$ (using the well-order theorem and incrementality of KBO). Thus, for any grounding substitution σ the terms $x\sigma$, $y\sigma$, and $z\sigma$ are totally ordered. Suppose for instance that $x\sigma > z\sigma > y\sigma$. Then there is an ordered rewrite sequence witnessing joinability:

$$\begin{array}{c}
z\sigma * (x\sigma * y\sigma) \qquad \qquad \qquad x\sigma * (y\sigma * z\sigma) \\
\swarrow (2) \quad \quad \quad \searrow (2) \\
z\sigma * (y\sigma * x\sigma) \qquad \quad y\sigma * (x\sigma * z\sigma) \\
\swarrow (3) \quad \quad \quad \searrow (3) \\
y\sigma * (z\sigma * x\sigma)
\end{array}$$

If, on the other hand, $x\sigma = y\sigma > z\sigma$ holds, there is a joining sequence as well:

$$\begin{array}{c}
x\sigma * (x\sigma * z\sigma) = x\sigma * (y\sigma * z\sigma) \\
\swarrow (2) \\
x\sigma * (z\sigma * x\sigma) \\
\swarrow (3) \\
z\sigma * (x\sigma * y\sigma) = z\sigma * (x\sigma * x\sigma)
\end{array}$$

8. Certified Equational Reasoning via Ordered Completion

By ensuring the existence of a joining sequence for all possible relationships between $x\sigma$, $y\sigma$, and $z\sigma$, ground joinability can be established. Using this approach to show that all extended critical pairs are ground joinable, it can be verified that \mathcal{E} is in fact ground complete.

The ground joinability test by Martin and Nipkow [83] is based on the idea illustrated in Example 8.4.3 above: perform a case analysis by considering ordered rewriting using all extensions of $>$ to instantiations of variables. Below, we give the corresponding formal definitions used in `IsaFoR`. For any relation R on terms, let $\sigma(R)$ denote the relation such that $s\sigma(R)t$ holds if and only if $s R t$.

Definition 8.4.4 (✓). A closure \mathcal{C} is a mapping between relations on terms that satisfies the following properties:

- (1) If $s \mathcal{C}(R) t$ then $s\sigma \mathcal{C}(\sigma(R)) t\sigma$, for all relations R , substitutions σ , and terms s and t .
- (2) If $R \subseteq R'$ then $\mathcal{C}(R) \subseteq \mathcal{C}(R')$, for all relations on terms R and R' .

The closure \mathcal{C} is compatible with a relation on terms R if $\mathcal{C}(R) \subseteq R$ holds.

In the remainder of this section we assume \mathcal{F} to be the signature of the input problem, we consider an \mathcal{F} -ground total reduction order $>$ as well as a closure \mathcal{C} that is compatible with $>$. Furthermore, we assume for every finite set of variables $V \subseteq \mathcal{V}$ and every equivalence relation \equiv on V a representation function $\text{rep}_{\mathcal{E} \cup \mathcal{R}}$ such that for any $x \in V$ we have $x \equiv \text{rep}_{\mathcal{E} \cup \mathcal{R}}(x)$, $\text{rep}_{\mathcal{E} \cup \mathcal{R}}(x) \in V$ and $x \equiv y$ implies $\text{rep}_{\mathcal{E} \cup \mathcal{R}}(x) = \text{rep}_{\mathcal{E} \cup \mathcal{R}}(y)$. Given an equivalence relation \equiv on V , let $\hat{\equiv}$ denote the substitution such that $\hat{\equiv}(x) = \text{rep}_{\mathcal{E} \cup \mathcal{R}}(x)$ for all $x \in V$.

Definition 8.4.5 (✓). Given an ES \mathcal{E} and a reduction order $>$, terms s and t are \mathcal{C} -joinable, written $s \downarrow_{\mathcal{E}}^{\mathcal{C}} t$, if for all equivalence relations \equiv on $\text{Var}(s, t)$ and every order \succ on the equivalence classes of \equiv it holds that

$$s \hat{\equiv} \xrightarrow[\mathcal{E}_{\mathcal{C}(\succ)}]{*} \cdot \xleftarrow[\mathcal{E}]{\hat{\equiv}} \cdot \xleftarrow[\mathcal{E}_{\mathcal{C}(\succ)}]{*} t \hat{\equiv} \quad (8.1)$$

Example 8.4.6. For instance, consider the terms $s = z * (x * y)$ and $t = x * (y * z)$ from Example 8.4.3. One possible equivalence relation \equiv on $\text{Var}(s, t) = \{x, y, z\}$ is given by the equivalence classes $\{x, y\}$ and $\{z\}$; one possible order on these is $\hat{\equiv}(x) \succ \hat{\equiv}(z)$ (corresponding to the second example for an order on the instantiations $x\sigma$ and $z\sigma$ in Example 8.4.3). By taking \mathcal{C} to be the KBO closure (see Definition 8.4.13 below), we have $x * z \mathcal{C}(\succ) z * x$ and $x * (z * x) \mathcal{C}(\succ) z * (x * x)$. Using the ES \mathcal{E} from Example 8.4.3 we thus obtain the ordered rewrite sequence

$$t \hat{\equiv} = x * (x * z) \xrightarrow[\mathcal{E}_{\mathcal{C}(\succ)}]{} x * (z * x) \xrightarrow[\mathcal{E}_{\mathcal{C}(\succ)}]{} z * (x * x) = s \hat{\equiv}$$

Ground joinability follows from \mathcal{C} -joinability. Since this is the key result for the ground joinability criterion of this subsection, we also sketch its proof.

Lemma 8.4.7 (✓). *If $s \downarrow_{\mathcal{E}}^{\mathcal{C}} t$ then $s \downarrow_{\mathcal{E}_{>}}^{\mathcal{G}} t$.*

Proof. We assume $s \downarrow_{\mathcal{E}}^{\mathcal{C}} t$ and consider a grounding substitution σ to show $s\sigma \downarrow_{\mathcal{E}_{>}} t\sigma$. There is some equivalence relation \equiv on $\text{Var}(s, t)$ such that $x \equiv y$ holds if and only if $\sigma(x) = \sigma(y)$ for all $x, y \in \text{Var}(s, t)$. Note that this implies $s\sigma = s \hat{=} \sigma$ and $t\sigma = t \hat{=} \sigma$.

We can define an order \succ on the equivalence classes of \equiv such that $[x]_{\equiv} \succ [y]_{\equiv}$ if and only if $\sigma(x) > \sigma(y)$. Hence $\sigma(\succ) \subseteq >$ holds, and by Definition 8.4.4(2) we have $\mathcal{C}(\sigma(\succ)) \subseteq \mathcal{C}(>)$. Compatibility implies $\mathcal{C}(>) \subseteq \succ$, and thus $\mathcal{C}(\sigma(\succ)) \subseteq \succ$.

From Definition 8.4.4(1) we can show that $u \rightarrow_{\mathcal{E}_{\mathcal{C}(\succ)}} v$ implies $u\sigma \rightarrow_{\mathcal{E}_{\mathcal{C}(\sigma(\succ))}} v\sigma$ for all terms u and v . So using the assumption $s \downarrow_{\mathcal{E}}^{\mathcal{C}} t$ we can apply σ to a conversion of the form (8.1) to obtain

$$s\sigma = s \hat{=} \sigma \xrightarrow[\mathcal{E}_{\mathcal{C}(\sigma(\succ))}]{*} \cdot \xleftrightarrow[\mathcal{E}]{=} \cdot \xleftarrow[\mathcal{E}_{\mathcal{C}(\sigma(\succ))}]{*} t \hat{=} \sigma = t\sigma \quad (8.2)$$

Ordered rewriting is monotone with respect to the order, and hence $\mathcal{C}(\sigma(\succ)) \subseteq \succ$ implies $\rightarrow_{\mathcal{E}_{\mathcal{C}(\sigma(\succ))}} \subseteq \rightarrow_{\mathcal{E}_{>}}$. Thus (8.2) implies the existence of a conversion

$$s\sigma \xrightarrow[\mathcal{E}_{>}]{*} \cdot \xleftrightarrow[\mathcal{E}_{>}]{=} \cdot \xleftarrow[\mathcal{E}_{>}]{*} t\sigma$$

where the $\leftrightarrow_{\mathcal{E}_{>}}$ step exists as any two \mathcal{F} -ground terms are comparable in $>$. $\square \quad \square$

Note that the proof above uses the monotonicity assumption for closures (Definition 8.4.4(2)), which is not present in [83]. The following counterexample illustrates that monotonicity is indeed necessary.

Example 8.4.8. *Consider the ES $\mathcal{E} = \{f(x) \approx a\}$ and suppose that $> = \mathcal{C}(>)$ is an LPO with precedence $a > b > c > f$. Moreover, take $s = f(b)$ and $t = f(c)$. Any order \succ as in Definition 8.4.5 is empty since $\text{Var}(s, t) = \emptyset$. As \mathcal{C} is not required to be monotone, the relation $\mathcal{C}(\succ)$ may contain $(f(b), a)$ and $(f(c), a)$. Then $s \rightarrow_{\mathcal{E}_{\mathcal{C}(\succ)}} a$ and $t \rightarrow_{\mathcal{E}_{\mathcal{C}(\succ)}} a$ imply $s \downarrow_{\mathcal{E}}^{\mathcal{C}} t$ even though $s \downarrow_{\mathcal{E}_{>}}^{\mathcal{G}} t$ does not hold.*

Below, we define an inductive predicate **gj** which is used to conclude ground joinability of a given equation.

Definition 8.4.9 (✓). *Given an ES \mathcal{E} and a reduction order $>$, **gj** is defined inductively by the following rules:*

delete	$\text{gj}(t, t)$
closure	$s \downarrow_{\mathcal{E}}^{\mathcal{C}} t \implies \text{gj}(s, t)$
step	$s \leftrightarrow_{\mathcal{E}} t \implies \text{gj}(s, t)$
rewrite left	$s \xrightarrow[\mathcal{E}_{>}]{} u \text{ and } \text{gj}(u, t) \implies \text{gj}(s, t)$
rewrite right	$t \xrightarrow[\mathcal{E}_{>}]{} u \text{ and } \text{gj}(s, u) \implies \text{gj}(s, t)$
congruence	$\text{gj}(s_i, t_i) \text{ for all } 1 \leq i \leq n \implies \text{gj}(f(s_1, \dots, s_n), f(t_1, \dots, t_n))$

8. Certified Equational Reasoning via Ordered Completion

This test differs from the one due to Martin and Nipkow [83] by the two **rewrite** rules, which were added to allow for more efficient checks, as illustrated next.

Example 8.4.10. Consider the ES \mathcal{E}

$$f(x) \approx f(y)$$

$$g(x, y) \approx f(x)$$

together with a KBO that can orient the second equation (for instance, one can take as precedence $g > f > c$ and let all function symbol weights as well as w_0 be 1). Then $gj(f(x), f(z))$ holds by the **step** rule, $gj(g(x, y), f(z))$ follows by an application of **rewrite left**, and $gj(g(x, y), g(z, w))$ by **rewrite right**. By Lemma 8.4.11 below it thus follows that the equation $g(x, y) \approx g(z, w)$ is ground joinable.

However, the criterion by Martin and Nipkow [83] lacks the **rewrite** steps. Hence ground joinability of $g(x, y) \approx g(z, w)$ can only be established by applying the **closure** rule. This amounts to checking ground joinability with respect to 81 relations between the four variables. Since the number of variable relations is in general exponential, the criterion stated in Definition 8.4.9 can in practice be exponentially more efficient than the test by Martin and Nipkow [83].

Using Lemma 8.4.7 it is not hard to show the following correctness results.

Lemma 8.4.11 (✓). Suppose for all $s \approx t$ in \mathcal{E} we have $s > t$ or $t \approx s$ in \mathcal{E} . Then $gj(s, t)$ implies $s \downarrow_{\mathcal{E}}^g t$. \square

Lemma 8.4.12 (✓). If for all $s \approx t$ in \mathcal{E} we have $s > t$ or $t \approx s$ in \mathcal{E} and $CP_{>}(\mathcal{E}) \subseteq \downarrow_{\mathcal{E}}^g$ then \mathcal{E} is ground confluent with respect to $>$. \square

This test can not only handle Example 8.4.3 but also the group theoretic problem from the introduction. Moreover, it subsumes Lemma 8.4.1 since whenever for some equation $s \approx t$ we have $s \downarrow_{\mathcal{E}}^g t$ by Lemma 8.4.1 then $gj(s, t)$ holds.

Closures for Knuth-Bendix Orders.

Definition 8.4.4 requires abstract properties on closures. In the following we define closures for KBO as used in **IsaFoR/CeTA**.

Similar to the already existing definition of KBO in **IsaFoR** [142] we define the closure $>_{kbo}^R$ as follows.

Definition 8.4.13 (✓). Let R be a relation on terms, $>$ a precedence on \mathcal{F} , and (w, w_0) a weight function. The KBO closure $>_{kbo}^R$ is a relation on terms inductively defined as follows: $s >_{kbo}^R t$ if $s R t$, or $|s|_x \geq |t|_x$ for all $x \in \mathcal{V}$ and either

- (a) $w(s) > w(t)$, or
- (b) $w(s) = w(t)$ and one of
 - (1) $s \notin \mathcal{V}$ and $t \in \mathcal{V}$, or
 - (2) $s = f(s_1, \dots, s_n)$, $t = g(t_1, \dots, t_m)$ and $f > g$, or

- (3) $s = f(s_1, \dots, s_n)$, $t = f(t_1, \dots, t_n)$ and there is some $i \leq n$ such that $s_j = t_j$ for all $1 \leq j < i$ and $s_i >_{\text{kbo}}^R t_i$

Note that even though Definition 8.4.13 resembles the usual definition of KBO, it defines a *closure* of a relation R in a KBO-like way rather than a reduction order. For instance, if $x \succ z$, as in Example 8.4.6, then $x * z >_{\text{kbo}}^> z * x$ holds.

We prove that $>_{\text{kbo}}^R$ is indeed a closure that is compatible with $>_{\text{kbo}}$ based on the same weight function and precedence.

Lemma 8.4.14. *Let R be a relation on terms, $>$ a precedence on \mathcal{F} , and (w, w_0) a weight function. Then all of the following hold:*

- (a) If $s >_{\text{kbo}} t$ then $s >_{\text{kbo}}^R t$ for all terms s and t . ✓
- (b) If $R \subseteq R'$ then $>_{\text{kbo}}^R \subseteq >_{\text{kbo}}^{R'}$. ✓
- (c) If $s >_{\text{kbo}}^R t$ then $s\sigma >_{\text{kbo}}^{\sigma(R)} t\sigma$, for all substitutions σ , and terms s and t . ✓
- (d) The closure $>_{\text{kbo}}^R$ is compatible with $>_{\text{kbo}}$. ✓

8.5. Applications

Ground complete rewrite systems can be used to decide equivalence of ground terms with respect to their induced equational theory. Here we highlight applications of this decision problem.

Deciding Ground Equations.

Suppose we obtain the ordered rewrite system $(\mathcal{E}, \mathcal{R})$ and the reduction order $>$ by applying ordered completion to an initial set of equations \mathcal{E}_0 . Then it is easy to decide whether two ground terms s and t are equivalent with respect to \mathcal{E}_0 (that is, whether $s \leftrightarrow_{\mathcal{E}_0}^* t$): it suffices to check if the (necessarily unique) normal forms of s and t with respect to $\mathcal{R} \cup \mathcal{E}_>$ coincide. Also if all variables of a non-ground goal equation are universally quantified, the goal can be decided by substituting fresh constants for its variables.

Equations with Existential Variables.

The following trick by Bachmair et al. [14] allows us to reduce equations with existentially quantified variables to the ground case: Let \mathcal{E} be a set of equations and $s \approx t$ a goal equation where all variables are existentially quantified. This corresponds to the question whether there is a substitution σ such that $s\sigma \leftrightarrow_{\mathcal{E}}^* t\sigma$ holds. We employ three fresh function symbols **eq**, **true**, and **false**, and define $\mathcal{E}_{s,t}^{\text{eq}}$ to denote \mathcal{E} extended by the two equations $\text{eq}(x, x) \approx \text{true}$ and $\text{eq}(s, t) \approx \text{false}$.

If a ground complete system equivalent to $\mathcal{E}_{s,t}^{\text{eq}}$ is found—for instance discovered by ordered completion—then it can be used to decide the goal, as stated next.

8. Certified Equational Reasoning via Ordered Completion

Lemma 8.5.1 (✓). *Let s , t , and \mathcal{E} all be over signature \mathcal{F} and let \mathcal{S} be a ground complete TRS such that $\leftrightarrow_{\mathcal{E}_{s,t}}^* \subseteq \leftrightarrow_{\mathcal{S}}^*$ on $\mathcal{T}(\mathcal{F})$. If $s\sigma \leftrightarrow_{\mathcal{E}}^* t\sigma$ then $\text{true}\downarrow_{\mathcal{S}} = \text{false}\downarrow_{\mathcal{S}}$.*

Proof. Since $s\sigma \leftrightarrow_{\mathcal{E}}^* t\sigma$, there is a conversion $s\sigma \leftrightarrow_{\mathcal{E}_{s,t}}^* t\sigma$ by construction of $\mathcal{E}_{s,t}^{\text{eq}}$. Moreover, (appealing to an earlier formalization about signature extensions [137]) there exists an \mathcal{F} -grounding substitution τ such that $s\tau \leftrightarrow_{\mathcal{E}_{s,t}}^* t\tau$. So we have

$$\text{true} \xleftarrow[\mathcal{E}_{s,t}^{\text{eq}}]{} \text{eq}(s\tau, s\tau) \xleftarrow[\mathcal{E}_{s,t}^{\text{eq}}]{}^* \text{eq}(s\tau, t\tau) \xrightarrow[\mathcal{E}_{s,t}^{\text{eq}}]{} \text{false}$$

and by the assumed conversion inclusion an \mathcal{S} -conversion between true and false . Several applications of ground confluence of \mathcal{S} yield joinability of $\text{true}\downarrow_{\mathcal{S}}$ and $\text{false}\downarrow_{\mathcal{S}}$. Since both of these terms are normal forms they coincide. \square \square

Infeasibility of Conditions.

A decision procedure for ground equations can also be harnessed to prove infeasibility of conditions in conditional term rewriting. Here a condition c is a sequence of pairs of terms $s_1 \approx t_1, \dots, s_k \approx t_k$ and we say that c is infeasible whenever there is no substitution such that $s_i\sigma \rightarrow_{\mathcal{R}}^* t_i\sigma$ holds for all $1 \leq i \leq k$. Now, it is obviously a sound overapproximation to ensure that there is no σ such that $s_i\sigma \leftrightarrow_{\mathcal{R}}^* t_i\sigma$ for all $1 \leq i \leq k$. This suggests that completion methods might be applicable.

But there are still two complications before we are able to achieve an infeasibility check: (1) the rules of a conditional term rewrite system (CTRS for short) \mathcal{R} may be guarded by conditions, making \mathcal{R} an unsuitable input for ordered completion, and (2) the conditions c are most of the time not ground. As is conventional when adopting TRS methods to conditional rewriting, we solve (1) by dropping all conditions from the rules of \mathcal{R} , resulting in the unconditional TRS \mathcal{R}_u whose rewrite relation overapproximates the one of \mathcal{R} . Of course if we can establish that there is no σ such that $s_i\sigma \rightarrow_{\mathcal{R}_u}^* t_i\sigma$ for all $1 \leq i \leq k$, then we also obtain infeasibility of c with respect to the CTRS \mathcal{R} . In order to solve (2) we use a fresh function symbol \mathbf{c} and apply Lemma 8.5.1 to decide the equation $s = \mathbf{c}(s_1, \dots, s_k) \approx \mathbf{c}(t_1, \dots, t_k) = t$ by applying ordered completion to $\mathcal{R}_{u_{s,t}}^{\text{eq}}$. If $s \not\leftrightarrow_{\mathcal{R}_{u_{s,t}}^{\text{eq}}}^* t$ we can conclude infeasibility of c .

Checking for infeasibility is for example useful when analyzing the confluence of a conditional rewrite system, since whenever we encounter a conditional critical pair whose conditions are infeasible, we can ignore it entirely. Since 2019 the Confluence Competition (CoCo)³ also features a dedicated infeasibility category.

8.6. Certification

In this section we describe the proof certificates for the different certifiable properties and summarize the corresponding Isabelle/HOL check functions.

³<http://project-coco.uibk.ac.at/2019/>

Here, *check functions* are the formal connection between general, abstract results and concrete certificates. For example, a check function for a KBO termination proof takes a certificate, containing a concrete TRS, a specific precedence, and fixed weight functions, as input. It checks that the KBO instance is admissible and orients all rules of the TRS from left to right. By appealing to the abstract result that compatibility of a TRS with an admissible KBO implies termination, it then concludes termination of the concrete instance.

Only check functions that are both executable and proven sound are allowed in the certifier. The latter means that success of the check function implies a concrete instance of the corresponding general result (in our example success proves termination of the given TRS). In case of failure it is customary for **CeTA** check functions to give a human readable reason for why a certificate is rejected.

Ordered Completion Certificates.

Here, the certificate consists of

- a set of initial equations \mathcal{E}_0 ,
- an ordered completion result $(\mathcal{E}, \mathcal{R})$ together with a reduction order $>$, and
- a sequence of inference steps according to Definition 8.3.1.

The corresponding check function verifies that (1) the inference steps form a valid run $(\mathcal{E}_0\pi, \emptyset) \vdash^* (\mathcal{E}, \mathcal{R})$ for some renaming π , (2) all extended critical pairs are joinable, by default according to Lemma 8.4.12, and (3) the reduction order is admissible, in case of KBO.

Next, we illustrate such an ordered completion proof by an example.

Example 8.6.1. *The certificate corresponding to Example 8.4.2 contains the equations \mathcal{E}_0 , the resulting system $(\mathcal{E}, \mathcal{R})$, and the reduction order $>_{\text{kbo}}$ with precedence $> > \mathbf{s} > \leq > \text{true} > - > \div > \langle \cdot, \cdot \rangle > 0$, $w_0 = 1$, and $w(0) = 2$, $w(\div) = w(\text{true}) = w(\mathbf{s}) = 1$, and all other symbols having weight 0. In addition, a sequence of inference steps explains how $(\mathcal{E}, \mathcal{R})$ is obtained from \mathcal{E}_0 :*

<i>simplify_{left}</i>	$x \div y \approx \langle \mathbf{s}(q), r \rangle \text{ to } \langle 0, y \rangle \approx \langle \mathbf{s}(q), r \rangle$	
<i>deduce</i>	$\langle 0, x \rangle \leftarrow \langle \mathbf{s}(u), v \rangle \rightarrow \langle 0, y \rangle$	
<i>deduce</i>	$\langle \mathbf{s}(x), y \rangle \leftarrow \langle 0, u \rangle \rightarrow \langle \mathbf{s}(q), r \rangle$	
<i>deduce</i>	$x > y \leftarrow \mathbf{s}(x) > \mathbf{s}(y) \rightarrow \mathbf{s}(\mathbf{s}(x)) > \mathbf{s}(\mathbf{s}(y))$	
<i>orient_l</i>	$0 \leq x \rightarrow \text{true}$	
<i>orient_r</i>	$\mathbf{s}(\mathbf{s}(x)) > \mathbf{s}(\mathbf{s}(y)) \rightarrow x > y$	(\star)
...		
<i>orient_l</i>	$\mathbf{s}(x) - \mathbf{s}(y) \rightarrow x - y$	
<i>orient_l</i>	$0 - x \rightarrow 0$	
<i>orient_l</i>	$\mathbf{s}(x) \leq \mathbf{s}(y) \rightarrow x \leq y$	
<i>collapse</i>	$\mathbf{s}(\mathbf{s}(x)) > \mathbf{s}(\mathbf{s}(y)) \rightarrow x > y \text{ to } x > y \approx x > y$	
<i>collapse</i>	$\mathbf{s}(\mathbf{s}(x)) > \mathbf{s}(0) \rightarrow \text{true} \text{ to } \mathbf{s}(x) > 0 \approx \text{true}$	

8. Certified Equational Reasoning via Ordered Completion

$\text{simplify}_{\text{left}} \quad \textcolor{green}{s(x)} > 0 \approx \text{true} \text{ to } \text{true} \approx \text{true}$
 $\text{delete} \quad \textcolor{brown}{x} > \textcolor{brown}{y} \approx \textcolor{brown}{x} > \textcolor{brown}{y}$
 $\text{delete} \quad \text{true} \approx \text{true}$

The first *collapse* step using rule (\star) above illustrates our relaxed inference rule, it would not have been possible according to the original inference system [14] due to the encompassment condition since $\textcolor{green}{s(s(x))} > \textcolor{green}{s(s(y))} \not\approx \textcolor{green}{s(s(x))} > \textcolor{brown}{s(s(y))}$.

We briefly comment on the differences to the certification of standard Knuth-Bendix completion as already present in CeTA [142]. For standard completion, the certificate contains the initial set of equations \mathcal{E}_0 , the resulting TRS \mathcal{R} together with a termination proof, and stepwise \mathcal{E}_0 -conversions from ℓ to r for each rule $\ell \rightarrow r \in \mathcal{R}$. The certifier first checks the termination proof to guarantee termination of \mathcal{R} . Then, confluence of \mathcal{R} can be guaranteed by ensuring that all critical pairs are joinable. At this point it is easy to verify the inclusion $\leftrightarrow_{\mathcal{E}_0}^* \subseteq \leftrightarrow_{\mathcal{R}}^*$: for each equation $s \approx t \in \mathcal{E}_0$ the \mathcal{R} -normal forms of s and t are computed and checked for syntactic equality. The converse inclusion $\leftrightarrow_{\mathcal{R}}^* \subseteq \leftrightarrow_{\mathcal{E}_0}^*$ is taken care of by the provided \mathcal{E}_0 -conversions. Overall, we obtain that \mathcal{R} is a complete presentation of \mathcal{E}_0 without mentioning a specific inference system.

Unfortunately, the same approach does not work for ordered completion: The inclusion $\leftrightarrow_{\mathcal{E}_0}^* \subseteq \leftrightarrow_{\mathcal{R} \cup \mathcal{E}_>}^*$ cannot be established by rewriting equations in \mathcal{E}_0 to normal form, since they may contain variables but $\mathcal{R} \cup \mathcal{E}_>$ is only ground confluent. Moreover, since ground joinability is undecidable no complete check can be performed. Therefore, we instead ask for certificates that contain explicit inference steps, as described above.

Equational Satisfiability Certificates.

We use the term “satisfiability” of unit equality problems in line with the terminology of TPTP [161]: given a set of equations \mathcal{E}_0 and a ground goal inequality $s \not\approx t$, show that this axiomatization is satisfiable. To this end, completion-based tools try to find a ground complete presentation \mathcal{S} of \mathcal{E}_0 and verify that $s \downarrow_{\mathcal{S}} \neq t \downarrow_{\mathcal{S}}$.

A certificate for this application extends an ordered completion certificate by the goal terms. The corresponding check function verifies that

- the presented ordered completion proof is valid as described above,
- the goal inequality is ground,
- the signature of \mathcal{E}_0 , \mathcal{E} , and \mathcal{R} is included in the signature of $>$, and
- the terms in the goal have different normal forms.

We chose the symbols mentioned by the reduction order to be the considered signature \mathcal{F} . In comparison to picking the signature of \mathcal{E}_0 , this has the advantage that it is easy to add additional function symbols. Moreover, since KBO precedences in the CPF input are lists of function symbols, no additional checks are required to ensure \mathcal{F} -ground totality of the constructed reduction order.

As a side note, unsatisfiability proofs are much easier to certify: a tool only needs to output a conversion between the two goal terms. Support for the corresponding certificates has already been added to **CeTA** earlier [160].

Infeasibility Certificates.

Actually we check (generalized) nonreachability [135] of a target t from a source s with respect to a TRS \mathcal{R} , that is, the property that, given a TRS \mathcal{R} and two terms s and t , there is no substitution σ with $s\sigma \rightarrow_{\mathcal{R}}^* t\sigma$.

The corresponding certificates list function symbols **eq**, **true**, and **false**, together with an equational satisfiability certificate. The check function first constructs, using **eq**, **true**, and **false** from the certificate the TRS $\mathcal{R}_{s,t}^{\text{eq}}$ and then verifies that the equation **true** \approx **false** is not satisfiable according to the supplied equational satisfiability certificate with $\mathcal{R}_{s,t}^{\text{eq}}$ as initial set of equations.

8.7. Experiments

Below we summarize experiments with our certifier on different problem sets. More details are available from the accompanying website.⁴

Ordered Completion.

Martin and Nipkow [83] give 10 examples. The criterion of Lemma 8.4.12 with KBO applies to 7 of those and **MædMax** produces corresponding proofs. Six of these proofs are certified by **CeTA**. The missing example uses a trick also used by Waldmeister [8]: certain *redundant* equations need not be considered for critical pair computation. This simplification is not yet supported by **CeTA**.

We also ran **MædMax** on the 138 problems [122] for standard completion collected from the literature. Using KBO, **MædMax** can complete 55 of them, and 52 of those are certified. (Using LPO and KBO, 91 are completed.) For the three remaining (AC) group examples, **MædMax** uses a stronger criterion [174] which is currently not supported by **CeTA**. Overall, this amounts to 58% certification coverage of all ordered completion proofs by **MædMax**.

Satisfiable Unit Equality Problems.

There are 144 unit equality problems (UEQ) in the TPTP 7.2.0 [161] benchmark that are classified as satisfiable, of which **MædMax** using KBO only can prove 11. All these proofs are certified by **CeTA**. With its general strategy **MædMax** can handle 14 problems, but two of those require duplicating rules, such that KBO is not applicable, and one has multiple goals, which is currently not supported by **CeTA**.

⁴<http://cl-informatik.uibk.ac.at/experiments/okb/>

Infeasibility Problems.

There are 148 oriented CTRSs in version 807 of the Cops⁵ benchmark (that is, the version of Cops where the highest problem number is 807) of CoCo. Here *oriented* means that a condition $s \approx t$ is satisfied by a substitution σ , whenever $s\sigma \rightarrow_{\mathcal{R}}^* t\sigma$. (This is the class of systems ConCon is specialized to, hence we restrict our experiments to the above 148 systems.)

Out of those 148 CTRSs, the previous version of ConCon (1.7) can prove (non)confluence of 109 with and of 112 without certification. The new version of ConCon (1.8), extended by infeasibility checks via ordered completion with MædMax, can handle 111 CTRSs with and 114 without certification. We thus obtain two new certified proofs, namely for Cops #340 and #361.

8.8. Conclusion

We presented our Isabelle/HOL formalization of ordered completion and two accompanying ground joinability criteria—now part of IsaFoR 2.37. It comes with check functions for ordered completion proofs, equational satisfiability proofs, and infeasibility proofs for conditional term rewriting. Formalizing soundness of these check functions allowed us to add support for corresponding certificates to the certifier CeTA that is code generated from IsaFoR. To the best of our knowledge, CeTA constitutes the first proof checker for ordered completion proofs. Indeed, it already helped us to detect a soundness error in MædMax, where in certain corner cases some extended critical pairs were ignored. Our experiments show that we can certify 58% of ordered completion proofs (corresponding to 94% of the KBO proofs) and 85% of the satisfiability proofs produced by MædMax (100% for KBO). The number of certified proofs of ConCon increased by two.

Moreover, CeTA is the only certifier used in the Confluence Competition; by certifying infeasibility proofs our work thus helps to validate more tool output. Regarding the recent CoCo 2019, certification currently covers roughly 83% of the benchmarks in the two categories (CTRS and TRS) that have certified counterparts (CPF-CTRS and CPF-TRS).

In the future, we plan to add support for closures of LPO and extend our certifier to verify proofs of pure, not necessarily unit, equality formulas, as well as ground confluence proofs by tools participating in the confluence competition.

Acknowledgments.

We thank the anonymous referees for their constructive comments and various suggestions for improvements.

⁵<http://cops.uibk.ac.at?q=1..807>

Bibliography

- [1] B. Alarcón, R. Gutiérrez, S. Lucas, and R. Navarro-Marset. Proving termination properties with MU-TERM. In *Proceedings of the 13th International Conference on Algebraic Methodology And Software Technology (AMAST)*, volume 6486 of *Lecture Notes in Computer Science*, pages 201–208. Springer, 2011. doi:[10.1007/978-3-642-17796-5_12](https://doi.org/10.1007/978-3-642-17796-5_12).
- [2] E. Alkassar, S. Böhme, K. Mehlhorn, and C. Rizkallah. Verification of certifying computations. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV)*, volume 6806 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2011. doi:[10.1007/978-3-642-22110-1_7](https://doi.org/10.1007/978-3-642-22110-1_7).
- [3] S. Antoy and M. Hanus. Functional logic programming. *Commun. ACM*, 53(4): 74–85, 2010. doi:[10.1145/1721654.1721675](https://doi.org/10.1145/1721654.1721675).
- [4] T. Aoto and M. Yamaguchi. ACP: System description for CoCo 2019. In *Proceedings of the 8th International Workshop on Confluence (IWC)*, page 51, 2019.
- [5] T. Aoto, J. Yoshida, and Y. Toyama. Proving confluence of term rewriting systems automatically. In *Proceedings of the 20th International Conference on Rewriting Techniques and Applications (RTA)*, volume 5595 of *Lecture Notes in Computer Science*, pages 93–102. Springer, 2009. doi:[10.1007/978-3-642-02348-4_7](https://doi.org/10.1007/978-3-642-02348-4_7).
- [6] M. Avanzini, C. Sternagel, and R. Thiemann. Certification of complexity proofs using CeTA. In *Proceedings of the 26th International Conference on Rewriting Techniques and Applications (RTA)*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468. Springer, 2015. doi:[10.1007/978-3-642-03359-9_31](https://doi.org/10.1007/978-3-642-03359-9_31).
- [7] J. Avenhaus and C. Loria-Sáenz. On conditional rewrite systems with extra variables and deterministic logic programs. In *Proceedings of the 5th International Conference on Logic for Programming and Automated Reasoning (LPAR)*, volume 822 of *Lecture Notes in Computer Science*, pages 215–229. Springer, 1994. doi:[10.1007/3-540-58216-9_40](https://doi.org/10.1007/3-540-58216-9_40).
- [8] J. Avenhaus, T. Hillenbrand, and B. Löchner. On using ground joinable equations in equational theorem proving. *Journal of Symbolic Computation*, 36(1-2):217–233, 2003. doi:[10.1016/S0747-7171\(03\)00024-5](https://doi.org/10.1016/S0747-7171(03)00024-5).
- [9] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998. doi:[10.1017/CB09781139172752](https://doi.org/10.1017/CB09781139172752).

Bibliography

- [10] L. Bachmair. *Canonical Equational Proofs*. Birkhäuser, 1991. doi:[10.1007/978-1-4684-7118-2](https://doi.org/10.1007/978-1-4684-7118-2).
- [11] L. Bachmair and N. Dershowitz. Commutation, transformation, and termination. In *Proceedings of the 8th International Conference on Automated Deduction (CADE)*, volume 230 of *Lecture Notes in Computer Science*, pages 5–20, 1986. doi:[10.1007/3-540-16780-3_76](https://doi.org/10.1007/3-540-16780-3_76).
- [12] L. Bachmair and N. Dershowitz. Equational inference, canonical proofs, and proof orderings. *Journal of the ACM*, 41(2):236–276, 1994. doi:[10.1145/174652.174655](https://doi.org/10.1145/174652.174655).
- [13] L. Bachmair, N. Dershowitz, and J. Hsiang. Orderings for equational proofs. In *Proceedings of the 1st IEEE Symposium on Logic in Computer Science (LICS)*, pages 346–357, 1986.
- [14] L. Bachmair, N. Dershowitz, and D. A. Plaisted. Completion without failure. In *Resolution of Equations in Algebraic Structures*, volume 2: Rewriting Techniques of *Progress in Theoretical Computer Science*, pages 1–30. Academic Press, 1989.
- [15] H. Becker, J. C. Blanchette, U. Waldmann, and D. Wand. A transfinite Knuth-Bendix order for lambda-free higher-order terms. In *Proceedings of the 26th International Conference on Automated Deduction (CADE)*, volume 10395 of *Lecture Notes in Computer Science*, pages 432–453. Springer, 2017. doi:[10.1007/978-3-319-63046-5_27](https://doi.org/10.1007/978-3-319-63046-5_27).
- [16] S. Berghofer. A constructive proof of Higman’s lemma in Isabelle. In *Proceedings of the 3rd International Workshop on Types for Proofs and Programs (TYPES)*, volume 3085 of *Lecture Notes in Computer Science*, pages 66–82. Springer, 2003. doi:[10.1007/978-3-540-24849-1_5](https://doi.org/10.1007/978-3-540-24849-1_5).
- [17] J. A. Bergstra and J. W. Klop. Conditional rewrite rules: Confluence and termination. *Journal of Computer and System Sciences*, 32(3):323–362, 1986. doi:[10.1016/0022-0000\(86\)90033-4](https://doi.org/10.1016/0022-0000(86)90033-4).
- [18] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development; Coq’Art: The Calculus of Inductive Construccdtions*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:[10.1007/978-3-662-07964-5](https://doi.org/10.1007/978-3-662-07964-5).
- [19] C. Blanchette, C. Kaliszyk, L. C. Paulson, and J. Urban. Hammering towards QED. *Journal of Formalized Reasoning*, 9(1):101–148, 2016. doi:[10.6092/issn.1972-5787/4593](https://doi.org/10.6092/issn.1972-5787/4593).
- [20] 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](https://doi.org/10.1017/S0960129511000120).

- [21] M. Blum and S. Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1), 1995. doi:[10.1145/200836.200880](https://doi.org/10.1145/200836.200880).
- [22] S. Böhme and T. Nipkow. Sledgehammer: Judgement day. In *Proceedings of the 5th International Joint Conference on Automated Reasoning (IJCAR)*, volume 6173 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2010. doi:[10.1007/978-3-642-14203-1_9](https://doi.org/10.1007/978-3-642-14203-1_9).
- [23] B. Boyer, T. Genet, and T. Jensen. Certifying a tree automata completion checker. In *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR)*, volume 5195 of *Lecture Notes in Computer Science*, pages 523–538. Springer, 2008. doi:[10.1007/978-3-540-71070-7_43](https://doi.org/10.1007/978-3-540-71070-7_43).
- [24] S. Burckel. Syntactical methods for braids of three strands. *Journal of Symbolic Computation*, 31:557–564, 2001. doi:[10.1006/jsco.2000.0473](https://doi.org/10.1006/jsco.2000.0473).
- [25] M. Codish, C. Fuhs, J. Giesl, and P. Schneider-Kamp. Lazy abstraction for size-change termination. In *Proceedings of the 16th International Conference on Logic for Programming and Automated Reasoning (LPAR)*, volume 6397 of *Lecture Notes in Computer Science*, pages 217–232. Springer, 2010. doi:[10.1007/978-3-642-16242-8_16](https://doi.org/10.1007/978-3-642-16242-8_16).
- [26] É. Contejean, P. Courtieu, J. Forest, O. Pons, and X. Urbain. Automated certified proofs with CiME3. In *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications (RTA)*, volume 10 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21–30. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:[10.4230/LIPIcs.RTA.2011.21](https://doi.org/10.4230/LIPIcs.RTA.2011.21).
- [27] S. de Gouw, F. de Boer, and J. Rot. Proof Pearl: The KeY to correct and stable sorting. *Journal of Automated Reasoning*, 53(2):129–139, 2014. doi:[10.1007/s10817-013-9300-y](https://doi.org/10.1007/s10817-013-9300-y).
- [28] S. de Gouw, J. Rot, F. S. de Boer, R. Bubel, and R. Hähnle. OpenJDK’s Java.utils.Collection.sort() is broken: The good, the bad and the worst case. In *Proceedings of the 27th International Conference on Computer Aided Verification (CAV)*, volume 9206 of *Lecture Notes in Computer Science*, pages 273–289. Springer, 2015. doi:[10.1007/978-3-319-21690-4_16](https://doi.org/10.1007/978-3-319-21690-4_16).
- [29] S. de Gouw, F. S. de Boer, R. Bubel, R. Hähnle, J. Rot, and D. Steinhöfel. Verifying OpenJDK’s sort method for generic collections. *Journal of Automated Reasoning*, 62(1):93–126, 2019. doi:[10.1007/s10817-017-9426-4](https://doi.org/10.1007/s10817-017-9426-4).
- [30] R. C. de Vrijer. Conditional linearization. *Indagationes Mathematicae*, 10(1):145 – 159, 1999. doi:[10.1016/S0019-3577\(99\)80012-3](https://doi.org/10.1016/S0019-3577(99)80012-3).
- [31] N. Dershowitz. A note on simplification orderings. *Information Processing Letters*, 9(5):212–215, 1979. doi:[10.1016/0020-0190\(79\)90071-1](https://doi.org/10.1016/0020-0190(79)90071-1).

- [32] N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17(3):279–301, 1982. doi:10.1016/0304-3975(82)90026-3.
- [33] N. Dershowitz, M. Okada, and G. Sivakumar. Confluence of conditional rewrite systems. In *Proceedings of the 1st International Workshop on Conditional and Typed Rewriting Systems*, volume 308 of *Lecture Notes in Computer Science*, pages 31–44. Springer, 1988. doi:10.1007/3-540-19242-5_3.
- [34] H. Devie. Linear completion. In *Proceedings of the 2nd International Workshop on Conditional and Typed Rewriting Systems*, pages 233–245. Springer, 1991. doi:10.1007/3-540-54317-1_94.
- [35] L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics*, 35(4):413–422, 1913. doi:10.2307/2370405.
- [36] B. Felgenhauer and R. Thiemann. Reachability analysis with state-compatible automata. In *International Confluence on Language and Automated Theory and Applications (LATA)*, volume 8370 of *Lecture Notes in Computer Science*, pages 347–359. Springer, 2014. doi:10.1007/978-3-319-04921-2_28.
- [37] G. Feuillade and T. Genet. Reachability in conditional term rewriting systems. In *Proceedings of the 4th International Workshop on First-Order Theorem Proving*, volume 86 of *Electronic Notes in Theoretical Computer Science*, pages 133–146, 2003. doi:10.1016/S1571-0661(04)80658-3.
- [38] J.-C. Filliâtre and N. Magaud. Certification of sorting algorithms in the Coq system. In *Theorem Proving in Higher Order Logics: Emerging Trends*, 1999. <http://www-sop.inria.fr/croap/TPHOLs99/proceeding.html>.
- [39] D. Fridlender. Higman’s lemma in type theory. In *Proceedings of the 1st International Workshop on Types for Proofs and Programs (TYPES)*, volume 1512 of *Lecture Notes in Computer Science*, pages 112–133. Springer, 1998. doi:10.1007/BFb0097789.
- [40] J. H. Gallier, P. Narendran, D. A. Plaisted, S. Raatz, and W. Snyder. An algorithm for finding canonical sets of ground rewrite rules in polynomial time. *Journal of the ACM*, 40(1):1–16, 1993. doi:10.1145/138027.138032.
- [41] T. Genet. Decidable approximations of sets of descendants and sets of normal forms. In *Proceedings of the 9th International Conference on Rewriting Techniques and Applications (RTA)*, volume 1379 of *Lecture Notes in Computer Science*, pages 151–165. Springer, 1998. doi:10.1007/BFb0052368.
- [42] T. Genet and V. Viet Triem Tong. Reachability analysis of term rewriting systems with timbuk. In *Proceedings of the 8th International Conference on Logic for Programming and Automated Reasoning (LPAR)*, volume 2250 of *Lecture Notes in*

- Computer Science*, pages 695–706. Springer, 2001. doi:10.1007/3-540-45653-8_48.
- [43] J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and disproving termination of higher-order functions. In *Proceedings of the 5th International Workshop on Frontiers of Combining Systems*, volume 3717 of *Lecture Notes in Computer Science*, pages 216–231. Springer, 2005. doi:10.1007/11559306_12.
 - [44] J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 281–286. Springer, 2006. doi:10.1007/11814771_24.
 - [45] J. Giesl, A. Rubio, C. Sternagel, J. Waldmann, and A. Yamada. The termination and complexity competition. In *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 11429 of *Lecture Notes in Computer Science*, pages 156–166. Springer, 2019. doi:10.1007/978-3-030-17502-3_10.
 - [46] K. Gmeiner. CoScart: Confluence prover in Scala. In *Proceedings of the 5th International Workshop on Confluence (IWC)*, page 82, 2016.
 - [47] K. Gmeiner, B. Gramlich, and F. Schernhammer. On (un)soundness of unravelings. In *Proceedings of the 21st International Conference on Rewriting Techniques and Applications (RTA)*, volume 6 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 119–134. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPIcs.RTA.2010.119.
 - [48] K. Gmeiner, N. Nishida, and B. Gramlich. Proving confluence of conditional term rewriting systems via unravelings. In *Proceedings of the 2nd International Workshop on Confluence (IWC)*, pages 35–39, 2013.
 - [49] M. Gordon. From LCF to HOL: a short history. In *Proof, Language, and Interaction, Essays In Honour of Robin Milner*, pages 169–186. MIT Press, 2000.
 - [50] M. J. C. Gordon, R. Milner, and C. P. Wadsworth. *Edinburgh LCF*, volume 78 of *Lecture Notes in Computer Science*. Springer, 1979. doi:10.1007/3-540-09724-4.
 - [51] B. Gramlich. On interreduction of semi-complete term rewriting systems. *Theoretical Computer Science*, 258(1-2):435–451, 2001. doi:10.1016/S0304-3975(00)00030-X.
 - [52] F. Haftmann and T. Nipkow. Code generation via higher-order rewrite systems. In *Proceedings of the 10th International Symposium on Functional and Logic Programming (FLOPS)*, volume 6009 of *Lecture Notes in Computer Science*, pages 103–117. Springer, 2010. doi:10.1007/978-3-642-12251-4_9.
 - [53] F. Haftmann, G. Klein, T. Nipkow, and N. Schirmer. L^AT_EX sugar for Isabelle documents, 2013. <http://isabelle.in.tum.de/website-Isabelle2013-2/dist/Isabelle2013-2/doc/sugar.pdf>.

Bibliography

- [54] M. Hamana and K. Kikuchi. The system `sol` version 2018. In *Proceedings of the 7th International Workshop on Confluence (IWC)*, page 70, 2018.
- [55] M. Hanus. On extra variables in (equational) logic programming. In *Proceedings of the 12th International Conference on Logic Programming*, pages 665–679. MIT Press, 1995.
- [56] H. Herbelin. A program from an A-translated impredicative proof of Higman’s lemma, 1994. <http://coq.inria.fr/pylons/contrijs/view/HigmanNW/v8.3>.
- [57] G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, s3-2(1):326–336, 1952. doi:10.1112/plms/s3-2.1.326.
- [58] N. Hirokawa, A. Middeldorp, and C. Sternagel. A new and formalized proof of abstract completion. In *Proceedings of the 5th International Conference on Interactive Theorem Proving (ITP)*, volume 8558 of *Lecture Notes in Computer Science*, pages 292–307, 2014. doi:10.1007/978-3-319-08970-6_19.
- [59] N. Hirokawa, A. Middeldorp, C. Sternagel, and S. Winkler. Infinite runs in abstract completion. In *Proceedings of the 2nd International Conference on Formal Structures for Computation and Deduction (FSCD)*, volume 84 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:16, 2017. doi:10.4230/LIPIcs.FSCD.2017.19.
- [60] P. Hudak, J. Peterson, and J. H. Fasel. A gentle introduction to Haskell. *SIGPLAN Notices*, 27(5), 1992. doi:doi.acm.org/10.1145/130697.130698. Original version at <http://doi.acm.org/10.1145/130697.130698>, updated version at <https://www.haskell.org/tutorial/>.
- [61] G. P. 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.
- [62] G. P. Huet. A complete proof of correctness of the Knuth-Bendix completion algorithm. *Journal of Computer and System Sciences*, 23(1):11–21, 1981. doi:10.1016/0022-0000(81)90002-7.
- [63] T. Ida and S. Okui. Outside-in conditional narrowing. *IEICE Transactions on Information and Systems*, E77-D(6):631–641, 1994.
- [64] F. Jacquemard. Decidable approximations of term rewriting systems. In *Proceedings of the 7th International Conference on Rewriting Techniques and Applications (RTA)*, volume 1103 of *Lecture Notes in Computer Science*, pages 362–376. Springer, 1996. doi:10.1007/3-540-61464-8_65.
- [65] S. Kamin and J.-J. Lévy. Two generalizations of the recursive path ordering. Unpublished manuscript, University of Illinois, 1980.

- [66] D. Kapur and P. Narendran. A finite Thue system with decidable word problem and without equivalent finite canonical system. *Theoretical Computer Science*, 35: 337–344, 1985. doi:[10.1016/0304-3975\(85\)90023-4](https://doi.org/10.1016/0304-3975(85)90023-4).
- [67] D. Kapur, D. R. Musser, and P. Narendran. Only prime superpositions need be considered in the Knuth-Bendix completion procedure. *Journal of Symbolic Computation*, 6(1):19–36, 1988. doi:[10.1016/S0747-7171\(88\)80019-1](https://doi.org/10.1016/S0747-7171(88)80019-1).
- [68] D. Kapur, P. Narendran, and F. Otto. On ground-confluence of term rewriting systems. *International Conference on*, 86(1):14–31, 1990. doi:[10.1016/0890-5401\(90\)90023-B](https://doi.org/10.1016/0890-5401(90)90023-B).
- [69] D. Klein and N. Hirokawa. Maximal completion. In *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications (RTA)*, volume 10 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 71–80. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:[10.4230/LIPIcs.RTA.2011.71](https://doi.org/10.4230/LIPIcs.RTA.2011.71).
- [70] J. W. Klop. *Combinatory Reduction Systems*. PhD thesis, Utrecht University, 1980.
- [71] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.
- [72] C. Kop. *Higher-Order Termination: Automatable Techniques for Proving Termination of Higher-Order Term Rewriting Systems*. PhD thesis, VU University Amsterdam, 2012. <http://hdl.handle.net/1871/39346>.
- [73] M. Korp, C. Sternagel, H. Zankl, and A. Middeldorp. Tyrolean Termination Tool 2. In *Proceedings of the 20th International Conference on Rewriting Techniques and Applications (RTA)*, volume 5595 of *Lecture Notes in Computer Science*, pages 295–304. Springer, 2009. doi:[10.1007/978-3-642-02348-4_21](https://doi.org/10.1007/978-3-642-02348-4_21).
- [74] A. Krauss. Partial and nested recursive function definitions in higher-order logic. *Journal of Automated Reasoning*, 44(4):303–336, 2010. doi:[10.1007/s10817-009-9157-2](https://doi.org/10.1007/s10817-009-9157-2).
- [75] A. Krauss. Recursive definitions of monadic functions. In *Proceedings of the Workshop on Partiality and Recursion in Interactive Theorem Proving (PAR)*, volume 43 of *Electronic Proceedings in Theoretical Computer Science*, pages 1–13, 2010. doi:[10.4204/EPTCS.43.1](https://doi.org/10.4204/EPTCS.43.1).
- [76] A. Krauss, C. Sternagel, R. Thiemann, C. Fuhs, and J. Giesl. Termination of Isabelle functions via termination of rewriting. In *Proceedings of the 2nd International Conference on Interactive Theorem Proving (ITP)*, volume 6898 of *Lecture Notes in Computer Science*, pages 152–167. Springer, 2011. doi:[10.1007/978-3-642-22863-6_13](https://doi.org/10.1007/978-3-642-22863-6_13).

- [77] J. B. Kruskal. Well-quasi-ordering, the tree theorem, and Vazsonyi’s conjecture. *Transactions of the American Mathematical Society*, 95(2):210–225, 1960. doi:10.2307/1993287.
- [78] P. Lammich. Efficient verified (UN)SAT certificate checking. *Journal of Automated Reasoning*, pages 1–20, 2019. doi:10.1007/s10817-019-09525-z.
- [79] K. R. M. Leino and P. Lucio. An assertional proof of the stability and correctness of natural mergesort. *ACM Transactions on Computational Logic*, 17(1):6:1–6:22, 2015. doi:10.1145/2814571.
- [80] S. Lucas and R. Gutiérrez. Use of logical models for proving infeasibility in term rewriting. *Information Processing Letters*, 136:90–95, 2018. doi:10.1016/j.ipl.2018.04.002.
- [81] C. Marché. Normalized rewriting: An alternative to rewriting modulo a set of equations. *Journal of Symbolic Computation*, 21(3):253–288, 1996. doi:10.1006/jsco.1996.0011.
- [82] S. Marlow. Haskell 2010 language report, 2019. <https://www.haskell.org/definition/haskell2010.pdf>.
- [83] U. Martin and T. Nipkow. Ordered Rewriting and Confluence. In *Proceedings of the 10th International Conference on Automated Deduction (CADE)*, volume 449 of *Lecture Notes in Computer Science*, pages 366–380, 1990. doi:10.1007/3-540-52885-7_100.
- [84] F. J. Martín-Mateos, J. L. Ruiz-Reina, J. A. Alonso, and M. J. Hidalgo. Proof Pearl: A formal proof of Higman’s lemma in ACL2. *Journal of Automated Reasoning*, 47(3):229–250, 2011. doi:10.1007/s10817-010-9178-x.
- [85] W. McCune. Solution of the Robbins problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997. doi:10.1023/A:1005843212881.
- [86] W. McCune, R. Veroff, B. Fitelson, K. Harris, A. Feist, and L. Vos. Short single axioms for Boolean algebra. *Journal of Automated Reasoning*, 29(1):1–16, 2002. doi:10.1023/A:1020542009983.
- [87] Y. Métivier. About the rewriting systems produced by the Knuth-Bendix completion algorithm. *Information Processing Letters*, 16(1):31–34, 1983. doi:10.1016/0020-0190(83)90009-1.
- [88] F. Meßner, J. Parsert, J. Schöpf, and C. Sternagel. A formally verified solver for homogeneous linear diophantine equations. In *Proceedings of the 9th International Conference on Interactive Theorem Proving (ITP)*, volume 10895 of *Lecture Notes in Computer Science*, pages 441–458. Springer, 2018. doi:10.1007/978-3-319-94821-8_26.

- [89] A. Middeldorp. Approximating dependency graphs using tree automata techniques. In *Proceedings of the 1st International Joint Conference on Automated Reasoning (IJCAR)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 593–610, 2001. doi:10.1007/3-540-45744-5_49.
- [90] A. Middeldorp and E. Hamoen. Completeness results for basic narrowing. *Applicable Algebra in Engineering, Communication and Computing*, 5:213–253, 1994. doi:10.1007/BF01190830.
- [91] A. Middeldorp and H. Zantema. Simple termination of rewrite systems. *Theoretical Computer Science*, 175(1):127–158, 1997. doi:10.1016/S0304-3975(96)00172-7.
- [92] A. Middeldorp, J. Nagele, and K. Shintani. Confluence competition 2019. In *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 11429 of *Lecture Notes in Computer Science*, pages 25–40. Springer, 2019. doi:10.1007/978-3-030-17502-3_2.
- [93] C. R. Murthy. *Extracting Constructive Content from Classical Proofs*. PhD thesis, Cornell University, 1990. <http://hdl.handle.net/1813/6991>.
- [94] J. Nagele. CoCo 2018 participant: CSI \hat{h} o 0.3.2. In *Proceedings of the 7th International Workshop on Confluence (IWC)*, page 68, 2018.
- [95] J. Nagele and A. Middeldorp. Certification of classical confluence results for left-linear term rewrite systems. In *Proceedings of the 7th International Conference on Interactive Theorem Proving (ITP)*, volume 9807 of *Lecture Notes in Computer Science*, pages 290–306. Springer, 2016. doi:10.1007/978-3-319-43144-4_18.
- [96] J. Nagele and R. Thiemann. Certification of confluence proofs using CeTA. In *Proceedings of the 3rd International Workshop on Confluence (IWC)*, 2014. Available at <http://cl-informatik.uibk.ac.at/users/thiemann/paper/IWC14CeTA.pdf>.
- [97] J. Nagele and H. Zankl. Certified rule labeling. In *Proceedings of the 26th International Conference on Rewriting Techniques and Applications (RTA)*, volume 36 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 269–284. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.RTA.2015.269.
- [98] J. Nagele, R. Thiemann, and S. Winkler. Certification of nontermination proofs using strategies and nonlooping derivations. In *Proceedings of the 6th Working Conference on Verified Software: Theories, Tools, and Experiments (VSTTE)*, volume 8471 of *Lecture Notes in Computer Science*, pages 216–232. Springer, 2014. doi:10.1007/978-3-319-12154-3_14.
- [99] J. Nagele, B. Felgenhauer, and H. Zankl. Certifying confluence proofs via relative termination and rule labeling. *Logical Methods in Computer Science*, 13(2):4:1–4:27, 2017. doi:10.23638/LMCS-13(2:4)2017.

Bibliography

- [100] C. S. J. A. Nash-Williams. On well-quasi-ordering finite trees. *Proceedings of the Cambridge Philosophical Society*, 59(4):833–835, 1963. doi:[10.1017/S0305004100003844](https://doi.org/10.1017/S0305004100003844).
- [101] M. H. A. Newman. On theories with a combinatorial definition of equivalence. *Annals of Mathematics*, 43(2):223–243, 1942. doi:[10.2307/1968867](https://doi.org/10.2307/1968867).
- [102] T. Nipkow. Equational reasoning in Isabelle. *Sci. Comput. Programming*, 12(2):123–149, 1989. doi:[10.1016/0167-6423\(89\)90038-5](https://doi.org/10.1016/0167-6423(89)90038-5).
- [103] T. Nipkow, L. C. 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](https://doi.org/10.1007/3-540-45949-9).
- [104] N. Nishida and Y. Maeda. CO3 (version 2.0). In *Proceedings of the 8th International Workshop on Confluence (IWC)*, page 50, 2019.
- [105] N. Nishida, M. Sakai, and T. Sakabe. Soundness of unravelings for conditional term rewriting systems via ultra-properties related to linearity. *Logical Methods in Computer Science*, 8:4:1–4:49, 2012. doi:[10.2168/LMCS-8\(3:4\)2012](https://doi.org/10.2168/LMCS-8(3:4)2012).
- [106] V. van Oostrom. Confluence by decreasing diagrams. *Theoretical Computer Science*, 126(2):259–280, 1994. doi:[10.1016/0304-3975\(92\)00023-K](https://doi.org/10.1016/0304-3975(92)00023-K).
- [107] V. van Oostrom. Random descent. In *Proceedings of the 18th International Conference on Rewriting Techniques and Applications (RTA)*, volume 4533 of *Lecture Notes in Computer Science*, pages 314–328, 2007. doi:[10.1007/978-3-540-73449-9_24](https://doi.org/10.1007/978-3-540-73449-9_24).
- [108] V. van Oostrom. Confluence by decreasing diagrams – converted. In *Proceedings of the 19th International Conference on Rewriting Techniques and Applications (RTA)*, volume 5117 of *Lecture Notes in Computer Science*, pages 306–320, 2008. doi:[10.1007/978-3-540-70590-1_21](https://doi.org/10.1007/978-3-540-70590-1_21).
- [109] V. van Oostrom and Y. Toyama. Normalisation by random descent. In *Proceedings of the 1st International Conference on Formal Structures for Computation and Deduction (FSCD)*, volume 52 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:18, 2016. doi:[10.4230/LIPIcs.FSCD.2016.32](https://doi.org/10.4230/LIPIcs.FSCD.2016.32).
- [110] M. Ogawa and C. Sternagel. Open induction. *The Archive of Formal Proofs*, Nov. 2012. ISSN 2150-914x. [afp:Open_Induction](https://arxiv.org/abs/1211.4036).
- [111] E. Ohlebusch. *Advanced Topics in Term Rewriting*. Springer, 2002. doi:[10.1007/978-1-4757-3661-8](https://doi.org/10.1007/978-1-4757-3661-8).
- [112] R. O’Keefe. A smooth applicative merge sort. Technical report, Department of Artificial Intelligence, University of Edinburgh, 1982.

- [113] K. Onozawa, K. Kikuchi, T. Aoto, and Y. Toyama. ACPH: System description for CoCo 2017. In *Proceedings of the 6th International Workshop on Confluence (IWC)*, page 70, 2017.
- [114] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In *Proceedings of the 11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, 1992. doi:10.1007/3-540-55602-8_217.
- [115] L. C. Paulson. *Logic and Computation: Interactive Proof with Cambridge LCF*. Cambridge University Press, 1987.
- [116] L. C. Paulson. *ML for the Working Programmer*. Cambridge University Press, New York, NY, USA, second edition, 1996. ISBN 0-521-56543-X.
- [117] 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.
- [118] D. Plaisted and A. Sattler-Klein. Proof lengths for equational completion. *International Conference on*, 125(2):154–170, 1996. doi:10.1006/inco.1996.0028.
- [119] E. L. Post. A variant of a recursively unsolvable problem. *Bull. Amer. Math. Soc.*, 52, 1946. doi:10.1090/S0002-9904-1946-08555-9.
- [120] N. Robertson and P. Seymour. Graph minors. i. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983. doi:10.1016/0095-8956(83)90079-5.
- [121] N. Robertson and P. Seymour. Graph minors. xx. wagner’s conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. doi:10.1016/j.jctb.2004.08.001.
- [122] H. Sato and S. Winkler. Encoding dependency pair techniques and control strategies for maximal completion. In *Proceedings of the 25th International Conference on Automated Deduction (CADE)*, volume 9195 of *Lecture Notes in Computer Science*, pages 152–162, 2015. doi:10.1007/978-3-319-21401-6_10.
- [123] F. Schernhammer and B. Gramlich. VMTL – a modular termination laboratory. In *Proceedings of the 20th International Conference on Rewriting Techniques and Applications (RTA)*, volume 5595 of *Lecture Notes in Computer Science*, pages 285–294. Springer, 2009. doi:10.1007/978-3-642-02348-4_20.
- [124] P. Schultz and R. Wisnesky. Algebraic data integration. *Journal of Functional Programming*, 27(e24):51 pages, 2017. doi:10.1017/S0956796817000168.
- [125] M. Seisenberger. *On the Constructive Content of Proofs*. PhD thesis, LMU Munich, 2003. <http://nbn-resolving.de/urn:nbn:de:bvb:19-16190>.

- [126] K. Shintani and N. Hirokawa. CoLL: A confluence tool for left-linear term rewrite systems. In *Proceedings of the 25th International Conference on Automated Deduction (CADE)*, volume 9195 of *Lecture Notes in Computer Science*, pages 127–136. Springer, 2015. doi:[10.1007/978-3-319-21401-6_8](https://doi.org/10.1007/978-3-319-21401-6_8).
- [127] K. Slind and M. Norrish. A brief overview of HOL4. In *Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, volume 5170 of *Lecture Notes in Computer Science*, pages 28–32. Springer, 2008. doi:[10.1007/978-3-540-71067-7_6](https://doi.org/10.1007/978-3-540-71067-7_6).
- [128] W. Snyder. A fast algorithm for generating reduced ground rewriting systems from a set of ground equations. *Journal of Symbolic Computation*, 15(4):415–450, 1993. doi:[10.1006/jSCO.1993.1029](https://doi.org/10.1006/jSCO.1993.1029).
- [129] C. Sternagel. *Automatic Certification of Termination Proofs*. PhD thesis, University of Innsbruck, 2010.
- [130] C. Sternagel. Efficient Mergesort. *The Archive of Formal Proofs*, Nov. 2011. ISSN 2150-914x. [afp:Efficient-Mergesort](https://arxiv.org/abs/1111.0455).
- [131] C. Sternagel. Well-Quasi-Orders. *The Archive of Formal Proofs*, Apr. 2012. ISSN 2150-914x. [afp:Well_Quasi_Orders](https://arxiv.org/abs/1208.1366).
- [132] C. Sternagel. A locale for minimal bad sequences. In *Isabelle Users Workshop*, 2012. [arXiv:1208.1366](https://arxiv.org/abs/1208.1366).
- [133] C. Sternagel. Certified Kruskal’s tree theorem. In *Proceedings of the 3rd International Conference on Certified Programs and Proofs (CPP)*, volume 8307 of *Lecture Notes in Computer Science*, pages 178–193. Springer, 2013. doi:[10.1007/978-3-319-03545-1_12](https://doi.org/10.1007/978-3-319-03545-1_12).
- [134] C. Sternagel and T. Sternagel. Level-confluence of 3-CTRSs in Isabelle/HOL. In *Proceedings of the 4th International Workshop on Confluence (IWC)*, pages 28–32, 2015. [arXiv:1602.07115](https://arxiv.org/abs/1602.07115).
- [135] C. Sternagel and T. Sternagel. Certifying confluence of almost orthogonal CTRSs via exact tree automata completion. In *Proceedings of the 1st International Conference on Formal Structures for Computation and Deduction (FSCD)*, volume 52 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:[10.4230/LIPIcs.FSCD.2016.29](https://doi.org/10.4230/LIPIcs.FSCD.2016.29).
- [136] C. Sternagel and T. Sternagel. Certifying confluence of quasi-decreasing strongly deterministic conditional term rewrite systems. In *Proceedings of the 26th International Conference on Automated Deduction (CADE)*, volume 10395 of *Lecture Notes in Computer Science*, pages 413–431. Springer, 2017. doi:[10.1007/978-3-319-63046-5_26](https://doi.org/10.1007/978-3-319-63046-5_26).

- [137] C. Sternagel and R. Thiemann. Signature extensions preserve termination. In *Proceedings of the 19th EACSL Annual Conference on Computer Science Logic (CSL)*, volume 6247 of *Lecture Notes in Computer Science*, pages 514–528. Springer, 2010. doi:[10.1007/978-3-642-15205-4_39](https://doi.org/10.1007/978-3-642-15205-4_39).
- [138] C. Sternagel and R. Thiemann. Certified subterm criterion and certified usable rules. In *Proceedings of the 21st International Conference on Rewriting Techniques and Applications (RTA)*, volume 6 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 325–340. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010. doi:[10.4230/LIPIcs.RTA.2010.325](https://doi.org/10.4230/LIPIcs.RTA.2010.325).
- [139] C. Sternagel and R. Thiemann. Modular and certified semantic labeling and unlabeling. In *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications (RTA)*, volume 10 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 329–344. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:[10.4230/LIPIcs.RTA.2011.329](https://doi.org/10.4230/LIPIcs.RTA.2011.329).
- [140] C. Sternagel and R. Thiemann. Generalized and formalized uncurrying. In *Proceedings of the 8th International Workshop on Frontiers of Combining Systems*, volume 6988 of *Lecture Notes in Computer Science*, pages 243–258. Springer, 2011. doi:[10.1007/978-3-642-24364-6_17](https://doi.org/10.1007/978-3-642-24364-6_17).
- [141] C. Sternagel and R. Thiemann. Certification of nontermination proofs. In *Proceedings of the 3rd International Conference on Interactive Theorem Proving (ITP)*, volume 7406 of *Lecture Notes in Computer Science*, pages 266–282. Springer, 2012. doi:[10.1007/978-3-642-32347-8_18](https://doi.org/10.1007/978-3-642-32347-8_18).
- [142] C. Sternagel and R. Thiemann. Formalizing Knuth-Bendix orders and Knuth-Bendix completion. In *Proceedings of the 24th International Conference on Rewriting Techniques and Applications (RTA)*, volume 21 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 287–302. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:[10.4230/LIPIcs.RTA.2013287](https://doi.org/10.4230/LIPIcs.RTA.2013287).
- [143] C. Sternagel and R. Thiemann. Certification monads. *The Archive of Formal Proofs*, Oct. 2014. ISSN 2150-914x. [afp:Certification_Monads](https://arxiv.org/abs/1410.0001).
- [144] C. Sternagel and R. Thiemann. Formalizing monotone algebras for certification of termination and complexity proofs. In *Proceedings of the Joint International Conference on Rewriting Techniques and Applications (RTA) and International Conference on Typed Lambda Calculi and Applications (TLCA) 2014*, volume 8560 of *Lecture Notes in Computer Science*, pages 441–455. Springer, 2014.
- [145] C. Sternagel and R. Thiemann. Haskell’s show-class in Isabelle/HOL. *The Archive of Formal Proofs*, July 2014. ISSN 2150-914x. [afp:Show](https://arxiv.org/abs/1407.0001).
- [146] C. Sternagel and R. Thiemann. Formalizing monotone algebras for certification of termination and complexity proofs. In *Proceedings of the Joint International*

- Conference on Rewriting Techniques and Applications (RTA) and International Conference on Typed Lambda Calculi and Applications (TLCA) 2014*, volume 8560 of *Lecture Notes in Computer Science*, pages 441–455. Springer, 2014. doi:[10.1007/978-3-319-08918-8_30](https://doi.org/10.1007/978-3-319-08918-8_30).
- [147] C. Sternagel and R. Thiemann. The certification problem format. In *Proceedings of the 11th International Workshop on User Interfaces for Theorem Provers (UITP)*, volume 167 of *Electronic Proceedings in Theoretical Computer Science*, pages 61–72, 2014. doi:[10.1007/11780274_28](https://doi.org/10.1007/11780274_28).
 - [148] C. Sternagel and R. Thiemann. Xml. *The Archive of Formal Proofs*, Oct. 2014. ISSN 2150-914x. [afp:XML](https://arxiv.org/abs/1410.0001).
 - [149] C. Sternagel and R. Thiemann. A framework for developing stand-alone certifiers. *Electronic Notes in Theoretical Computer Science*, 312:51–67, 2015. doi:[10.1016/j.entcs.2015.04.004](https://doi.org/10.1016/j.entcs.2015.04.004).
 - [150] C. Sternagel and R. Thiemann. Deriving comparators and show-functions in Isabelle/HOL. In *Proceedings of the 6th International Conference on Interactive Theorem Proving (ITP)*, volume 9236 of *Lecture Notes in Computer Science*, pages 421–437. Springer, 2015. doi:[10.1007/978-3-319-22102-1_28](https://doi.org/10.1007/978-3-319-22102-1_28).
 - [151] C. Sternagel and S. Winkler. Certified ordered completion. In *Proceedings of the 7th International Workshop on Confluence (IWC)*, 2018. [arXiv:1805.10090](https://arxiv.org/abs/1805.10090).
 - [152] C. Sternagel and S. Winkler. Certified equational reasoning via ordered completion. In *Proceedings of the 27th International Conference on Automated Deduction (CADE)*, *Lecture Notes in Computer Science*, pages 508–525. Springer, 2019. doi:[10.1007/978-3-030-29436-6_30](https://doi.org/10.1007/978-3-030-29436-6_30).
 - [153] C. Sternagel and A. Yamada. Reachability analysis for termination and confluence of rewriting. In *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 11427 of *Lecture Notes in Computer Science*, pages 262–278. Springer, 2019. doi:[10.1007/978-3-030-17462-0_15](https://doi.org/10.1007/978-3-030-17462-0_15).
 - [154] C. Sternagel, R. Thiemann, S. Winkler, and H. Zankl. CeTA – a tool for certified termination analysis. In *Proceedings of the 10th International Workshop on Termination (WST)*, 2009. [arXiv:1208.1591](https://arxiv.org/abs/1208.1591).
 - [155] T. Sternagel and A. Middeldorp. Conditional confluence (system description). In *Proceedings of the Joint International Conference on Rewriting Techniques and Applications (RTA) and International Conference on Typed Lambda Calculi and Applications (TLCA) 2014*, volume 8560 of *Lecture Notes in Computer Science*, pages 456–465, 2014. doi:[10.1007/978-3-319-08918-8_31](https://doi.org/10.1007/978-3-319-08918-8_31).
 - [156] T. Sternagel and A. Middeldorp. Infeasible conditional critical pairs. In *Proceedings of the 4th International Workshop on Confluence (IWC)*, pages 13–17, 2015.

- [157] T. Sternagel and C. Sternagel. Formalized confluence of quasi-decreasing, strongly deterministic conditional TRSs. In *Proceedings of the 5th International Workshop on Confluence (IWC)*, pages 60–64, 2016. [arXiv:1609.03341](#).
- [158] T. Sternagel and C. Sternagel. Certified non-confluence with ConCon 1.5. In *Proceedings of the 6th International Workshop on Confluence (IWC)*, 2017. [arXiv:1709.05162](#).
- [159] T. Sternagel and H. Zankl. KBCV – Knuth-Bendix completion visualizer. In *Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR)*, volume 7364 of *Lecture Notes in Computer Science*, pages 530–536. Springer, 2012. [doi:10.1007/978-3-642-31365-3_41](#).
- [160] T. Sternagel, S. Winkler, and H. Zankl. Recording completion for certificates in equational reasoning. In *Proceedings of the 4th International Conference on Certified Programs and Proofs (CPP)*, pages 41–47, 2015. [doi:10.1145/2676724.2693171](#).
- [161] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts. *Journal of Automated Reasoning*, 43(4):337–362, 2009. [doi:10.1007/s10817-009-9143-8](#).
- [162] T. Suzuki, A. Middeldorp, and T. Ida. Level-confluence of conditional rewrite systems with extra variables in right-hand sides. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications (RTA)*, volume 914 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 1995. [doi:10.1007/3-540-59200-8_56](#).
- [163] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003. ISBN 9780521391153.
- [164] R. Thiemann. Formalizing bounded increase. In *Proceedings of the 4th International Conference on Interactive Theorem Proving (ITP)*, volume 7995 of *Lecture Notes in Computer Science*, pages 245–260. Springer, 2013. [doi:10.1007/978-3-642-39634-2_19](#).
- [165] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468. Springer, 2009. [doi:10.1007/978-3-642-03359-9_31](#).
- [166] R. Thiemann, G. Allais, and J. Nagele. On the formalization of termination techniques based on multiset orderings. In *Proceedings of the 23rd International Conference on Rewriting Techniques and Applications (RTA)*, volume 15 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 339–354, 2012. [doi:10.4230/LIPIcs.RTA.2012.339](#).
- [167] Y. Toyama. How to prove equivalence of term rewriting systems without induction. *Theoretical Computer Science*, 90(2):369–390, 1991.

- [168] W. Veldman. An intuitionistic proof of Kruskal’s theorem. *Archive for Mathematical Logic*, 43(2):215–264, 2004. doi:10.1007/s00153-003-0207-x.
- [169] J. Waldmann. Matchbox: A tool for match-bounded string rewriting. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA)*, volume 3091 of *Lecture Notes in Computer Science*, pages 85–94. Springer, 2004. doi:10.1007/978-3-540-25979-4_6.
- [170] I. Wehrman, A. Stump, and E. Westbrook. Slothrop: Knuth-Bendix completion with a modern termination checker. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA)*, volume 4098 of *Lecture Notes in Computer Science*, pages 287–296. Springer, 2006. doi:10.1007/11805618_22.
- [171] M. Wenzel. *Isabelle/Isar – A Versatile Environment for Human-readable Formal Proof Documents*. PhD thesis, Technische Universität München, 2002. <http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2002/wenzel.pdf>.
- [172] M. Wenzel, L. C. Paulson, and T. Nipkow. The Isabelle framework. In *Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, volume 5170 of *Lecture Notes in Computer Science*, pages 33–38. Springer, 2008. doi:10.1007/978-3-540-71067-7_7.
- [173] F. Winkler and B. Buchberger. A criterion for eliminating unnecessary reductions in the Knuth-Bendix algorithm. In *Proc. Colloquium on Algebra, Combinatorics and Logic in Computer Science, Vol. II*, volume 42 of *Colloquia Mathematica Societatis J. Bolyai*, pages 849–869, 1986.
- [174] S. Winkler. A ground joinability criterion for ordered completion. In *Proceedings of the 6th International Workshop on Confluence (IWC)*, pages 45–49, 2017.
- [175] S. Winkler and A. Middeldorp. Normalized completion revisited. In *Proceedings of the 24th International Conference on Rewriting Techniques and Applications (RTA)*, volume 21 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 318–333. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPIcs.RTA.2013.318.
- [176] S. Winkler and G. Moser. MædMax: A maximal ordered completion tool. In *Proceedings of the 9th International Joint Conference on Automated Reasoning (IJCAR)*, volume 10900 of *Lecture Notes in Computer Science*, pages 472–480, 2018. doi:10.1007/978-3-319-94205-6_31.
- [177] S. Winkler and R. Thiemann. Formalizing soundness and completeness of unravelings. In *Proceedings of the 10th International Workshop on Frontiers of Combining Systems*, volume 9322 of *Lecture Notes in Computer Science*, pages 239–255. Springer, 2015. doi:10.1007/978-3-319-24246-0_15.

- [178] S. Winkler, H. Sato, A. Middeldorp, and M. Kurihara. Optimizing mkbTT. In *Proceedings of the 21st International Conference on Rewriting Techniques and Applications (RTA)*, volume 6 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 373–384. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPIcs.RTA.2010.373.
- [179] C. Wu, X. Zhang, and C. Urban. The Myhill-Nerode theorem based on regular expressions. *The Archive of Formal Proofs*, Aug. 2011. ISSN 2150-914x. [afp:Myhill-Nerode](#).
- [180] C. Wu, X. Zhang, and C. Urban. A formalisation of the Myhill-Nerode theorem based on regular expressions. *Journal of Automated Reasoning*, 52(2):1–30, 2014. doi:10.1007/s10817-013-9297-2.
- [181] A. Yamada, K. Kusakari, and T. Sakabe. Nagoya termination tool. In *Proceedings of the Joint International Conference on Rewriting Techniques and Applications (RTA) and International Conference on Typed Lambda Calculi and Applications (TLCA) 2014*, volume 8560 of *Lecture Notes in Computer Science*, pages 466–475. Springer, 2014. doi:10.1007/978-3-319-08918-8_32.
- [182] A. Yamada, C. Sternagel, R. Thiemann, and K. Kusakari. AC dependency pairs revisited. In *Proceedings of the 25th EACSL Annual Conference on Computer Science Logic (CSL)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CSL.2016.8.
- [183] H. Zankl, C. Sternagel, D. Hofbauer, and A. Middeldorp. Finding and certifying loops. In *Proceedings of the 36th International Conference on Theory and Practice of Computer Science (SOFSEM)*, volume 5901 of *Lecture Notes in Computer Science*, pages 755–766. Springer, 2010. doi:10.1007/978-3-642-11266-9_63.
- [184] H. Zankl, B. Felgenhauer, and A. Middeldorp. CSI – a confluence tool. In *Proceedings of the 23rd International Conference on Automated Deduction (CADE)*, volume 6803 of *Lecture Notes in Artificial Intelligence*, pages 499–505. Springer, 2011. doi:10.1007/978-3-642-22438-6_38.
- [185] Y. Zhang, Y. Zhao, and D. Sanan. A verified timsort C implementation in Isabelle/HOL, 2018. [arXiv:1812.03318](#).