Department of Computer Science
Technical Report

The DP Framework for Proving Termination of Term Rewriting

René Thiemann

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2007-17

RWTH Aachen · Department of Computer Science · October 2007

The publications of the Department of Computer Science of RWTH Aachen (*Aachen University of Technology*) are in general accessible through the World Wide Web.

`http://aib.informatik.rwth-aachen.de/`

The DP Framework for Proving Termination of Term Rewriting

Von der Fakultät für Mathematik, Informatik und
Naturwissenschaften der Rheinisch-Westfälischen
Technischen Hochschule Aachen zur Erlangung des
akademischen Grades eines Doktors der
Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Informatiker
René Thiemann
aus
Stadtlohn

Berichter: Prof. Dr. Jürgen Giesl
Prof. Dr. Aart Middeldorp

Tag der mündlichen Prüfung: 24.10.2007

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online verfügbar

Abstract

Termination is the fundamental property of a program that for each input, the evaluation will eventually stop and return some output. Although the question whether a given program terminates is undecidable, many techniques have been developed which can be used to answer the question of termination for many programs automatically. Especially, termination of *term rewriting* is an interesting and widely studied area: Since the basic evaluation mechanism of many programming languages is term rewriting, one can successfully apply the termination techniques for term rewriting to analyze termination of programs automatically. Nevertheless, there still remain many programs that cannot be handled by any current technique that is amenable to automation.

In this thesis, we extend existing techniques and develop new methods for mechanized termination analysis of term rewrite systems. Currently, one of the most powerful techniques is the dependency pair approach. Up to now, it was regarded as one of several possible methods to prove termination. We show that dependency pairs can instead be used as a general concept to integrate arbitrary techniques for termination analysis. In this way, the benefits of different techniques can be combined and their modularity and power are increased significantly. We refer to this new concept as the "dependency pair *framework*" to distinguish it from the old "dependency pair *approach*".

Moreover, this framework facilitates the development of new methods for termination analysis. To demonstrate this, we design several novel techniques within the dependency pair framework. They can successfully be applied to prove termination of previously challenging programs. For example, our work describes new ways how to handle programs using accumulators, programs written in higher-order languages, and programs which only terminate w.r.t. a given evaluation strategy. We additionally show how to disprove termination, even under strategies.

All presented techniques are formulated in a uniform setting and are implemented in our *fully automated* termination prover AProVE. The significance of our results is demonstrated at the annual international Termination Competition, where the leading automated tools try to analyze termination of programs from different areas of computer science: Without the contributions of this thesis, AProVE would not have reached the highest scores both for proving and disproving termination in the years 2004 – 2007.

Acknowledgments

Zuallererst möchte ich mich bei Prof. Dr. Jürgen Giesl bedanken. Seit meiner Zeit am Lehr- und Forschungsgebiet hat er mich in jeder erdenklichen Weise unterstützt, sei es durch die vielen interessanten Anregungen und Diskussionen, durch die gemeinsame Arbeit bei der Entwicklung neuer Papiere, durch die Freiheit, meinen Arbeitsschwerpunkt sowohl thematisch als auch geographisch weitestgehend frei zu wählen, oder durch die Unterstützung zum Umgang mit den "Kleinen" Freuden des privaten Lebens.

I am grateful to Prof. Dr. Aart Middeldorp for being a member of my thesis committee, and for being a stimulating adversary in developing techniques for termination proving.

Ich danke vor allem auch meinem Kollegen Peter Schneider-Kamp. Zusammen mit ihm gab es immer wieder neue interessante Ideen zu diskutieren und zu entwickeln, und ich hoffe, nicht nur ich habe es genossen, selbst die abschreckendsten Beweise zur gegenseitigen Überprüfung auszutauschen. Zudem hätte ich ohne ihn wohl so manche halbe Stunde häufiger am Bahnhof gesessen.

Ich danke auch meinen Kollegen Stephan Swiderski und Carsten Fuhs für so manches interessante Gespräch und für das Korrekturlesen meiner Dissertation.

Thanks are given to everybody on the floor of I2 for a pleasant and friendly research environment.

Vielen Dank geht auch an die restlichen Mitglieder des AProVE-Teams. Ohne ihre Unterstützung hätten kaum so viele Techniken vom Papier auf den Rechner gebracht werden können. Zudem haben die vielen guten Ideen bei der Implementierung mit Sicherheit einen nicht unbeachtlichen Teil zum Erfolg bei den Wettbewerben beigetragen.

Und natürlich will ich mich auch bei Karin bedanken. Obwohl ich vor mehreren Deadlines kaum für sie ansprechbar war, hat sie mich immer unterstützt und mir Rückhalt gegeben.

René Thiemann

Contents

1. Introduction **1**

2. The Dependency Pair Framework **7**
- 2.1. $\mathcal{Q}$-Restricted Rewriting 7
- 2.2. Dependency Pairs 10

3. Processors Without Search **17**
- 3.1. Dependency Graph 18
- 3.2. Switching to Innermost Termination 24
- 3.3. Usable Rules 27
- 3.4. Star-Estimation of the Dependency Graph 31
- 3.5. Reducing $\mathcal{Q}$ 33

4. Processors Based on Orders **37**
- 4.1. Reduction Pairs 37
- 4.2. Needed Rules 42
- 4.3. Rule Removal 53
- 4.4. Usable Rules w.r.t. an Argument Filter 54
- 4.5. Needed Rules w.r.t. an Argument Filter 56
- 4.6. Subterm Criterion 60

5. Processors Based on Pair Transformations **67**
- 5.1. Instantiation 71
- 5.2. Forward Instantiation 72
- 5.3. Rewriting 73
- 5.4. Narrowing 79

6. Processors for Applicative Rewriting **85**
- 6.1. From Applicative to Functional Form 87
- 6.2. Needed Rules for Applicative DP Problems 93
- 6.3. Argument Filters for Applicative DP Problems 97

7. Processors Based on Semantic Labeling **105**
- 7.1. Semantic Labeling with Models 105
- 7.2. Semantic Labeling with Quasi-Models 113
- 7.3. Semantic Labeling and Unlabeling 120

8. Processors for Non-Termination Analysis **129**
- 8.1. Looping Problems 130
- 8.2. Switching to Termination 133
- 8.3. Detecting Looping Problems 135

9. Conclusion 143

Bibliography 149

A. Proofs 159
A.2. Proofs of Chapter 2 159
A.3. Proofs of Chapter 3 160
A.4. Proofs of Chapter 4 164
A.5. Proofs of Chapter 5 177
A.6. Proofs of Chapter 6 182
A.7. Proofs of Chapter 7 190
A.8. Proofs of Chapter 8 194

List of Processors 201

Index 203

1. Introduction

Termination is an essential property of programs and thus, techniques to verify and to analyze termination are of great interest in program verification and analysis. To apply termination techniques successfully even on complex programs, these techniques should be as powerful as possible. Additionally, it is desirable to have a push-button approach which does not need human guidance. Therefore, all these termination techniques should be fully mechanizable. However, the halting problem is undecidable, and the question whether a program terminates on all inputs is not even semi-decidable. Since we want to answer the latter question, our goal can only be to develop and extend termination techniques which can successfully analyze termination of many typical programs.

Whereas the research on automated termination analysis of imperative programming languages is a young discipline [BCDO06, BMS05, CPR06, CS02, PR04a, PR04b, Tiw04], more research has been done for functional programming languages [Abe04, BFG$^+$04, Gie95, GWB98, LJB01, PS97, TT00, Wal94, Xi02], and for logic programming [CLS05, DD94, DS02, LMS03, MR03, Sma04].

Since *term rewriting* is the fundamental principle underlying the evaluation in many programming languages, it is possible to reduce the question of termination of programs to termination of term rewrite systems (TRSs). For example, transformational approaches for logic programs have been developed in [AM93, AZ95, GW93, KKS98, Mar94, Mar96, Raa97, SGST06] and a transformational approach for the functional programming language Haskell has been recently developed in [GSST06]. In this way, termination analysis of TRSs becomes especially important since every enhancement in this area will have a positive impact on program verification of many programming languages.

Therefore, in this thesis we aim at improving the power of automated termination analysis of TRSs.

Termination of TRSs has been studied for decades. While early work focused on the development of suitable *reduction orders* [Der87, KB70, KL80, Lan79, Ste95], in the last years new techniques were developed which build upon these orders and which try to increase their power and applicability. One of the most powerful such techniques is the *dependency pair approach* (DP approach) [AG00, GA01, GAO02]. Apart from the DP approach, there also exist several other recent powerful techniques (e.g., *semantic labeling* [Zan95], the *monotonic semantic path order* [BFR00], *match-bounds* [GHW03, GHWZ07], etc.) for termination proofs of TRSs. Up to now, all these techniques were seen as separate approaches on their own.

However, in this thesis we first design a framework which allows the combination of termination techniques in a modular and uniform way. And second, existing techniques to prove termination of TRSs are improved and new techniques are developed, including methods to disprove termination. In these two ways, we improve the power of automated termination analysis significantly.

To be more precise, we make the following contributions. Here, the first three items (i) – (iii) are novel results which are not tailored to a specific termination technique, and each of the remaining items (iv) – (ix) are improvements of particular termination techniques, where sometimes a known method is extended or where we design completely new techniques.

(i) We show that dependency pairs are suitable as a general framework to integrate arbitrary methods for termination proofs. In this way, the benefits of all available termination techniques can be combined in a completely modular way and the classical DP *approach* is just a special case of our new DP *framework*. By combining termination techniques within the DP framework (instead of trying to apply them on a TRS directly, one after another), the flexibility, modularity, and power of these techniques are increased significantly.

(ii) To treat the termination problem for several different evaluation strategies in a uniform way, we introduce the notion of *$\mathcal{Q}$-restricted rewriting*, which generalizes both rewriting and innermost rewriting. We extend every termination technique investigated in this thesis to $\mathcal{Q}$-restricted rewriting and present all of them in a *uniform setting*. This even includes some techniques like semantic labeling which have not been applicable for strategies like innermost rewriting before.

   Moreover, due to the increased expressiveness of $\mathcal{Q}$-restricted rewriting we achieve many new *completeness results* which cannot be obtained for innermost rewriting directly. These results are especially useful for disproving termination.

(iii) Many termination techniques make use of common notions which are defined purely *syntactically*, e.g., there is a notion of usable rules [AG00] applied in several termination methods. However, there have been various incomparable syntactic refinements to obtain less usable rules [GTS05b, GTSF06, TGS04], and similar refinements also exist for other notions. The problem here is that with every improvement of such a syntactic notion, one has to check for every termination technique which uses that notion, whether one can replace the old syntactic version by the improved one. This boils down to adapting all proofs of all these termination techniques, clearly being a large effort.

   To solve this problem, in this thesis we introduce new *semantic* notions, e.g., for the usable rules, such that all previous syntactic notions are just estimations of the new semantic notion, and we integrate the semantic notions into every termination technique such that one can use them with every estimation. Hence, due to our results one can integrate every future improvement of an estimation into every termination technique without adapting a single proof.

   Moreover, for the automation we design *improved estimations* which are syntactically defined and which encompass all previous syntactic versions. These improvements considerably increase the power of around half of the termination techniques examined in the thesis.

(iv) There exist many termination techniques which are more powerful if they are used to prove innermost termination. Therefore, it is beneficial if it suffices to prove *innermost termination instead of termination*. In the previous work of [Gra95], classes of TRSs have been identified where this switch can be done.

   We extend that work to the DP framework, which results in two major benefits. First, in the DP framework there are strictly less requirements to switch to innermost termination than on the level of TRSs. And second, due to the modular structure of the DP framework, one can simplify the original termination problem before trying to switch to innermost termination. For that reason, the conditions for this switch are satisfied much more often, since only a *subproblem* instead of the *whole* TRS has to satisfy the requirements.

(v) The most important technique to prove termination is to use well-founded orders. Then termination can be shown by finding an order on terms that satisfies a certain set of constraints. While there is still active research on developing new kinds of orders (e.g., there are novel orders which are based on negative polynomial interpretations [HM07] or on matrix interpretations [EWZ06, HW06]) we focus on the orthogonal goal to *reduce the set of constraints.*

Our work is a significant extension of the work of [Urb01]: due to our results the constraints for termination and for innermost termination are almost identical. Thus, proving full termination becomes almost as easy as proving only innermost termination. Moreover, we develop techniques to reduce the set of constraints for both termination and innermost termination proofs further, e.g., by considering that certain positions of terms are ignored by the order. This is especially important for programs using an accumulator, and in that way the resulting set of constraints is often satisfied by the simple embedding order, whereas the original set of constraints is not solvable by any order that is currently amenable to automation.

(vi) Some termination techniques of the DP approach transform dependency pairs, namely the techniques of instantiation, rewriting, and narrowing [AG00, GA01]. We develop the additional new transformation of *forward instantiation*, which is sometimes crucial for a successful termination proof.

But even more important is our *extension of the narrowing transformation.* Compared to the original narrowing transformation, our extension produces far less new pairs, which is often required to successfully prove termination, especially if one considers programs using an accumulator.

(vii) To analyze programs of higher-order functional languages like Haskell, one can transform them into *applicative TRSs* [GSST06]. Since standard termination techniques often fail on TRSs of this special form, we design new dedicated termination techniques, including an extension of the result in [KKSV96] to transform applicative TRSs back to standard TRSs.

(viii) *Semantic labeling* [Zan95] is a powerful technique to prove termination of TRSs. We completely integrate this important technique into the DP framework, including a generalization to $\mathcal{Q}$-restricted rewriting for both models and quasi-models, and including new completeness results. The benefit is that one can now apply semantic labeling at any time during a termination proof, and one often has to search for models or quasi-models for only a small subproblem. In this way, semantic labeling becomes even more powerful.

(ix) Finally, we develop techniques to disprove termination. Often, the reason for nontermination is a *loop*, a notion we generalize from unrestricted rewriting to $\mathcal{Q}$-restricted rewriting. While up to now it was not even known whether it is decidable if a given loop is also a loop for innermost rewriting, we present a *decision procedure* for the more general case of $\mathcal{Q}$-restricted rewriting. Although the correctness proof of the procedure is highly non-trivial, the resulting algorithm is easy to implement.

> Moreover, we integrate the concept of a loop into the DP framework, whereas it was originally defined for TRSs. One benefits from this integration since one can use all preceding termination techniques to detect those parts of a TRS which are possibly non-terminating. In this way, one often has to search for loops only in a small part of the initial system.
>
> Thus, with our contributions one can efficiently disprove termination of programs, even if one fixes the evaluation strategy to innermost.

Major parts of these contributions have already been published as joint work in 17 articles in international journals or in conference proceedings [CSL$^+$06, FGM$^+$07, GSST06, GST06, GTS05a, GTS05b, GTSF03, GTSF04, GTSF06, GTSS07, SGST06, STA$^+$07, TG03, TG05, TGS04, TM07, TZGS07].

However, the thesis does not contain all the material of these articles, since here we focus only on those parts of our work which aim at proving termination of TRSs. To be more precise, the thesis does not present the results of our papers about transforming programs to TRSs, and we do not provide all details about the automation of our methods.

But on the other hand, the thesis contains many important new contributions which have been developed solely by the author and which are unpublished up to now. They include the new semantic notions and the improved estimations in contribution (iii), the extension of the narrowing transformation in (vi), the integration and generalization of semantic labeling in (viii), and the decision procedure of (ix) to detect loops under strategies.

All techniques presented in this thesis can be efficiently automated, where for the major search problems one can often use SAT encodings and analyze termination with the help of modern SAT solvers. This has been done in our implementation of the DP framework: AProVE [GST06, GTSF04] is a fully automated tool for termination analysis, which is developed at the Research Group Computer Science 2, RWTH Aachen University. Being one of the leaders of the AProVE group, the author has designed major parts of AProVE and contributed over 60,000 lines of Java code to the AProVE source.

In the years 2004 – 2007 AProVE participated in the international Termination Competition [MZ07]. In this annual competition several termination tools try to solve as many termination problems as possible from the termination problem data base (TPDB) [TPDB], a collection of termination problems from several sources and different areas of computer science. For each problem there is a short amount of time to give one of the possible answers “Yes, it terminates”, “No, it does not terminate”, or “Don’t know” together with a corresponding proof. In every year AProVE was able to prove and to disprove termination of more TRSs than any other tool. The fact that without our contributions AProVE would not have been the winner of the competitions clearly demonstrates that the results of this thesis indeed improve the power of automated termination analysis significantly.

The thesis is organized as follows. We start in Chapter 2 by motivating and formally introducing $\mathcal{Q}$-restricted rewriting and the DP framework (contributions (i) and (ii)). In this framework one investigates *DP problems* instead of TRSs. Then in Chapters 3 – 8 the various termination techniques are developed, explained, and formulated in a uniform way as so-called *processors*. At the end of each of these chapters there is a short summary. They contain remarks on the automation of the presented processors, comparisons with related work, and possible ideas for future work. Moreover, for each contribution it is remarked, where it has been published or whether it is unpublished up to now.

Processors which always simplify DP problems and which have a low computational complexity are shown in Chapter 3, including the new processor to switch to innermost termination (contribution (iv)). Additionally, most of the new semantic notions as well as the corresponding improved estimations (contribution (iii)) are introduced in that chapter. The processors of Chapter 4 also simplify DP problems, but to apply them, one needs to find orders satisfying corresponding sets of constraints, which most often is at least an NP-hard problem. Here, we also illustrate how one can reduce the set of generated constraints (contribution (v)). The dependency pair transformations are investigated in Chapter 5, including the new forward instantiation processor and the improved narrowing transformation (contribution (vi)). They are especially useful if one analyzes TRSs which define functions by using tests and selectors instead of using pattern matching. New techniques to handle higher-order functions which are encoded by applicative TRSs (contribution (vii)) are discussed in Chapter 6. Moreover, in that chapter we again demonstrate ways to decrease the set of generated constraints (contribution (v)). How to adapt semantic labeling to the DP framework (contribution (viii)) can be seen in Chapter 7. Finally, techniques to disprove termination are developed in Chapter 8, including the decision procedure to detect whether a loop respects the evaluation strategy (contribution (ix)).

The thesis ends with a conclusion in Chapter 9, which additionally contains some remarks about our termination prover AProVE along with an outlook on future work. Moreover, a strategy how to combine all the different processors is presented.

All proofs can be found in the appendix.

2. The Dependency Pair Framework

In this chapter we present the basis for our method to analyze termination of term rewrite systems (TRSs).

First, in Section 2.1 we introduce basic notions and notations including $\mathcal{Q}$-restricted rewriting [GTS05a], and we give some initial arguments why we investigate termination of $\mathcal{Q}$-restricted rewriting instead of full termination or innermost termination. The main reasons are that with $\mathcal{Q}$-restricted rewriting both full rewriting and innermost rewriting can be represented in a uniform way. And even more important, certain techniques are more powerful for $\mathcal{Q}$-restricted rewriting than for innermost rewriting, since innermost rewriting is not expressive enough.

Afterwards, we recapitulate the basics of dependency pairs in Section 2.2. There we present the two main components of our dependency pair framework to prove termination of TRSs. The problems we are working on are so called "dependency pair problems", or "DP problems" for short. Then a termination proof in the dependency pair framework is nothing more but a repeated transformation of DP problems into simpler DP problems by "processors", where one starts with the initial DP problem and ends if no DP problems remains. Of course, it will be possible to disprove termination in the dependency pair framework, too. It turns out that in this way, one can perform termination proofs in a modular way, and the framework can easily be extended by developing new processors.

2.1. $\mathcal{Q}$-Restricted Rewriting

A signature $\mathcal{F}$ is a finite set of function symbols where each $f \in \mathcal{F}$ has an *arity*, written $ar(f)$. The set $\mathcal{T}(\mathcal{F}, \mathcal{V})$ denotes the set of terms over $\mathcal{F}$ and the infinite set of variables $\mathcal{V}$. For a term t we write $\mathcal{V}(t)$ for the set of variables occurring in t, the set of *positions* of t is $Pos(t)$, and t is called *linear* iff each variable in t occurs at most once. The *root* of a non-variable term is defined as $root(f(t_1, \ldots, t_n)) = f$. A *substitution* is a mapping from variables to terms which is homomorphically extended to a mapping from terms to terms. The domain of a substitution σ is the set $Dom(\sigma) = \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$. We often write $t\sigma$ instead of $\sigma(t)$ and we use $\{x_1/t_1, \ldots, x_n/t_n\}$ to represent substitutions with finite domain, but we also allow substitutions which have an infinite domain.

A TRS $\mathcal{R}$ is a finite set of rewrite rules $\ell \to r$ where ℓ and r are from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ and where every rule $\ell \to r$ satisfies the *variable condition*, i.e., $\ell \notin \mathcal{V}$ and $\mathcal{V}(\ell) \supseteq \mathcal{V}(r)$. A *collapsing rule* is a rule where the right-hand side is a variable. A TRS $\mathcal{R}$ is *left-linear*, resp. *right-linear*, iff all left-hand sides of $\mathcal{R}$ ($lhs(\mathcal{R})$), resp. right-hand sides of $\mathcal{R}$, are linear. The rewrite relation w.r.t. $\mathcal{R}$ is defined as $s \to_{\mathcal{R}} t$ iff there are a context C, a position p, and a substitution σ such that $s = C[\ell\sigma]_p$, $t = C[r\sigma]_p$, and $\ell \to r \in \mathcal{R}$. In that case the subterm $\ell\sigma$ is called a *redex*. To denote that rewriting is only allowed at a certain position p we use the notation $\to_{\mathcal{R},p}$, and similarly, if we only allow rewriting for a single rule $\ell \to r$, then we write $\to_{\ell \to r}$ instead of $\to_{\{\ell \to r\}}$. A term t is in *normal form* w.r.t. some binary relation $\to$ iff there is no s such that $t \to s$. The set $NF(\mathcal{R})$ of $\mathcal{R}$-normal forms consists of those terms which are in normal form w.r.t. $\to_{\mathcal{R}}$. A TRS $\mathcal{R}$ is *terminating* iff

$\to_{\mathcal{R}}$ is well-founded, i.e., if there is no infinite rewrite sequence $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} t_3 \to_{\mathcal{R}} \ldots$. We identify rules which only differ by their variable names and we assume that each occurrence of a rule is variable-renamed when used for rewriting, etc.

In contrast to $\to_{\mathcal{R}}$ (which we also call the *full* rewrite relation), there also is the *innermost* rewrite relation which is denoted by $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ and defined as $s \xrightarrow{\mathsf{i}}_{\mathcal{R}} t$ iff $s \to_{\mathcal{R}} t$ where all direct subterms of the redex are in $\mathcal{R}$-normal form.

We refer to [BN98, Ter03] for further details on term rewriting.

Example 2.1. The following TRS $\mathcal{R}$ of [AG00, Example 2] computes subtraction and division of natural numbers. Here, the naturals are encoded by $\mathsf{0}$ and s where the numbers $0, 1, 2, \ldots$ are represented by the terms $\mathsf{0}, \mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(\mathsf{0})), \ldots$.

$$\begin{array}{rcll}
\mathsf{minus}(x, \mathsf{0}) & \to & x & (1)\\
\mathsf{minus}(\mathsf{0}, \mathsf{s}(y)) & \to & \mathsf{0} & (2)\\
\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}(x, y) & (3)\\
\mathsf{div}(\mathsf{0}, \mathsf{s}(y)) & \to & \mathsf{0} & (4)\\
\mathsf{div}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(y))) & (5)
\end{array}$$

The computation of "$2 - 2 = 1$" corresponds to the following rewrite sequence

$$\begin{array}{rll}
\mathsf{div}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{s}(\mathsf{s}(\mathsf{0}))) & \to_{(5)} & \mathsf{s}(\mathsf{div}(\mathsf{minus}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0})), \mathsf{s}(\mathsf{s}(\mathsf{0}))))\\
& \to_{(3)} & \mathsf{s}(\mathsf{div}(\mathsf{minus}(\mathsf{0}, \mathsf{0}), \mathsf{s}(\mathsf{s}(\mathsf{0}))))\\
& \to_{(1)} & \mathsf{s}(\mathsf{div}(\mathsf{0}, \mathsf{s}(\mathsf{s}(\mathsf{0}))))\\
& \to_{(4)} & \mathsf{s}(\mathsf{0})
\end{array}$$

where in the last step the context $C = \mathsf{s}(\Box)$ and the substitution $\sigma = \{y/\mathsf{s}(\mathsf{0})\}$ is used for the reduction, for example. Since there is no possibility to continue the rewrite sequence any further, the term $\mathsf{s}(\mathsf{0})$ is in $\mathcal{R}$-normal form.

If we consider the term $t = \mathsf{div}(\mathsf{s}(\mathsf{minus}(\mathsf{0}, \mathsf{0})), \mathsf{s}(\mathsf{0}))$ then both reductions $t \to_{\mathcal{R}} \mathsf{div}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0}))$ and $t \to_{\mathcal{R}} \mathsf{s}(\mathsf{div}(\mathsf{minus}(\mathsf{minus}(\mathsf{0}, \mathsf{0}), \mathsf{0}), \mathsf{s}(\mathsf{0})))$ are possible, but only the former is an innermost reduction.

To handle different evaluation strategies (like innermost or full rewriting) in a uniform way, we introduce the following notion of *$\mathcal{Q}$-restricted rewriting*, cf. [GTS05a, Definition 1]. In $\mathcal{Q}$-restricted rewriting, one may only perform a rewrite step if the proper subterms of the redex are not reducible w.r.t. $\mathcal{Q}$ (i.e., if they are *$\mathcal{Q}$-normal forms*). This notion is particularly useful when defining techniques for innermost termination proofs later on.

Definition 2.2 ($\mathcal{Q}$-Normal Form, $\mathcal{Q}$-Restricted Rewriting, $\mathcal{Q}$-Termination)**.** *Let $\mathcal{Q}$ be a set of non-variable terms and $\mathcal{R}$ be a TRS. We define the set $NF(\mathcal{Q})$ of* $\mathcal{Q}$-normal forms *as the normal forms of the TRS $\{q \to \mathsf{a} \mid q \in \mathcal{Q}\}$*[1] *where a is some constant. Then the* $\mathcal{Q}$-restricted rewrite relation *is defined as $s \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t$ iff $s \to_{\mathcal{R},p} t$ for some position p such that all direct subterms of the redex $s|_p$ are in $\mathcal{Q}$-normal form. A TRS $\mathcal{R}$ is* $\mathcal{Q}$-terminating *iff $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ is well founded.*

---

[1] In [GTS05a] $\mathcal{Q}$ is a TRS and not just a set of non-variable terms. However, there it was already mentioned that one can drop the right-hand sides, since for determining whether a term is in normal form, it is sufficient to know the left-hand sides. We will illustrate that a set of terms $\mathcal{Q}$ is advantageous to a TRS $\mathcal{Q}$ in more detail directly after Definition 6.2 and after Definition 7.16.

Example 2.3. Consider $\mathcal{R} = \{\mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{a}), \mathsf{a} \to \mathsf{b}\}$. If $\mathcal{Q}$ contains the term a, then the step from $\mathsf{f}(\mathsf{a})$ to $\mathsf{f}(\mathsf{a})$ is no longer possible with $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ since the proper subterm a of the redex $\mathsf{f}(\mathsf{a})$ is not a $\mathcal{Q}$-normal form. Thus, $\mathcal{R}$ is $\mathcal{Q}$-terminating. On the other hand, if $\mathcal{Q} = \varnothing$, then any ordinary rewrite step is also possible with $\mathcal{Q}$-restricted rewriting and we obtain $\mathsf{f}(\mathsf{a}) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \mathsf{f}(\mathsf{a})$. Hence, then $\mathcal{R}$ is not $\mathcal{Q}$-terminating.

$\mathcal{Q}$-restricted rewriting subsumes both innermost and full rewriting. Full rewriting is $\mathcal{Q}$-restricted rewriting for $\mathcal{Q} = \varnothing$ and innermost rewriting is $\mathcal{Q}$-restricted rewriting with $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$ ($\to_{\mathcal{R}} = \xrightarrow{\varnothing}_{\mathcal{R}}$ and $\xrightarrow{\mathrm{i}}_{\mathcal{R}} = \xrightarrow{\mathit{lhs}(\mathcal{R})}_{\mathcal{R}}$). The following lemma states that $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ is "increasing" if $\mathcal{R}$ is "increasing" and $\mathit{NF}(\mathcal{Q})$ is "increasing".

Lemma 2.4 (Monotonicity of $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$)**.** *If $\mathcal{R} \subseteq \mathcal{R}'$ and $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{Q}')$ then $\xrightarrow{\mathcal{Q}}_{\mathcal{R}} \subseteq \xrightarrow{\mathcal{Q}'}_{\mathcal{R}'}$.*

Lemma 2.4 indicates that for termination proving it is always advantageous to get large sets $\mathcal{Q}$ as $\mathcal{Q} \supseteq \mathcal{Q}'$ implies $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{Q}')$ and hence, $\xrightarrow{\mathcal{Q}}_{\mathcal{R}} \subseteq \xrightarrow{\mathcal{Q}'}_{\mathcal{R}}$. Moreover, Lemma 2.4 already indicates why $\mathcal{Q}$-restricted rewriting is better suitable for termination analysis than innermost rewriting. There exist several techniques which can simplify termination proofs by removing rules from the TRS $\mathcal{R}$. For full rewriting and also for $\mathcal{Q}$-restricted rewriting, removal of rules is always advantageous, since it can never introduce non-termination (termination of $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ implies termination of $\xrightarrow{\mathcal{Q}}_{\mathcal{R}'}$ if $\mathcal{R}' \subseteq \mathcal{R}$). But for innermost rewriting, this is not true. For instance, by removing the rule $\mathsf{a} \to \mathsf{b}$ from the innermost terminating TRS $\mathcal{R}$ of Example 2.3, we result in a TRS $\mathcal{R}'$ that is not innermost terminating (hence, $\xrightarrow{\mathrm{i}}_{\mathcal{R}'} \not\subseteq \xrightarrow{\mathrm{i}}_{\mathcal{R}}$). Here, $\mathcal{Q}$-restricted rewriting has the advantage that the terms $\mathcal{Q}$ which restrict the set of possible redexes are separated from the rules $\mathcal{R}$ used for rewriting and thus, $\mathcal{R}$ and $\mathcal{Q}$ can be changed independently.

Often it is interesting whether we are at least as restricted as in the innermost case. A sufficient but not necessary requirement is $\mathcal{Q} \supseteq \mathit{lhs}(\mathcal{R})$. This property is often demanded in [GTS05a] to apply techniques that are only sound in the innermost case.[2] However, if we choose $\mathcal{Q} = \{\mathsf{a}\}$ in Example 2.3 then $\xrightarrow{\mathcal{Q}}_{\mathcal{R}} = \xrightarrow{\mathrm{i}}_{\mathcal{R}}$ but $\mathcal{Q} \not\supseteq \mathit{lhs}(\mathcal{R})$. Nevertheless, the application of techniques for innermost rewriting would be prohibited if they required $\mathcal{Q} \supseteq \mathit{lhs}(\mathcal{R})$.

That this flexibility is sometimes required when modeling real programing languages by term rewriting is demonstrated in the following extension of Example 2.1.

Example 2.5. Let $\mathcal{R}$ be the TRS of Example 2.1 with the following additional rule.

$$\mathsf{f}(x) \quad \to \quad \mathsf{f}(\mathsf{div}(\mathsf{0},\mathsf{0}))$$

To simulate innermost evaluation of programs we can of course set $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$. But then it possible to construct an infinite rewrite sequence:

$$\mathsf{f}(\mathsf{0}) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \mathsf{f}(\mathsf{div}(\mathsf{0},\mathsf{0})) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \mathsf{f}(\mathsf{div}(\mathsf{0},\mathsf{0})) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \dots$$

However, in programming languages the evaluation would stop with an error at the second step, and there is no infinite evaluation possible.

To solve this problem, one can model innermost evaluation of programs more accurately if we define $\mathcal{Q} = \{\mathsf{f}(x), \mathsf{div}(x,y), \mathsf{minus}(x,y)\}$. Then, there is no reduction of $\mathsf{f}(\mathsf{div}(\mathsf{0},\mathsf{0}))$ possible.

However, in that case $\mathcal{Q} \supseteq \mathit{lhs}(\mathcal{R})$ does not hold, but we still want to apply all methods to proof innermost termination since $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ is even more restricted than $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$.

---

[2] To be more precise, the used criterion in [GTS05a] is $\mathcal{Q} \supseteq \mathcal{R}$ as $\mathcal{Q}$ in [GTS05a] is a TRS and not just a set of non-variable terms.

To solve this problem here we use a precise criterion to characterize that $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ is at least as restricted as the innermost rewrite relation $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$, namely we demand $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$. However, as the sets $\mathit{NF}(\mathcal{Q})$ and $\mathit{NF}(\mathcal{R}) = \mathit{NF}(\mathit{lhs}(\mathcal{R}))$ are both infinite we need a way to decide this property. This can be done with the following criterion where we only have to ensure for the finitely many terms of $\mathit{lhs}(\mathcal{R})$ that they are not in $\mathcal{Q}$-normal form.

Lemma 2.6 (Deciding $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{Q}')$)**.** *Let $\mathcal{Q}$ and $\mathcal{Q}'$ be sets of terms. Then $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{Q}')$ iff $\mathcal{Q}' \cap \mathit{NF}(\mathcal{Q}) = \varnothing$.*

Using Lemma 2.6 we can indeed detect that for $\mathcal{Q} = \{\mathsf{a}\}$ we are in the innermost case in Example 2.3. The reason is that both left-hand sides a and $\mathsf{f}(\mathsf{a})$ of $\mathcal{R}$ contain the $\mathcal{Q}$-redex a, and thus, are not in $\mathcal{Q}$-normal form. In the same way it is detected that for the modified set $\mathcal{Q}$ in Example 2.5 innermost rewriting is less restrictive than $\mathcal{Q}$-restricted rewriting.

Moreover, Lemma 2.6 allows to reduce a set $\mathcal{Q}$ to a smaller set $\mathcal{Q}'$ without changing the corresponding rewrite relations, i.e., $\xrightarrow{\mathcal{Q}}_{\mathcal{R}} = \xrightarrow{\mathcal{Q}'}_{\mathcal{R}}$. Whenever a subterm of a term $q \in \mathcal{Q}$ can already be matched by another term q' in $\mathcal{Q}$ then one can safely delete q from $\mathcal{Q}$ resulting in a smaller set $\mathcal{Q}'$. By Lemma 2.6 we directly obtain $\xrightarrow{\mathcal{Q}}_{\mathcal{R}} = \xrightarrow{\mathcal{Q}'}_{\mathcal{R}}$. The advantage of small sets $\mathcal{Q}$ is that the property "$t \in \mathit{NF}(\mathcal{Q})$" can be determined more efficiently. Consider Example 2.3 with innermost strategy. We obtain $\mathcal{Q} = \mathit{lhs}(\mathcal{R}) = \{\mathsf{f}(\mathsf{a}), \mathsf{a}\}$. As $\mathsf{f}(\mathsf{a})$ contains the subterm a we can safely remove $\mathsf{f}(\mathsf{a})$ from $\mathcal{Q}$ and result in $\mathcal{Q}' = \{\mathsf{a}\}$. Although $\mathcal{Q}' \subset \mathcal{Q}$ we have the same rewrite relation $\xrightarrow{\mathsf{i}}_{\mathcal{R}} = \xrightarrow{\mathcal{Q}}_{\mathcal{R}} = \xrightarrow{\mathcal{Q}'}_{\mathcal{R}}$ as mentioned before.

Note that even if we start a termination proof for $\mathcal{Q} = \varnothing$ we will often have to consider termination problems where $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$ or with $\mathcal{Q} \supset \mathit{lhs}(\mathcal{R})$. The reason is that there are classes where termination and innermost termination are equivalent (so one obtains $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$), and afterwards one will remove rules of $\mathcal{R}$ (then $\mathcal{Q} \supset \mathit{lhs}(\mathcal{R})$). In these cases the $\mathcal{Q}$-restricted rewrite relation is at least as restricted as the innermost rewrite relation. When using the technique of semantic labeling we can even get termination problems where $\mathcal{Q}$ and $\mathit{lhs}(\mathcal{R})$ are incomparable, cf. Chapter 7. Thus, there is an urgent need to generalize the termination techniques from full and innermost rewriting to $\mathcal{Q}$-restricted rewriting.

2.2. Dependency Pairs

Now we extend the *dependency pair approach* to a *dependency pair framework* for the combination of arbitrary termination techniques. We refer to [AG00, GAO02] for further details on the dependency pair approach and to [GTS05a] for a simplified version of the dependency pair framework.

First, we present a termination criterion for $\mathcal{Q}$-restricted rewriting based on dependency pairs. For a TRS $\mathcal{R}$ the *defined symbols* are $\mathcal{D}_{\mathcal{R}} = \{\mathit{root}(\ell) \mid \ell \to r \in \mathcal{R}\}$. For every $f \in \mathcal{D}_{\mathcal{R}}$ we introduce a fresh *tuple symbol* $f^\sharp$, where $f^\sharp$ has the same arity as f and we often write F for $f^\sharp$. If $t = g(t_1, \ldots, t_m)$ with $g \in \mathcal{D}_{\mathcal{R}}$, we let $t^\sharp$ denote $g^\sharp(t_1, \ldots, t_m)$.

Definition 2.7 (Dependency Pair [AG00])**.** *The set of* dependency pairs *for a TRS $\mathcal{R}$ is $\mathit{DP}(\mathcal{R}) = \{\ell^\sharp \to t^\sharp \mid \ell \to C[t] \in \mathcal{R},\ \mathit{root}(t) \in \mathcal{D}_{\mathcal{R}},$ and t is no proper subterm of $\ell\}$.*[3]

---

[3] This definition differs from the classical definition of dependency pairs in [AG00] by integrating a recent observation of [Der04], that one does not have to build dependency pairs where the right-hand side is a proper subterm of the left-hand side.

Example 2.8. The defined symbols of the TRS is Example 2.1 are minus and div. Thus, one obtains the following dependency pairs:

$$\begin{aligned} \mathsf{MINUS}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{MINUS}(x,y) && (6)\\ \mathsf{DIV}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{MINUS}(x,y) && (7)\\ \mathsf{DIV}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{DIV}(\mathsf{minus}(x,y),\mathsf{s}(y)) && (8) \end{aligned}$$

To verify $\mathcal{Q}$-termination, we use the notion of *chains*. Intuitively, a dependency pair corresponds to a function call and a chain represents a possible sequence of calls that can occur in a reduction. For termination, we try to prove that there are no infinite chains.

In the following definition, $\mathcal{P}$ is a directed graph where the nodes are pairs. Note that initially, every node is a dependency pair. However, as we will later on see techniques which transform dependency pairs into arbitrary rewrite rules, we only speak about "pairs" and not about "dependency pairs". But we do not call the nodes of $\mathcal{P}$ "rules" to distinguish them from the rules of the TRS $\mathcal{R}$.

Definition 2.9 (Pair-Graph, Chain). *A* pair-graph *is a directed graph $\mathcal{P} = (N, E)$ where the nodes N are a finite set of pairs and $E \subseteq N \times N$ are the edges.*

For a pair-graph $\mathcal{P} = (N, E)$, a set of terms $\mathcal{Q}$, and a TRS $\mathcal{R}$ a (possibly infinite) sequence of pairs $s_1 \to t_1, s_2 \to t_2, \ldots$ is a $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain *iff $(s_i \to t_i, s_{i+1} \to t_{i+1}) \in E$, and there is a substitution σ such that $t_i\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} s_{i+1}\sigma$ for all i and all $s_i\sigma$ are in $\mathcal{Q}$-normal form. A chain is* minimal *iff all $t_i\sigma$ are terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$.*

We often identify a pair-graph $\mathcal{P} = (N, E)$ with the set of pairs N. In this way $s \to t \in \mathcal{P}$ is a shorthand for $s \to t \in N$, and $\mathcal{P} \setminus \{s \to t\}$ is the pair-graph $(N \setminus \{s \to t\}, E')$ where E' is like E but one removes all edges that are adjacent to the node $s \to t$, i.e. $E' = E \cap (N \setminus \{s \to t\})^2$. The graph operations $\setminus$ and $\subseteq$ are defined component-wise: $(N, E) \subseteq (N', E')$ iff $N \subseteq N'$ and $E \subseteq E'$, and $(N, E) \setminus (N', E') = (N \setminus N', E \setminus E')$. We also use a set of pairs as a pair-graph where we implicitly assume that there are edges between every two (possibly identical) pairs.

Note that the pair-graph is new compared to the standard definition of a chain [AG00, GTS05a]. Up to now, $\mathcal{P}$ always was just a set of pairs without any graph structure. The reason for changing from sets of pairs to a graph over pairs is mainly improved efficiency and modularity (cf. the discussion on page 19 and Example 5.4), but sometimes even the power will be increased (cf. Example 4.40 and Example 5.12).

Example 2.10. If $\mathcal{Q} \subseteq \mathit{lhs}(\mathcal{R})$ then the TRS of Example 2.8 has the following chain which consists of two occurrences of the dependency pair (8).

$$\begin{aligned} \mathsf{DIV}(\mathsf{s}(x_1),\mathsf{s}(y_1)) &\to \mathsf{DIV}(\mathsf{minus}(x_1,y_1),\mathsf{s}(y_1)),\\ \mathsf{DIV}(\mathsf{s}(x_2),\mathsf{s}(y_2)) &\to \mathsf{DIV}(\mathsf{minus}(x_2,y_2),\mathsf{s}(y_2)) \end{aligned}$$

The reason is that $\mathsf{DIV}(\mathsf{minus}(x_1,y_1),\mathsf{s}(y_1))\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} \mathsf{DIV}(\mathsf{s}(x_2),\mathsf{s}(y_2))\sigma$ holds for some substitution σ (e.g., $\sigma(x_1) = \mathsf{s}(\mathsf{0})$ and $\sigma(x_2) = \sigma(y_i) = \mathsf{0}$ for $i \in \{1,2\}$) such that all instantiated left-hand sides $\mathsf{DIV}(\mathsf{s}(x_i),\mathsf{s}(y_i))\sigma$ are in $\mathcal{Q}$-normal form. Moreover, the chain is minimal, since all instantiated right-hand sides of the dependency pairs are terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$.

As mentioned above, termination corresponds to absence of infinite chains. Here, it suffices to consider minimal chains, since minimal non-terminating terms (whose proper subterms are terminating) correspond to infinite minimal chains. The termination criterion for dependency pairs [AG00, Theorems 6 and 31] can easily be generalized to $\mathcal{Q}$-restricted rewriting.

Theorem 2.11 (Termination Criterion [GTS05a]). *The following three properties are equivalent.*

(i) $\mathcal{R}$ *is* $\mathcal{Q}$*-terminating*

(ii) there is no infinite $(DP(\mathcal{R}), \mathcal{Q}, \mathcal{R})$*-chain*

(iii) there is no infinite minimal $(DP(\mathcal{R}), \mathcal{Q}, \mathcal{R})$*-chain*

The termination criterion in Theorem 2.11 states that to prove $\mathcal{Q}$-termination, it is sufficient to show that there is no infinite *minimal* chain. But in order to *disprove* termination, it is enough to find any infinite chain (which may also be non-minimal).

With this criterion, we can now state the dependency pair framework. The basic idea of this framework is to examine a pair-graph $\mathcal{P}$, the set of terms $\mathcal{Q}$, and the TRS $\mathcal{R}$ and to prove absence of infinite (minimal) $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chains instead of examining the relation $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ directly. There are several advantages of this approach compared to a direct termination proof.

For example, consider a direct termination proof where one has to find a reduction order satisfying the constraints $\ell \succ r$ for all rules of $\mathcal{R}$. Note that common reduction orders fail on the constraints for Example 2.8 whereas it is easily possible to prove absence of infinite chains using the techniques of Chapter 4 in combination with the embedding order. This is possible since in contrast to a direct termination proof, the order $\succ$ does not have to be monotonic when analyzing chains.

Moreover, in the direct termination proof one has to solve termination of minus and div together. In contrast, it will be possible to decompose a so-called *dependency pair problem* (DP problem, for short) into several independent sub-problems, e.g., one problem for MINUS and one for DIV. These problems can then be solved separately using different techniques, which leads to a very modular approach to termination proving.

Formally, a DP problem consists of a pair-graph $\mathcal{P}$, a set of terms $\mathcal{Q}$, a TRS $\mathcal{R}$ (where initially, $\mathcal{P} = DP(\mathcal{R})$), and a flag $f \in \{\mathbf{m}, \mathbf{a}\}$ which stands for "$\underline{\mathbf{m}}$inimal" or "$\underline{\mathbf{a}}$rbitrary". Initially, we have $f = \mathbf{m}$. Our goal is to show that there is no infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain if $f = \mathbf{m}$ and that there is no infinite (possibly non-minimal) $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain if $f = \mathbf{a}$. In this case, we call the problem *finite*.

A DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ that is not finite is called *infinite*. But additionally, $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ is already *infinite* whenever $\mathcal{R}$ is not $\mathcal{Q}$-terminating. So in particular, the existence of any (possibly non-minimal) infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$-chain suffices to conclude that $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ is infinite, even if $f = \mathbf{m}$. While the initial DP problem $(DP(\mathcal{R}), \mathcal{Q}, \mathcal{R}, f)$ is either finite or infinite, other DP problems $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ which can occur in termination proofs can be both finite and infinite.

Example 2.12. As an example consider the TRS $\mathcal{R} = \{\mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{a}), \mathsf{a} \to \mathsf{a}\}$, let $\mathcal{Q} = \varnothing$, and let $\mathcal{P}$ consist of the dependency pair $\mathsf{F}(\mathsf{a}) \to \mathsf{F}(\mathsf{a})$. Then $\mathsf{F}(\mathsf{a}) \to \mathsf{F}(\mathsf{a})$, $\mathsf{F}(\mathsf{a}) \to \mathsf{F}(\mathsf{a}), \ldots$ is an infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain, but there is no infinite *minimal* $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain, since $\mathsf{F}(\mathsf{a})$ is not terminating. So the dependency pair problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$ is both finite and infinite.

Such DP problems do not lead to any difficulties. If one detects an infinite DP problem during a termination proof, one can always abort the proof, since termination has been disproved (provided that all proof steps were "complete", i.e., that they preserved the termination behavior). If the problem is both finite and infinite, then even if one only considers it as being finite, the proof will still be correct, since then there exists another resulting dependency pair problem which is infinite and not finite. The reason is that by Theorem 2.11, non-termination implies that there exists an infinite (minimal) chain. Indeed, when proving termination of the TRS of Example 2.12 one would also obtain a DP problem with the infinite minimal chain $\mathsf{A} \to \mathsf{A}$, $\mathsf{A} \to \mathsf{A}, \ldots$

Termination techniques should now operate on dependency pair problems instead of TRSs. They transform a DP problem into a new set of problems which then have to be solved instead. Alternatively, they can also return the answer "no". We refer to such techniques as *processors.*

Definition 2.13 (DP Problem)**.** *A* DP problem $\mathcal{D}$ *is a quadruple* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ *that consists of a pair-graph* $\mathcal{P}$*, a set of terms* $\mathcal{Q}$*, a TRS* $\mathcal{R}$*, and a flag* $f \in \{\mathbf{m}, \mathbf{a}\}$*. A DP problem* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ *is* finite *iff there is no infinite minimal* $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$*-chain and* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{a})$ *is* finite *iff there is no infinite* $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$*-chain. A DP problem* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ *is* infinite *iff it is not finite or if* $\mathcal{R}$ *is not* $\mathcal{Q}$*-terminating.*

A processor *is a function Proc which takes a DP problem as input and returns either a set of DP problems or the result "*no*". A processor Proc is* sound *if for all DP problems* $\mathcal{D}$*,* $\mathcal{D}$ *is finite whenever Proc*$(\mathcal{D})$ *is not "*no*" and all DP problems in Proc*$(\mathcal{D})$ *are finite. A processor Proc is* complete *if for all DP problems* $\mathcal{D}$*,* $\mathcal{D}$ *is infinite whenever Proc*$(\mathcal{D})$ *is "*no*" or when Proc*$(\mathcal{D})$ *contains an infinite DP problem.*

Thus, soundness is required in order to use a processor *Proc* to prove termination (in particular, to conclude that $\mathcal{D}$ is finite if *Proc*$(\mathcal{D}) = \varnothing$). Completeness is needed in order to use *Proc* to prove non-termination (in particular, to conclude that $\mathcal{D}$ is infinite if *Proc*$(\mathcal{D}) = \mathsf{no}$). Even if one is only interested in proving termination, completeness is still advantageous, since it ensures that one does not transform non-infinite DP problems into infinite ones (i.e., applying the processor does not "harm"). The reason for the above non-symmetric definition of "finite" and "infinite" is that in this way there are more finite resp. infinite DP problems and therefore, it becomes easier to detect (in)finiteness of a problem.

That a DP problem is already "infinite" if $\mathcal{R}$ is not $\mathcal{Q}$-terminating will be required for the completeness of the rewriting processor, of the narrowing processor, and of the processor to switch to termination (cf. Theorems 5.10, 5.19, and 8.9, and Example 5.15 in Chapter 5) and that a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ is already "finite" if there are no infinite *minimal* chains will be required for the soundness of many processors (cf. Theorems 3.14, 4.12, 4.18, 4.20, 4.32, 4.41, 6.17, and 6.22).

The fact that there are many processors relying on minimality clearly shows that a processor which changes the minimality flag from $\mathbf{m}$ to $\mathbf{a}$ has a severe drawback. In that case one should try to a develop an alternative processor with a similar result which does not change the minimality flag. How this can be done is discussed with the processor of Theorem 4.12 in Section 4.2 in more detail.

The following corollary introduces the dependency pair framework ("DP framework", for short). The idea is to start with the initial DP problem where $\mathcal{P} = DP(\mathcal{R})$ and $f = \mathbf{m}$. Then this problem is transformed repeatedly by sound DP processors. If the final processors return empty sets of DP problems, then termination is proved. If one of the processors returns "no" and all processors used before were complete, then one has proved that the original TRS is not $\mathcal{Q}$-terminating. The proof of Corollary 2.14 is immediate from Definition 2.13 and Theorem 2.11.

Corollary 2.14 (Dependency Pair Framework [GTS05a])**.** *Let* $\mathcal{R}$ *be a TRS and* $\mathcal{Q}$ *be a set of terms. We construct a tree whose nodes are labelled with DP problems or "*yes*"* *or "no" and whose root is labelled with* $(DP(\mathcal{R}), \mathcal{Q}, \mathcal{R}, \mathbf{m})$. *For every inner node labelled with* $\mathcal{D}$*, there is a sound processor Proc satisfying one of the following conditions:*

- $Proc(\mathcal{D}) = \mathsf{no}$ *and the node has just one child, labelled with "no"*
- $Proc(\mathcal{D}) = \varnothing$ *and the node has just one child, labelled with "yes"*
- $Proc(\mathcal{D}) \neq \mathsf{no}$, $Proc(\mathcal{D}) \neq \varnothing$*, and the children of the node are labelled with the DP problems in* $Proc(\mathcal{D})$

If all leaves of the tree are labelled with "yes", then $\mathcal{R}$ *is* $\mathcal{Q}$*-terminating. Otherwise, if there is a leaf labelled with "no" and if all processors used on the path from the root to this leaf are complete, then* $\mathcal{R}$ *is not* $\mathcal{Q}$*-terminating.*

Example 2.15. If $\mathcal{D}_0$ is the initial DP problem $(DP(\mathcal{R}), \mathcal{Q}, \mathcal{R}, \mathbf{m})$ and $Proc_0$, $Proc_1$, and $Proc_2$ are sound processors with $Proc_0(\mathcal{D}_0) = \{\mathcal{D}_1, \mathcal{D}_2\}$, $Proc_1(\mathcal{D}_1) = \varnothing$, and $Proc_2(\mathcal{D}_2) = \varnothing$ (left tree in Figure 2.16), then one can conclude termination. But if $Proc_1(\mathcal{D}_1) = \{\mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_5\}$ and $Proc_2(\mathcal{D}_2) = \mathsf{no}$, and both $Proc_0$ and $Proc_2$ are complete (right tree in Figure 2.16), then one can conclude non-termination.

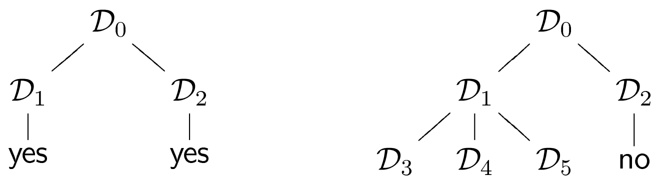


Figure 2.16.: Trees for Termination Proofs in the Dependency Pair Framework

In the remainder of the thesis, we present several sound processors which can be used for termination analysis within the DP framework. Of course, it is desirable to find processors which transform a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ into a set of "simpler" problems and whose application can never "harm". Therefore, we are particularly interested in processors which decrease $\mathcal{P}$, $NF(\mathcal{Q})$, and $\mathcal{R}$. As stated by Lemma 2.4, decreasing the set of rules $\mathcal{R}$ and decreasing $NF(\mathcal{Q})$ leads to a more restricted rewrite relation and thus, it can never transform a non-infinite DP problem into an infinite one. In other words, any processor which removes nodes and edges from $\mathcal{P}$, which removes rules from $\mathcal{R}$, and which adds terms to $\mathcal{Q}$ (and thereby decreases $NF(\mathcal{Q})$) is complete.

A corresponding lemma in the DP approach also holds for the termination case but not for the innermost case. Consider for example $\mathcal{P} = \{\mathsf{F}(\mathsf{a}) \to \mathsf{F}(\mathsf{a})\}, \mathcal{R} = \{\mathsf{a} \to \mathsf{b}\}$. There is no infinite innermost $\mathcal{P}$-chain over $\mathcal{R}$, but there is an infinite innermost $\mathcal{P}$-chain over the empty TRS. Thus, removing rules can introduce infinite chains and thereby prevent a successful innermost termination proof.

Lemma 2.17 (Completeness of Processors). *Let Proc be a processor where for all DP problems* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ *the result of* $Proc((\mathcal{P}, \mathcal{Q}, \mathcal{R}, f))$ *is not* no *and for all* $(\mathcal{P}', \mathcal{Q}', \mathcal{R}', f') \in Proc((\mathcal{P}, \mathcal{Q}, \mathcal{R}, f))$ *we have* $\mathcal{P}' \subseteq \mathcal{P}$, $NF(\mathcal{Q}') \subseteq NF(\mathcal{Q})$*, and* $\mathcal{R}' \subseteq \mathcal{R}$. *Then Proc is complete.*

Summary of Chapter 2

In this chapter we have introduced $\mathcal{Q}$-restricted rewriting which allows to represent full and innermost rewriting in a uniform way. The additional flexibility of $\mathcal{Q}$ also helps to model innermost evaluation in programming languages more accurately. Moreover, we have extended the DP approach of [AG00] to a DP framework which allows a modular combination of termination techniques by formulating them as processors.

Most of this work has already been published by us in [GTS05a], but there are some extensions which are only present in this thesis, namely the pair-*graph*, the *set* $\mathcal{Q}$, and the improved check to detect that $\mathcal{Q}$-restricted rewriting is more restrictive than innermost rewriting.

It remains to present and extend existing processors and to develop new processors. This will be done in the following chapters.

3. Processors Without Search

In this chapter we will see processors which always simplify DP problems as they never increase $\mathcal{P}$, $\mathcal{R}$, or $NF(\mathcal{Q})$.

The processors of Section 3.1 are based on the dependency graph, a concept which is well known already in the early work about dependency pairs [AG00]. Using the dependency graph one can decompose DP problems. For example, these processors allow to prove termination of DIV and MINUS in Example 2.8 independently. As the dependency graph is not computable, different estimations are proposed in [AG00, GTS05b, HM05, Mid01, Mid02]. The problem is that all these estimations have been developed for full- or for innermost-rewriting but not for $\mathcal{Q}$-restricted rewriting. Therefore, in Section 3.1 and Section 3.4 we present new estimations for dependency graphs which integrate most ideas of the known estimations[4] and which generalize these ideas to $\mathcal{Q}$-restricted rewriting.

An improved version of the processor of [GTS05a] which can enlarge the set $\mathcal{Q}$ to $lhs(\mathcal{R})$ is presented in Section 3.2. This processor allows to switch from full termination to innermost termination and subsumes the result of [Gra95] that for locally confluent overlay systems innermost termination and termination coincide.

In Section 3.3 we recapitulate the well-known concept of usable rules [AG00]. The usable rules are those rules that are used to evaluate the arguments of the right-hand sides of the dependency pairs. Using the corresponding processor one can replace the TRS $\mathcal{R}$ of a DP problem by the smaller TRS containing only the usable rules. For example, for MINUS there are no usable rules and for DIV only the minus-rules are usable in Example 2.8. The problem with the concept of usable rules is again that up to now they were only presented for full- and for innermost-rewriting. But the situation is even worse.

There are different definitions of usable rules [AG00, GTS05b, GTSF06][5] and it is unclear whether one can exchange the definition of usable rules in different papers without invalidating the theorems that make use of usable rules. To this end we present a new semantic version of usable rules and show that all previous versions of usable rules are estimations of the semantic version of usable rules. Moreover, we show for the various processors that rely on usable rules that they are correct for arbitrary estimations of usable rules (which over-approximate the set of usable rules). Then we present a new estimation in Section 3.3 for the general case of $\mathcal{Q}$-restricted rewriting which is better than every previous version of usable rules. A similar solution will be presented in Section 3.1 in order to combine the different syntactic versions of *Cap*.

Finally, in Section 3.5 we will show processors to remove terms from $\mathcal{Q}$. Although their application results in larger sets $NF(\mathcal{Q})$, they do not harm, since only terms are removed which cannot block reductions of $\mathcal{R}$ and $\mathcal{P}$ any more. On the contrary, often these processors can transform DP problems with $NF(\mathcal{Q}) \subset NF(\mathcal{R})$ to DP problems with $NF(\mathcal{Q}) = NF(\mathcal{R})$. In this way, all those techniques become applicable which are only available for innermost termination.

---

[4] We do not integrate the estimations based on tree automata of [Mid01].

[5] We do not consider the *generalized usable rules* of [GTSS07, Definition 5] in this chapter since these rules depend on a *reduction pair*, a concept we will introduce in the next chapter.

What all processors of this chapter have in common is that they can be automated efficiently, as there are no challenging search problems arising when implementing these processor.

To illustrate the different processors we use the following running example throughout this chapter. Note that we do not continue with the termination proof of Example 2.8 as that TRS was already shown to be terminating in many papers using standard orders with techniques that have already been developed in [AG00].

Example 3.1. We consider a TRS for addition and multiplication of natural numbers combined with a variant of the TRS of Toyama [Toy87]. Here, multiplication is computed with the help of an accumulator in the third argument.

$$\begin{aligned}
\mathsf{isZero}(\mathsf{0}) &\to \mathsf{true} && (9)\\
\mathsf{isZero}(\mathsf{s}(x)) &\to \mathsf{false} && (10)\\
\mathsf{plus}(\mathsf{0}, y) &\to y && (11)\\
\mathsf{plus}(\mathsf{s}(x), y) &\to \mathsf{s}(\mathsf{plus}(x, y)) && (12)\\
\mathsf{plus}(\mathsf{plus}(x, y), z) &\to \mathsf{plus}(x, \mathsf{plus}(y, z)) && (13)\\
\mathsf{times}(x, y) &\to \mathsf{mul}(x, y, \mathsf{0}) && (14)\\
\mathsf{mul}(x, y, z) &\to \mathsf{if}(\mathsf{isZero}(x), x, y, z) && (15)\\
\mathsf{if}(\mathsf{true}, x, y, z) &\to z && (16)\\
\mathsf{if}(\mathsf{false}, \mathsf{s}(x), y, z) &\to \mathsf{mul}(x, y, \mathsf{plus}(y, z)) && (17)\\
\mathsf{f}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(\mathsf{0})), x) &\to \mathsf{f}(x, x, x) && (18)
\end{aligned}$$

3.1. Dependency Graph

We now present a processor to decompose a DP problem into several separate sub-problems. To this end, one tries to determine which pairs can follow each other in chains by constructing a so-called *dependency graph.* In contrast to the classical DP approach, now the dependency graph can be (re-)computed at any time during the termination proof. This leads to very modular proofs, since one may always decompose DP problems into sub-problems which can be solved independently, e.g., by different DP processors.

Definition 3.2 (Dependency Graph). *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ be a DP problem where the pair-graph $\mathcal{P}$ has the set of nodes N. The* $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-dependency graph *is a pair-graph (N, E) and E contains an edge from $s \to t$ to $u \to v$ iff $s \to t, u \to v$ is a $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain (iff $s \to t, u \to v$ is a path in $\mathcal{P}$ and there is substitution σ such that $t\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} u\sigma$ and $\{s\sigma, u\sigma\} \subseteq NF(\mathcal{Q})$).*

Obviously, whenever there is no edge between $s \to t$ and $u \to v$ in the dependency graph then one can delete all corresponding edges in $\mathcal{P}$. Since the dependency graph is in general not computable, for automation one constructs an *estimated* graph. To this end, one has to approximate whether two pairs $s \to t$ and $u \to v$ form a $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain. In this case, one draws an arc from $s \to t$ to $u \to v$. In this thesis we only consider sound estimations that contain the real dependency graph. Then, one may remove all edges in $\mathcal{P}$ that are not contained in the estimated graph.

Theorem 3.3 (Processors Based on the Dependency Graph)**.** *The following processor Proc is sound and complete. For a DP problem* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ *with* $\mathcal{P} = (N, E)$ *the processor Proc returns* $\{(\mathcal{P} \cap \mathcal{P}', \mathcal{Q}, \mathcal{R}, f)\}$*, where* $\mathcal{P}'$ *is an estimated* $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$*-dependency graph.*

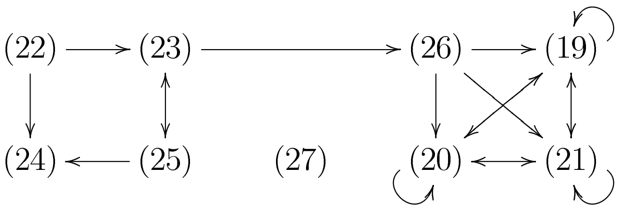
The next step is to exploit the graph-structure of the pair-graph of a given DP problem. An infinite chain corresponds to an infinite path in the pair-graph. And as this graph is finite, every infinite chain in the graph must correspond to a cycle in the graph. Therefore, for every DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ it is sufficient to prove absence of infinite (minimal) chains for maximal cycles (so-called *strongly connected components*, SCCs) of $\mathcal{P}$. To be more precise, a subgraph $\mathcal{P}'$ of $\mathcal{P}$ is called a *cycle* iff in $\mathcal{P}'$ every node n is reachable from every other node (including n itself). A cycle $\mathcal{P}'$ is called an *SCC* if $\mathcal{P}'$ is not a proper subgraph of any other cycle.

Theorem 3.4 (Processor Based on Graph Decomposition)**.** *The following processor Proc is sound and complete. For a DP problem* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ *with* $\mathcal{P} = (N, E)$ *the processor Proc returns* $\{(\mathcal{P}_1, \mathcal{Q}, \mathcal{R}, f), \ldots, (\mathcal{P}_n, \mathcal{Q}, \mathcal{R}, f)\}$*, where* $\mathcal{P}_1, \ldots, \mathcal{P}_n$ *are the SCCs of* $\mathcal{P}$*.*

Example 3.5. For the TRS of the running example of this chapter (Example 3.1) we obtain the following set of dependency pairs.

$$\mathsf{PLUS}(\mathsf{s}(x), y) \to \mathsf{PLUS}(x, y) \tag{19}$$
$$\mathsf{PLUS}(\mathsf{plus}(x, y), z) \to \mathsf{PLUS}(x, \mathsf{plus}(y, z)) \tag{20}$$
$$\mathsf{PLUS}(\mathsf{plus}(x, y), z) \to \mathsf{PLUS}(y, z) \tag{21}$$
$$\mathsf{TIMES}(x, y) \to \mathsf{MUL}(x, y, \mathsf{0}) \tag{22}$$
$$\mathsf{MUL}(x, y, z) \to \mathsf{IF}(\mathsf{isZero}(x), x, y, z) \tag{23}$$
$$\mathsf{MUL}(x, y, z) \to \mathsf{ISZERO}(x) \tag{24}$$
$$\mathsf{IF}(\mathsf{false}, \mathsf{s}(x), y, z) \to \mathsf{MUL}(x, y, \mathsf{plus}(y, z)) \tag{25}$$
$$\mathsf{IF}(\mathsf{false}, \mathsf{s}(x), y, z) \to \mathsf{PLUS}(y, z) \tag{26}$$
$$\mathsf{F}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(\mathsf{0})), x) \to \mathsf{F}(x, x, x) \tag{27}$$

The dependency graph of the initial DP problem has the following structure.



This graph contains two SCCs. The first SCC is the subgraph containing the recursive dependency pairs for multiplication (23) and (25) and the second SCC contains all dependency pairs for the addition rules (19), (20), and (21). Thus, when applying Theorem 3.3 and Theorem 3.4 we obtain the two new DP problems $((23) \leftrightarrow (25), \varnothing, \mathcal{R}, \mathbf{m})$ and $(\{(19), (20), (21)\}, \varnothing, \mathcal{R}, \mathbf{m})$.[6] From now on we can prove termination of addition and multiplication independently.

In [GTS05a] the previous two theorems were integrated in one theorem, which computed the SCCs $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of the (estimated) dependency graph and then returned the set of pairs of every $\mathcal{P}_i$ as result. Note that this combination was necessary as $\mathcal{P}$ was a set of pairs and there was no possibility to store the structure of the (estimated) dependency graph in $\mathcal{P}$. This older approach has the disadvantage that when using the combined theorem a second time in a termination proof, one has to recompute all edges. However, as we have defined $\mathcal{P}$ to be a graph, we just have to check the connections between pairs that are still connected. So in Example 3.5 we will never again check whether (23) is connected to itself. Thus, the graph structure increases efficiency. For an example where this effect occurs we refer to a later state in the termination proof of the running example of this section (Example 3.22).

---

[6] Recall that the set $\{(19), (20), (21)\}$ represents the pair-graph where every two (possibly identical) pairs are connected.

Moreover, the graph structure increases modularity. It is now easily possible to separately apply various (possible incomparable) dependency graph estimations to delete all edges that can be removed by at least one estimation. Of course, it is possible to develop a processor that integrates all techniques to delete edges and performs this accumulated deletions of edges internally. However, this would require a large combined theorem which has to be updated with every new technique for edge deletion that is discovered in the future. And such a combined theorem is clearly less desirable than presenting many separate ideas to delete edges and then combine them in a modular way.

Still, the question how to estimate the dependency graph remains. In the classical dependency pair approach, several such approximations were developed and for example, all of them would return the graph given in Example 3.5 with an additional edge from (27) to itself.[7] However, instead of $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chains, here one only considered chains where $\mathcal{Q} = \varnothing$ (for full termination) or where $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$ (for innermost termination). The latter were called "*innermost* chains". By Lemma 2.4, every $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain is also a $(\mathcal{P}, \varnothing, \mathcal{R})$-chain (i.e., an ordinary chain in the classical dependency pair approach) and if $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$, it is also an innermost chain. Thus, all existing methods to (over-)approximate chains in the dependency pair approach can also be used to approximate $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chains for any $\mathcal{Q}$. Moreover, if $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$, then all approximations for innermost chains can be applied as well. Hence, one can still use the existing estimation techniques for (innermost) dependency graphs in order to estimate $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-dependency graphs.

The basic idea of most current estimations is the following. To estimate whether two pairs $s \to t$ and $u \to v$ form a chain, essentially it must be checked whether there is some substitution σ such that $s\sigma$ and $u\sigma$ are in $\mathcal{Q}$-normal form and $t\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} u\sigma$. To this end it is approximated what the term $t\sigma$ looks like after a reduction. Therefore, a function *Cap*[8] replaces all subterms of t at position p by a fresh variable, whenever in the reduction of $t\sigma$ it is possible to perform a rewrite step at position p. Then it is checked whether the resulting term $\mathit{Cap}(t)$ can be unified with u. If this is not the case it can be guaranteed that $s \to t$ and $u \to v$ do not form a chain. Moreover, in the innermost case the information that $s\sigma$ and $u\sigma$ are normal forms is sometimes also integrated into *Cap*. This is done by providing an additional parameter $\mathcal{S}$ for *Cap* where $\mathcal{S}$ contains terms like s and u which are instantiated to normal forms. The set $\mathcal{S}$ can also be used if $\mathit{Cap}^{\mathcal{S}}(t)$ and u are unifiable. If the mgu instantiates a term of $\mathcal{S}$ such that this instance is not in normal form then there can also be no chain. We illustrate both uses of $\mathcal{S}$ in the following small example.

---

[7] For the approximation of [Mid01] we considered the three presented approximation mappings s, nv, and g.

[8] In [AG00] one additionally uses the function *Ren*. But as in [GTS05b], we here combine the traditional functions *Ren* and *Cap* of [AG00] into one new function *Cap*.

Example 3.6. Let $\mathcal{R}$ consist of the following rules and let $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$.

$$\begin{array}{rcl}
\mathsf{s}(\mathsf{p}(x)) & \to & x \\
\mathsf{p}(\mathsf{s}(x)) & \to & x \\
\mathsf{isNegative}(\mathsf{p}(x)) & \to & \mathsf{true} \\
\mathsf{f}(\mathsf{s}(x), \mathsf{true}) & \to & \mathsf{f}(\mathsf{s}(x), \mathsf{isNegative}(x)) \\
\mathsf{g}(\mathsf{s}(y)) & \to & \mathsf{h}(y) \\
\mathsf{h}(\mathsf{p}(z)) & \to & \mathsf{g}(\mathsf{s}(\mathsf{p}(z)))
\end{array}$$

Here, s and p denote the successor- and the predecessor-function on the integers, and $\mathsf{isNegative}$ checks whether a number is negative. To demonstrate the use of the set $\mathcal{S}$ for *Cap*, we consider the connections between the following three DPs.

$$\mathsf{F}(\mathsf{s}(x), \mathsf{true}) \to \mathsf{F}(\mathsf{s}(x), \mathsf{isNegative}(x)) \tag{28}$$
$$\mathsf{G}(\mathsf{s}(y)) \to \mathsf{H}(y) \tag{29}$$
$$\mathsf{H}(\mathsf{p}(z)) \to \mathsf{G}(\mathsf{s}(\mathsf{p}(z))) \tag{30}$$

We first check whether (28) has an outgoing edge in the dependency graph. Therefore we compute $\mathit{Cap}^{\mathcal{S}}(\mathsf{F}(\mathsf{s}(x), \mathsf{isNegative}(x)))$ for $\mathcal{S} = \{\mathsf{F}(\mathsf{s}(x), \mathsf{true}), \dots\}$. Obviously, the subterm $\mathsf{isNegative}(x)\sigma$ can only be reduced if x is instantiated by $\mathsf{p}(\dots)$. But this results in a conflict w.r.t. $\mathcal{S}$, since then the term in $\mathcal{S}$ is instantiated to $\mathsf{F}(\mathsf{s}(\mathsf{p}(\dots), \mathsf{true}))$ which is not in normal form. Hence, due to $\mathcal{S}$ we can detect that $\mathit{Cap}^{\mathcal{S}}(\mathsf{F}(\mathsf{s}(x), \mathsf{isNegative}(x))) = \mathsf{F}(\mathsf{s}(x), \mathsf{isNegative}(x))$. And since that term is not unifiable with any left-hand side of the three dependency pairs, we have proven that (28) has no outgoing edges.

Another effect can be seen when checking the connection between (29) and (30). There, we have $\mathcal{S} = \{\mathsf{G}(\mathsf{s}(y)), \mathsf{H}(\mathsf{p}(z))\}$ and the term $\mathit{Cap}^{\mathcal{S}}(\mathsf{H}(y)) = \mathsf{H}(y)$ is unifiable with $\mathsf{H}(\mathsf{p}(z))$, the left-hand side of (30). However, the mgu instantiates y by $\mathsf{p}(z)$ and thus, the term $\mathsf{G}(\mathsf{s}(y)) \in \mathcal{S}$ is instantiated such that it is not in $\mathcal{Q}$-normal form any more. Hence, again due to $\mathcal{S}$ there is no connection.

Note that in both cases the use of $\mathcal{S}$ was essential. If one drops the condition that left-hand sides of dependency pairs have to be instantiated to normal forms, then both $(28), (28), \dots$ and $(29), (30), (29), (30), \dots$ are infinite chains. And exactly this condition is exploited by $\mathcal{S}$.

However, there have been different definitions of the function *Cap* which are all syntactical, cf. [AG00, GTS05b, HM05]. Here, we now use the new approach where we define *Cap semantically* and then define a concept of an *estimated Cap function.* It turns out that all previous definitions of *Cap* indeed estimate our *Cap* function.

We obtain the following benefit from a semantic definition of the *Cap* function: We can easily reformulate all processors relying (possibly indirectly) on *Cap* in a way that these processors can be used for arbitrary estimations of *Cap*. Then we adapt the proof of these processors for an estimated *Cap*-function *once*. In contrast, if there is no concept of an estimated *Cap*-function then *every time* we change the definition of *Cap*, we have to check and possibly adapt the proof for *every* processor relying on *Cap*. Note that this usually involves a lot of work, since many processors rely (indirectly) on *Cap*. To be more precise we make use of *Cap* in the processors of Theorems 3.3 and 3.25, in 5 of the 10 processors of Chapter 4, in all processors presented in Chapter 5, and in half of the processors in Chapter 6.

Note that we allow *Cap* to be applied on *generalized TRSs*, i.e., TRSs where the rules are arbitrary pairs of terms which do not have to satisfy the variable condition. This will be required later in Section 3.4, which deals with alternative techniques to estimate the dependency graph.

Definition 3.7 (*Cap*-Function)**.** *The* *Cap*-function *is a mapping from a generalized TRS* $\mathcal{R}$*, two sets of terms* $\mathcal{Q}$ *and* $\mathcal{S}$*, and a term* t *to another term* $t' = Cap^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$*. The term* t' *is obtained by replacing all maximal subterms* $t|_p$ *of* t *by fresh variables whenever there is a substitution* σ *and a term* u *such that* $\mathcal{S}\sigma \subseteq NF(\mathcal{Q})$ *and* $t\sigma \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} \overset{\mathcal{Q}}{\to}_{\mathcal{R},p} u$.[9] *Moreover, whenever* $t|_p$ *is a variable and there is any reduction possible in* $t|_p\sigma$ *then we have to replace at least* $t|_p$ *by a fresh variable.*[10]

An estimated *Cap*-function *ECap is a function with the following property. Whenever Cap replaces a subterm at a position* p *by a fresh variable then there is a subterm at a higher position* $p' \leq p$ *which is replaced by a fresh variable using ECap.*

The essential property of an estimated *Cap*-function is that $ECap(t)$ contains the structure of the term $t\sigma$ after any number of reduction steps.

Lemma 3.8 (Properties of *Cap*)**.** *Let ECap be any estimation of the Cap function. If* $t\sigma \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} u$ *for some substitution* σ *such that* $\mathcal{S}\sigma \subseteq NF(\mathcal{Q})$ *then* $u = ECap^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)\mu$ *for some substitution* μ *which differs from* σ *only on the fresh variables that are introduced by ECap.*

Using an estimated *Cap* function we can now give a first estimation of the dependency graph.

Definition 3.9 (Estimation of the Dependency Graph)**.** *Let ECap be an estimated Cap function. Then the corresponding estimated* $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$*-dependency graph with* $\mathcal{P} = (N, E)$ *is defined as* (N, E') *where* $(s \to t, u \to v) \in E'$ *iff* $ECap^{\{s,u\}}_{\mathcal{R},\mathcal{Q}}(t)$ *and* u *are unifiable by an mgu* δ *such that* $s\delta$ *and* $u\delta$ *are in* $\mathcal{Q}$*-normal form.*

Theorem 3.10 (Soundness of the Dependency Graph Estimation)**.** *The estimated dependency graph of Definition 3.9 contains the dependency graph.*

Now that we know how to estimate the dependency graph by using *Cap* of course we also need an estimated *Cap*-function. The estimation in [AG00] replaces all subterms with defined root by a fresh variable.[11] However, subterms that occurred in the set $\mathcal{S}$ are not replaced, if one is in the innermost case. The more recent approach of [GTS05b] uses unification with left-hand sides instead of just looking at the root symbol. We generalize the approach of [GTS05b] for full- or innermost-rewriting to $\mathcal{Q}$-restricted rewriting. Basically, we take the estimation of [GTS05b] and integrate various checks on $\mathcal{Q}$-normal forms resulting in our new improved estimation *ICap*.

The main idea to compute $ICap^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$ is as follows. If t is a variable then it is replaced by a fresh one if it cannot be guaranteed that t is instantiated with a normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$.

---

[9] Throughout this thesis $\to_1\to_2$ denotes the composition of binary relations $\to_1$ and $\to_2$.

[10] It may be the case that we cannot directly rewrite at the position of the variable but only at a position deeper in the term. Consider $\mathcal{Q} = \{\mathsf{f}(\mathsf{a})\}, t = \mathsf{f}(x), \mathcal{S} = \{t\}, \mathcal{R} = \{\mathsf{a} \to \mathsf{b}\}$. Then clearly for $\sigma = \{x/\mathsf{g}(\mathsf{a})\}$ we obtain $t\sigma = \mathsf{f}(\mathsf{g}(\mathsf{a})) \overset{\mathcal{Q}}{\to}_{\mathcal{R}} \mathsf{f}(\mathsf{g}(\mathsf{b}))$ for the $\mathcal{Q}$-normal form $t\sigma$. But there is no possibility to rewrite $x\sigma$ at the root position as the only possible substitution $\sigma' = \{x/\mathsf{a}\}$ conflicts with the condition that $t\sigma'$ must be a $\mathcal{Q}$-normal form.

[11] In the termination case additionally the function *Ren* is applied, which replaces every occurrence of a variable by a fresh one. Here, the effect of *Ren* is already integrated in *Cap*.

Otherwise, if t is a function application one applies *ICap* on the arguments of t resulting in t'. In this way by Lemma 3.8 we know that t' has the structure of the term resulting from rewriting an instance of t below the root. Then t' is the result if one can guarantee that there is no reduction at the root position. However, a root reduction can only be performed if t' is unifiable with some left-hand side of a rule. If the mgu satisfies some normal-form conditions then *ICap* estimates that a reduction may be possible at the root level and returns a fresh variable. Otherwise, t' is returned. This idea is presented more formally in the following definition.

Definition 3.11 (*ICap*)**.** *Let $\mathcal{Q}$ and $\mathcal{S}$ be sets of terms, let $\mathcal{R}$ be a generalized TRS. We define the* improved estimated *Cap*-function *ICap as*

- $\mathit{ICap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(x) = x$ *if* $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$ *and x is a subterm of a term in $\mathcal{S}$.*
- $\mathit{ICap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(f(t_1,\dots,t_n)) = f(\mathit{ICap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1),\dots,\mathit{ICap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n))$ *if for every mgu δ of* $f(\mathit{ICap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1),\dots,\mathit{ICap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n))$ *and some left-hand side ℓ of a rule $\ell \to r \in \mathcal{R}$ there is some term in* $\mathcal{S}\delta \cup \{\ell\delta|_1,\dots,\ell\delta|_n\}$[12] *that is not in $\mathcal{Q}$-normal form.*
- $\mathit{ICap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$ *is a fresh variable, otherwise.*

Lemma 3.12. *ICap is an estimated Cap-function.*

Of course one can omit some checks on $\mathcal{Q}$-normal forms in Definition 3.11. Then one still has an estimated *Cap*-function as then subterms are replaced by fresh variables at even higher positions.

To see the difference between the estimation *ECap* of [AG00] which replaces every term with defined root by a fresh variable and the new estimation *ICap*, we consider the TRS $\mathcal{R}$ of Example 3.5, $\mathcal{Q} = \mathcal{S} = \varnothing$, and the term $t = \mathsf{isZero}(\mathsf{true})$. We obtain $\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) = x_{\mathsf{fresh}}$ as isZero is defined in $\mathcal{R}$. In contrast $\mathit{ICap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) = t$ since all isZero-rules require that the argument is $\mathsf{0}$ or $\mathsf{s}(x)$, but there is no rule for true as argument.

Nevertheless, especially for TRSs representing first-order functional programs we often get the same estimated graphs regardless of whether we use the estimated *Cap*-function of [AG00] or *ICap* of Definition 3.11. However, for TRSs representing higher-order functional programs, *ICap* produces much better estimations. This is demonstrated already in [GTS05b] and here we refer to Example 6.1 in Chapter 6 about handling TRSs that encode higher-order functions.

Example 3.13. We continue the termination proof of the TRS of Example 3.5.

The estimated dependency graph computed by Definition 3.9 using *ICap* just contains one edge more than the real dependency graph. The problem is that it cannot be detected that there is no edge from (27) to itself. We have to unify the terms $\mathit{ICap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(\mathsf{F}(x,x,x)) = \mathsf{F}(x_1,x_2,x_3)$ and $\mathsf{F}(\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{s}(\mathsf{0})),y)$ for the set $\mathcal{S} = \{\mathsf{F}(\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{s}(\mathsf{0})),x), \mathsf{F}(\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{s}(\mathsf{0})),y)\}$. These terms are obviously unifiable and as $\mathcal{Q} = \varnothing$ every term is in $\mathcal{Q}$-normal form. Thus, Definition 3.9 cannot detect that there is no edge from (27) to (27) in the dependency graph.

Hence, after applying Theorem 3.3 and Theorem 3.4 we partition the initial DP problem into the three new DP problems $\mathcal{D}_1 = (\{(19),(20),(21)\},\varnothing,\mathcal{R},\mathbf{m})$, $\mathcal{D}_2 = ((23) \leftrightarrow (25),\varnothing,\mathcal{R},\mathbf{m})$, and $\mathcal{D}_3 = (\{(27)\},\varnothing,\mathcal{R},\mathbf{m})$.

---

[12] We do not apply δ on the subterms of ℓ as ℓ may be a variable for a generalized TRS.

Note that in the innermost case where $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$ we would be able to drop the edge from (27) to (27). The reason is that then $\mathit{ICap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(\mathsf{F}(x,x,x))$ is the term $\mathsf{F}(x,x,x)$ which does not unify with $\mathsf{F}(\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{s}(\mathsf{0})),y)$.

In Example 3.13 we have seen a first reason that it is desirable to have a processor which changes from termination to innermost termination, as then the DP problem $\mathcal{D}_3$ can be deleted. This processor will be presented in the following section.

3.2. Switching to Innermost Termination

As indicated by Lemma 2.17, it is always advantageous to obtain a smaller set $\mathit{NF}(\mathcal{Q})$ which can be done by enlarging $\mathcal{Q}$. It is especially important to obtain a DP problem $(\mathcal{P},\mathcal{Q},\mathcal{R},f)$ with $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$ as there are many processors which produce better result for the innermost case. We have already seen this in the estimation of the dependency graph (Section 3.1), and we will see it again many times throughout this thesis. For example, there are less usable rules (Section 3.3) and the transformations of DPs are more often applicable (Chapter 5).

In the classical dependency pair approach, a switch from termination to innermost termination was only possible if the *whole* TRS belongs to a class where innermost termination implies termination. In particular, this holds for overlay systems (i.e., TRSs where no left-hand side unifies with a non-variable proper subterm of another left-hand side) which are locally confluent [Gra95]. So in particular, this includes non-overlapping TRSs.

Instead, the following processor only requires local confluence for the rules $\mathcal{R}$ of the current DP problem and no left-hand side of a rule may unify with a non-variable subterm of a left-hand side of a pair in $\mathcal{P}$. Note that often the TRS $\mathcal{R}$ is usually just a small subset of the original TRS due to previous simplifications by other processors, and $\mathcal{R}$ does not have to be an overlay system, but the rules may have arbitrary critical pairs. One only requires that $\mathcal{R}$ may not overlap with the pairs in $\mathcal{P}$. And again, after applying the processors in Section 3.1 each component $\mathcal{P}$ contains (often strictly) less pairs than the dependency pairs of the original TRS which is crucial for the running example, cf. Example 3.22. All this clearly extends the known classes where innermost termination implies termination. Finally, we show that our results also provide a new simple proof for the above result of [Gra95].

Theorem 3.14 (Processor to Switch to Innermost Termination). *The following processor is sound and complete. For a problem $(\mathcal{P},\mathcal{Q},\mathcal{R},f)$, Proc returns*

- $\{(\mathcal{P},\mathit{lhs}(\mathcal{R}),\mathcal{R},f)\}$, *if*
  - *for all $s \to t \in \mathcal{P}$, non-variable subterms of s do not unify with left-hand sides of rules from $\mathcal{R}$,*
  - *$\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ is locally confluent on the set of $\mathcal{Q}$-terminating terms w.r.t. $\mathcal{R}$,*
  - *$\mathit{NF}(\mathcal{R}) \subseteq \mathit{NF}(\mathcal{Q})$, and*
  - *$f = \mathbf{m}$*
- $\{(\mathcal{P},\mathcal{Q},\mathcal{R},f)\}$, *otherwise.*

The following examples show that each of the requirements is necessary for the soundness of Theorem 3.14. Here, we use the important property that in an innermost chain we always have to rewrite the instantiated right-hand sides to a normal form w.r.t. $\mathcal{R}$.

Example 3.15. If we do not require minimality then it may happen that there is no normal form that we can reach from the right-hand sides of $\mathcal{P}$. Consider $\mathcal{P} = \{\mathsf{F}(x) \to \mathsf{F}(\mathsf{a})\}$ together with $\mathcal{R} = \{\mathsf{a} \to \mathsf{a}\}$ and $\mathcal{Q} = \varnothing$. Here, there obviously is an infinite chain although it is not minimal. However, there is no infinite $(\mathcal{P}, \mathit{lhs}(\mathcal{R}), \mathcal{R})$-chain any more, as $\mathsf{F}(\mathsf{a})$ is not reducible to an $\mathcal{R}$-normal form.

Example 3.16. It might occur that rewriting right-hand sides can reach instances of left-hand sides of $\mathcal{P}$ but when rewriting further to a normal form one obtains no instance of a left-hand side of $\mathcal{P}$ any more. This is especially true if there are overlaps between $\mathcal{P}$ and $\mathcal{R}$ as then a reduction with $\mathcal{R}$ might destroy the redex w.r.t. $\mathcal{P}$. For $\mathcal{P} = \{\mathsf{F}(\mathsf{a}) \to \mathsf{F}(\mathsf{a})\}$, $\mathcal{R} = \{\mathsf{a} \to \mathsf{b}\}$, and $\mathcal{Q} = \varnothing$ there obviously is an infinite minimal chain. But if we change $\mathcal{Q}$ to be the set of left-hand sides of $\mathcal{R}$ then there is no infinite chain any more. As $\mathsf{F}(\mathsf{a})$ is not a normal form we have to reduce it to $\mathsf{F}(\mathsf{b})$ which cannot be matched by the left-hand side of the pair in $\mathcal{P}$ any more.

Example 3.17. A possible extension of the theorem would be to drop the requirement $\mathit{NF}(\mathcal{R}) \subseteq \mathit{NF}(\mathcal{Q})$ and then return as new second component either $\mathcal{Q} \cup \mathit{lhs}(\mathcal{R})$ or just $\mathit{lhs}(\mathcal{R})$.

In the former case we encounter the same kind of problem as in the previous example. Consider $\mathcal{P} = \{\mathsf{F}(x) \to \mathsf{F}(\mathsf{a})\}$, $\mathcal{R} = \{\mathsf{a} \to \mathsf{b}\}$, and $\mathcal{Q} = \{\mathsf{b}\}$. Then there is an infinite minimal chain but again we are forced to evaluate the right-hand side $\mathsf{F}(\mathsf{a})$ to $\mathsf{F}(\mathsf{b})$ which is not in normal form w.r.t. $\mathcal{Q} \cup \mathit{lhs}(\mathcal{R})$. Hence, returning $\{(\mathcal{P}, \mathcal{Q} \cup \mathit{lhs}(\mathcal{R}), \mathcal{R}, f)\}$ is unsound.

To obtain a counterexample for the latter case we keep the same $\mathcal{P}$ and $\mathcal{Q}$ and just replace $\mathcal{R}$ by $\{\mathsf{a} \to \mathsf{g}(\mathsf{b}), \mathsf{g}(x) \to \mathsf{g}(x)\}$. Then again there is the obvious chain for the original DP problem where we never perform a rewrite step with $\mathcal{R}$. Note that this chain really is a minimal chain as the infinite reduction of $\mathsf{a} \to_{\mathcal{R}} \mathsf{g}(\mathsf{b}) \to_{\mathcal{R}} \mathsf{g}(\mathsf{b}) \ldots$ is blocked by $\mathcal{Q}$. But when replacing $\mathcal{Q}$ by $\mathit{lhs}(\mathcal{R})$ the infinite reduction is not blocked any more and like in Example 3.15 we cannot reduce the right-hand side to a normal form any more.

Example 3.18. As a final counterexample for Theorem 3.14 we show that local confluence is needed. Here we can choose the example of Toyama [Toy87]. For $\mathcal{P} = \{\mathsf{F}(\mathsf{0}, \mathsf{1}, x) \to \mathsf{F}(x, x, x)\}$, $\mathcal{R} = \{\mathsf{c}(x, y) \to x, \mathsf{c}(x, y) \to y\}$, and $\mathcal{Q} = \varnothing$ there is an infinite chain if one instantiates x by $\mathsf{c}(\mathsf{0}, \mathsf{1})$:

$$\mathsf{F}(\mathsf{0},\mathsf{1},\mathsf{c}(\mathsf{0},\mathsf{1})) \to_{\mathcal{P}} \mathsf{F}(\mathsf{c}(\mathsf{0},\mathsf{1}),\mathsf{c}(\mathsf{0},\mathsf{1}),\mathsf{c}(\mathsf{0},\mathsf{1})) \to_{\mathcal{R}}^{*} \mathsf{F}(\mathsf{0},\mathsf{1},\mathsf{c}(\mathsf{0},\mathsf{1})) \to_{\mathcal{P}} \ldots$$

However, in the innermost case x must be instantiated by a normal form which cannot match both $\mathsf{0}$ and $\mathsf{1}$. Thus, there is no $(\mathcal{P}, \mathit{lhs}(\mathcal{R}), \mathcal{R})$-chain which clearly shows that local confluence is essential.

Before showing the advantages of Theorem 3.14 at the end of Section 3.2 by continuing the termination proof of Example 3.13, we will consider how this processor can be applied to a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$. All conditions but the second one can be easily checked. The only problem is to check local confluence of $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$. In practice, Theorem 3.14 is usually applied for $\mathcal{Q} = \varnothing$ (i.e., to switch from full to innermost termination). Then local confluence is equivalent to joinability of critical pairs. This property can easily be approximated by checking joinability in a given number of rewriting steps.

For $\mathcal{Q} \neq \varnothing$ we will at least give a sufficient criterion for local confluence of $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$. In this case it suffices if $\mathcal{R}$ has only trivial critical pairs. With such syntactic sufficient conditions for its applicability, Theorem 3.14 can easily be automated.

Lemma 3.19. *Let all critical pairs of $\mathcal{R}$ be trivial and let $\mathit{NF}(\mathcal{R}) \subseteq \mathit{NF}(\mathcal{Q})$. Then $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ is locally confluent on the set of $\mathcal{Q}$-terminating terms w.r.t. $\mathcal{R}$.*

The following two examples demonstrate that the restriction $\mathit{NF}(\mathcal{R}) \subseteq \mathit{NF}(\mathcal{Q})$ is essential and that indeed we can only guarantee local confluence on those terms that are terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$. However, we already have the requirement $\mathit{NF}(\mathcal{R}) \subseteq \mathit{NF}(\mathcal{Q})$ in Theorem 3.14 and we only need confluence on terminating terms. Thus, these two restriction are not very severe.

Example 3.20. This example demonstrates that local confluence can only be ensured for terminating terms. Let $\mathcal{R} = \{\mathsf{f}(x) \to \mathsf{c}, \mathsf{a} \to \mathsf{b}, \mathsf{b} \to \mathsf{b}\}$ and $\mathcal{Q} = \{\mathsf{b}\}$. Then $\mathcal{R}$ is even orthogonal and $\mathit{NF}(\mathcal{R}) \subseteq \mathit{NF}(\mathcal{Q})$. But for the term $t = \mathsf{f}(\mathsf{a})$ we have the two reductions $t \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \mathsf{c}$ and $t \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \mathsf{f}(\mathsf{b})$. And these two new terms are not joinable as the only way to reduce $\mathsf{f}(\mathsf{b})$ is to reduce it to itself, and c is a normal form w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$.

Example 3.21. If $\mathit{NF}(\mathcal{R})$ is not a subset of $\mathit{NF}(\mathcal{Q})$ then nothing at all can be said about local confluence of $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$. For $\mathcal{R} = \{\mathsf{f}(x) \to \mathsf{c}, \mathsf{a} \to \mathsf{b}\}$ and $\mathcal{Q} = \{\mathsf{b}\}$ the term $t = \mathsf{f}(\mathsf{a})$ is terminating and it has two different normal forms w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$: we obtain the reductions $t \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \mathsf{c}$ and $t \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \mathsf{f}(\mathsf{b})$.

As future work it remains to be investigated, whether one really needs trivial critical pairs of $\mathcal{R}$, or whether it is possible to develop some kind of joinability criterion.

We continue in the termination proof of the running example by using Theorem 3.14.

Example 3.22. We recapitulate some rules and pairs of Example 3.5.

$$\mathsf{plus}(\mathsf{plus}(x,y),z) \to \mathsf{plus}(x,\mathsf{plus}(y,z)) \tag{13}$$

$$\begin{aligned}
\mathsf{PLUS}(\mathsf{s}(x),y) &\to \mathsf{PLUS}(x,y) && (19)\\
\mathsf{PLUS}(\mathsf{plus}(x,y),z) &\to \mathsf{PLUS}(x,\mathsf{plus}(y,z)) && (20)\\
\mathsf{PLUS}(\mathsf{plus}(x,y),z) &\to \mathsf{PLUS}(y,z) && (21)\\
\mathsf{MUL}(x,y,z) &\to \mathsf{IF}(\mathsf{isZero}(x),x,y,z) && (23)\\
\mathsf{IF}(\mathsf{false},\mathsf{s}(x),y,z) &\to \mathsf{MUL}(x,y,\mathsf{plus}(y,z)) && (25)\\
\mathsf{F}(\mathsf{0},\mathsf{s}(\mathsf{0}),x) &\to \mathsf{F}(x,x,x) && (27)
\end{aligned}$$

At the end of Example 3.13 the DP problems $\mathcal{D}_1 = (\{(19),(20),(21)\},\varnothing,\mathcal{R},\mathbf{m})$, $\mathcal{D}_2 = ((23) \leftrightarrow (25),\varnothing,\mathcal{R},\mathbf{m})$, and $\mathcal{D}_3 = (\{(27)\},\varnothing,\mathcal{R},\mathbf{m})$ remained.

Note that $\mathcal{R}$ is locally confluent but it is not an overlay system as rule (13) overlaps with itself at a non-root position. Thus a global approach working on the whole TRS is not applicable. Moreover, even for the initial DP problem Theorem 3.14 is not applicable as here the dependency pairs (20) and (21) violate the condition that $\mathcal{P}$ and $\mathcal{R}$ do not overlap. However, after having split the initial DP problem by the processors of Theorem 3.3 and Theorem 3.4, the situation is different. For the two DP problems $\mathcal{D}_2$ and $\mathcal{D}_3$ the requirements of Theorem 3.14 are satisfied. Thus, we can replace these DP problems by $\mathcal{D}_4 = ((23) \leftrightarrow (25), \mathit{lhs}(\mathcal{R}), \mathcal{R}, \mathbf{m})$ and $\mathcal{D}_5 = (\{(27)\}, \mathit{lhs}(\mathcal{R}), \mathcal{R}, \mathbf{m})$.

As we have increased the $\mathcal{Q}$-component of these two DP problems it now makes sense to test whether we can delete further edges by Theorem 3.3. For $\mathcal{D}_4$ we cannot delete any edge, but note that due to the graph structure of the $\mathcal{P}$-component of DP problems we only have to examine the two existing edges instead of all four possible edges.

For $\mathcal{D}_5$ we delete the looping edge as we are now in the innermost case. That this is possible was already explained in more detail in Example 3.13. Hence, by Theorem 3.4 this DP problem is solved. This is especially important as processors based on well-founded orders – which we will see in Chapter 4 – can only be successful on $\mathcal{D}_5$ if the order is not $\mathcal{C}_\varepsilon$-compatible. (A $\mathcal{C}_\varepsilon$-compatible order $\underset{(\sim)}{\succ}$ must satisfy $\mathsf{c}(x, y) \underset{(\sim)}{\succ} x$ and $\mathsf{c}(x, y) \underset{(\sim)}{\succ} y$ for some fresh function symbol c.) The problem is that almost all orders used for automated termination proving are $\mathcal{C}_\varepsilon$-compatible.[13]

To summarize, we now remain with the new DP problems $\mathcal{D}_4$ and the unchanged DP problem $\mathcal{D}_1$.

Note that by Theorem 3.14, the observation that innermost termination implies termination for locally confluent overlay systems is obtained as a corollary. While the original proof for this important result of Gramlich [Gra95] is not at all trivial, the proof of Theorem 3.14 is quite simple. While there already exists another easy proof [Mid94], in this way we get an alternative simple proof for Gramlich's result.

Corollary 3.23 ([Gra95, Theorem 3.23]). *Let $\mathcal{R}$ be a locally confluent overlay system. If $\mathcal{R}$ is innermost terminating, then it is terminating.*

3.3. Usable Rules

If the $\mathcal{Q}$-restricted rewrite relation is contained in the innermost rewrite relation (i.e., if $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$), one can remove certain rules from the rewrite system $\mathcal{R}$. That this has no disadvantage is a result of Lemma 2.17, but on the contrary, often it is crucial for a successful termination proof. For example, when using the processors of Chapter 4 which are based on well-founded orders, then less rules imply less constraints to satisfy.

The essential idea to reduce the rewrite system $\mathcal{R}$ is that whenever there is a reduction $t\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} s\sigma$ for some $\mathcal{R}$-normal substitution σ then only the so-called *usable rules* can be used in this reduction. Hence, instead of all rules one just has to consider the subset of usable rules which is often much smaller.

Unfortunately, here we encounter the same scenario as for the *Cap*-function, cf. the discussion on page 21. There is no semantic definition of usable rules but one can find different syntactic definitions of usable rules [AG00, GTS05b, GTSF06] which produce incomparable sets of usable rules. Up to now with *every* improvement of usable rules one has to check for *every* processor relying on usable rules whether that processor is also correct with the improved version of usable rules.

For that reason we present a new *semantic* definition of usable rules. It turns out that all previous syntactic definitions of usable rules are just estimations of our semantic version of usable rules. Moreover, for all but one processor we have adapted the proof in a way that the theorem is valid for every estimation of usable rules without changing the theorem. Only in case of the rewriting processor in Theorem 5.10 we had to add an additional requirement. However, this new requirement is satisfied for all current estimations of usable rules. For details we refer to Section 5.3 about the rewriting processor.

---

[13] One exception is the class of polynomial orders with negative coefficients [HM07] which are able to solve the constraints of $\mathcal{D}_5$. However, even these orders are not applicable on $\mathcal{D}_3$ due to the constraints of the rules of $\mathcal{R}$. Hence, again the switch to innermost is required.

Originally, the usable rules were only introduced for innermost rewriting. For each term t the usable rules of t should contain all rules that can be used to rewrite an instance of t. However, as one is in the innermost case the variables of t may only be instantiated by normal forms. If $s \to t$ is a dependency pair then one can restrict the possible substitutions σ even further: one is only interested in those rules that can be used in a reduction of $t\sigma$ if $s\sigma$ is in normal form. We generalize this idea to $\mathcal{Q}$-restricted rewriting and we replace the single term s by a set $\mathcal{S}$ of terms that must all be instantiated to normal forms.

Definition 3.24 (Usable Rules)**.** *Let $\mathcal{Q}$ and $\mathcal{S}$ be sets of terms, let $\mathcal{R}$ be a TRS. We define the* usable rules $\mathcal{U}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$ *of a term t as the smallest subset of $\mathcal{R}$ which satisfies the following condition. Whenever there is a substitution σ and a term u such that $\mathcal{S}\sigma \subseteq NF(\mathcal{Q})$ and $t\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} \overset{\mathcal{Q}}{\to}_{\ell\to r} u$ for some rule $\ell \to r \in \mathcal{R}$ then $\ell \to r \in \mathcal{U}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$.*

The usable rules of a DP problem $(\mathcal{P},\mathcal{Q},\mathcal{R},f)$ are $\mathcal{U}(\mathcal{P},\mathcal{Q},\mathcal{R}) = \bigcup_{s\to t\in\mathcal{P}} \mathcal{U}^{\{s\}}_{\mathcal{R},\mathcal{Q}}(t)$.

A function $\mathcal{EU}$ estimates *the usable rules iff $\mathcal{U}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{EU}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{R}$ for all possible inputs $\mathcal{R}$, $\mathcal{Q}$, $\mathcal{S}$, and t. Again, $\mathcal{EU}(\mathcal{P},\mathcal{Q},\mathcal{R}) = \bigcup_{s\to t\in\mathcal{P}} \mathcal{EU}^{\{s\}}_{\mathcal{R},\mathcal{Q}}(t)$.*

Using this definition one immediately obtains the following theorem.

Theorem 3.25 (Processors Based on Usable Rules)**.** *Let $\mathcal{EU}$ be an estimation of usable rules. Then the following processor Proc is sound and complete. For a DP problem $(\mathcal{P},\mathcal{Q},\mathcal{R},f)$, Proc returns $\{(\mathcal{P},\mathcal{Q},\mathcal{EU}(\mathcal{P},\mathcal{Q},\mathcal{R}),f)\}$.*

Note that for DP problems $(\mathcal{P},\mathcal{Q},\mathcal{R},f)$ with $\mathcal{Q} = \varnothing$ virtually always all rules are usable as one can instantiate a variable with a term containing redexes for every rule. Hence, for termination there is hardly any difference between the usable rules and $\mathcal{R}$, unless $\mathcal{P}$ is right-ground. But for innermost termination one often obtains $\mathcal{U}(\mathcal{P},\mathcal{Q},\mathcal{R}) \subset \mathcal{R}$ and can successfully apply the above processor to simplify a DP problem.

The completeness of this processor is only due to our new notions of "$\mathcal{Q}$-restricted rewriting" and of "$(\mathcal{P},\mathcal{Q},\mathcal{R})$-chains", which use two separate components $\mathcal{Q}$ and $\mathcal{R}$. With $\mathcal{Q}$ we restrict potential redexes and $\mathcal{R}$ gives the possible rewrite steps. However, in the original dependency pair approach there is only one corresponding component: the TRS $\mathcal{R}$ determines both the redexes and the restriction on redexes. As a consequence, this processor would be incomplete in the original dependency pair approach, where one regarded innermost termination and "innermost chains". As an example let $\mathcal{P} = \{\mathsf{F}(\mathsf{a},x) \to \mathsf{F}(x,x)\}$ and $\mathcal{R} = \{\mathsf{f}(\mathsf{a},x) \to \mathsf{f}(x,x), \mathsf{a} \to \mathsf{b}\}$. Now there is no infinite innermost chain (i.e., no infinite $(\mathcal{P}, \mathit{lhs}(\mathcal{R}), \mathcal{R})$-chain), since the left-hand side of the dependency pair in $\mathcal{P}$ is not in $\mathit{lhs}(\mathcal{R})$-normal form. As there are no usable rules, this processor would replace $\mathcal{R}$ by the empty set. In the DP framework, one would obtain the DP problem $(\mathcal{P}, \mathit{lhs}(\mathcal{R}), \varnothing, f)$ which still has no infinite chain but in the classical dependency pair approach, the second component of this DP problem would be disregarded. Since there is an infinite (minimal) innermost chain of $\mathcal{P}$'s dependency pair if the underlying TRS is empty, then this processor would be incomplete.

Before presenting an estimation of usable rules let us first recapitulate the ideas of previous versions of usable rules. In [AG00, GTSF06] usable rules are essentially computed as follows. First, all f-rules are marked as usable whenever a symbol f occurs in rule of $\mathcal{P}$. Then, whenever a rule is usable and its right-hand side contains a symbol g then all g-rules are usable as well. Further improvements to compute usable rules use the fact that left-hand sides of $\mathcal{P}$ are instantiated to normal-forms [GTSF03], and unification is performed instead of just looking at the root symbols [GTS05b]. Now we combine all these improvements into one new definition and generalize from the innermost rewrite relation to $\mathcal{Q}$-restricted rewriting.

Definition 3.26 (Improved Estimated Usable Rules)**.** *Let $\mathcal{Q}$ and $\mathcal{S}$ be sets of terms and let $\mathcal{R}$ be a TRS. Let ECap be an estimated Cap-function. The* improved estimated usable rules *of a term t are defined as the smallest set $\mathcal{IU}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{R}$ such that*

(i) If $t = f(t_1, \ldots, t_n)$, $\ell = f(\ell_1, \ldots, \ell_n) \to r \in \mathcal{R}$, and if the terms $f(\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1), \ldots, \mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n))$ and ℓ are unifiable with the mgu δ such that all terms in $(\{\ell_1, \ldots, \ell_n\} \cup \mathcal{S})\delta$ are in $\mathcal{Q}$-normal form, then $\ell \to r \in \mathcal{IU}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$.

(ii) If $t = f(t_1, \ldots, t_n)$ then $\mathcal{IU}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_i) \subseteq \mathcal{IU}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$.

(iii) If $\ell = f(\ell_1, \ldots, \ell_n) \to r \in \mathcal{IU}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$ then $\mathcal{IU}^{\{\ell_1, \ldots, \ell_n\}}_{\mathcal{R},\mathcal{Q}}(r) \subseteq \mathcal{IU}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$.

(iv) If $t = x$ then $\mathcal{IU}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) = \mathcal{R}$ in the case that x is not a subterm of any term in $\mathcal{S}$ or $\mathit{NF}(\mathcal{Q}) \not\subseteq \mathit{NF}(\mathcal{R})$.

Note that in the computation of the improved usable rules for a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ case (iv) never applies if $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$: In that computation t always is a subterm of a right-hand side of $\mathcal{P} \cup \mathcal{R}$ and $\mathcal{S}$ contains the corresponding left-hand side or all direct subterms of the left-hand side. Then due to the variable conditions of the TRSs $\mathcal{P}$ and $\mathcal{R}$ each variable of t must be contained in $\mathcal{S}$. This shows that Definition 3.26 produces less rules than previous definitions of usable rules in [AG00, GTS05b, GTSF06]. Since the usable rules in [GTSF06] do not use unification and since in [GTS05b] there is no component $\mathcal{S}$, it is easy to construct examples where Definition 3.26 results in strictly less rules. Moreover, in [GTSF06] only the *left-hand sides of pairs* are integrated for normal form checks in the usable rules calculation. This is in contrast to Definition 3.26 which is the first estimation which also considers the *left-hand sides of rules* for normal form checks. That this can be useful is demonstrates in the following small example.

Example 3.27. Consider the TRS $\mathcal{R}$ with the following rules.

$$\begin{array}{rcl}
\mathsf{s}(\mathsf{p}(x)) & \to & x \\
\mathsf{p}(\mathsf{s}(x)) & \to & x \\
\mathsf{isPositive}(\mathsf{s}(x)) & \to & \mathsf{true} \\
\mathsf{isPositive}(\mathsf{0}) & \to & \mathsf{false} \\
\mathsf{isPositive}(\mathsf{p}(x)) & \to & \mathsf{isPositive}(x) \\
\mathsf{f}(\mathsf{p}(x), \mathsf{true}) & \to & \mathsf{f}(\mathsf{p}(x), \mathsf{isPositive}(x))
\end{array}$$

To prove innermost termination we get a DP problem where $\mathcal{P}$ consists of the following dependency pair.

$$\mathsf{F}(\mathsf{p}(x), \mathsf{true}) \to \mathsf{F}(\mathsf{p}(x), \mathsf{isPositive}(x))$$

Since the recursive $\mathsf{isPositive}$-rule is usable, one has to add $\mathcal{IU}^{\{\mathsf{p}(x)\}}_{\mathcal{R},\mathcal{Q}}(\mathsf{isPositive}(x))$ to the usable rules. Then, without the normal form check with the argument $\mathsf{p}(x)$ of the left-hand side, all $\mathsf{isPositive}$-rules are usable. However, if one performs the normal form check then the rule $\mathsf{isPositive}(\mathsf{s}(x)) \to \mathsf{true}$ is not usable.

Note that the TRS is not terminating. Therefore, one cannot prove innermost termination if one does not consider the normal form checks. This shows that the integration of additional normal form checks in the estimation of usable rules is sometimes essential. Here, after having deleted the unusable rule $\mathsf{isPositive}(\mathsf{s}(x)) \to \mathsf{true}$ one can indeed easily prove innermost termination using the processors of the next chapter.

The following theorem provides the essential property of $\mathcal{IU}$ that it is an estimation of $\mathcal{U}$.

Theorem 3.28 (Soundness of the Improved Estimated Usable Rules)**.** *The function $\mathcal{IU}$ to compute the improved estimated usable rules estimates $\mathcal{U}$.*

It is easy to see that the improved estimation of usable rules has a severe drawback with condition (iv): if $NF(\mathcal{Q}) \not\subseteq NF(\mathcal{R})$ then nearly always we obtain the situation that there is no difference between usable rules and the whole TRS unless $\mathcal{P}$ and all rules that are usable w.r.t. conditions (i)-(iii) are right-ground. Therefore, it might be tempting to delete condition (iv) in Definition 3.26. Unfortunately, the following example of [Toy87] shows that this is not possible.

Example 3.29. Let $\mathcal{P} = \{\mathsf{F}(0,1,x) \to \mathsf{F}(x,x,x)\}$, let $\mathcal{Q} = \varnothing$, and let $\mathcal{R} = \{\mathsf{c}(x,y) \to x, \mathsf{c}(x,y) \to y\}$. As already seen in Example 3.18, there is an infinite chain if one instantiates the variable x in $\mathcal{P}$ by $\mathsf{c}(0,1)$. According to the conditions (i)-(iii) in Definition 3.26 there are no usable rules for this DP problem. If we now drop condition (iv), the usable rules processor of Theorem 3.25 would result in the DP problem $(\mathcal{P}, \varnothing, \varnothing, f)$ which is finite. Thus, condition (iv) is essential for the correctness of $\mathcal{IU}$.

Nevertheless, in Section 4 we will present new methods that can drop condition (iv) even if one is not in the innermost case. Using these methods diminishes the difference between proving termination and innermost termination considerably.

We now show how the DP problems in our running example can be simplified by the usable rules processor of Theorem 3.25.

Example 3.30. We recapitulate the following pairs.

$$\mathsf{PLUS}(\mathsf{s}(x), y) \to \mathsf{PLUS}(x, y) \tag{19}$$
$$\mathsf{MUL}(x, y, z) \to \mathsf{IF}(\mathsf{isZero}(x), x, y, z) \tag{23}$$
$$\mathsf{IF}(\mathsf{false}, \mathsf{s}(x), y, z) \to \mathsf{MUL}(x, y, \mathsf{plus}(y, z)) \tag{25}$$

At the end of Example 3.22 we had to handle the two DP problems $\mathcal{D}_1$ and $\mathcal{D}_4$. We first deal with the DP problem $\mathcal{D}_1 = (\{(19), (20), (21)\}, \varnothing, \mathcal{R}, \mathbf{m})$ containing the dependency pairs for the plus-rules. As $NF(\varnothing) \not\subseteq NF(\mathcal{R})$ and as there are variables in the right-hand side $\mathsf{PLUS}(x, y)$ of (19), every rule is usable for this DP problem. Hence, an application of Theorem 3.25 cannot simplify $\mathcal{D}_1$.

This is in contrast to the DP problem $\mathcal{D}_4 = ((23) \leftrightarrow (25), \mathit{lhs}(\mathcal{R}), \mathcal{R}, \mathbf{m})$. Here, the subterms $\mathsf{isZero}(x)$ and $\mathsf{plus}(y, z)$ clearly unify with all isZero-rules, resp. plus-rules of $\mathcal{R}$. As $NF(\mathit{lhs}(\mathcal{R})) \subseteq NF(\mathcal{R})$ we do not have to consider case (iv) in Definition 3.26 and thus the improved estimated usable rules of this DP problem are $\mathcal{R}' = \{(9), (10), (11), (12), (13)\}$. It turns out that for this DP problem, $\mathcal{IU}$ even computes the usable rules exactly. Thus, when applying Theorem 3.25 we delete all remaining rules for computing multiplication and for f. As a result we obtain the simplified DP problem $\mathcal{D}_6 = ((23) \leftrightarrow (25), \mathit{lhs}(\mathcal{R}), \mathcal{R}', \mathbf{m})$.

Here, we see a first example where from a termination proof of a TRS we result in a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ where $\mathit{NF}(\mathcal{Q}) \subset \mathit{NF}(\mathcal{R})$. This indicates that the generalizations from innermost termination techniques to techniques handling the case $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$ are strongly required.

Note that in the previous example it would be sufficient to determine the estimated usable rules by just looking at the root symbols. As for the improved *Cap*-estimation, we refer to the TRSs in Chapter 6 like Example 6.1. There, it is clearly demonstrated why one needs the more powerful estimation of usable rules which is based on unification.

Before we can solve the remaining DP problems of Example 3.30 with the help of well-founded orders in Chapter 4, in the following section we will see that usable rules can also be used to estimate the dependency graph.

3.4. Star-Estimation of the Dependency Graph

To estimate the dependency graph in Section 3.1 we analyzed $t\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} u\sigma$ by approximating what $t\sigma$ will look like after some reduction steps. An alternative approach is taken in the graph approximation EDG* [Mid02]. There, one reverses the rewrite direction, i.e., one looks at the reversed TRS $\mathcal{R}^{-1} = \{r \to \ell \mid \ell \to r \in \mathcal{R}\}$ and the reversed reduction $u\sigma \to^{*}_{\mathcal{R}^{-1}} t\sigma$. Now, the idea is again to look at what $u\sigma$ will look like after some reduction steps with $\mathcal{R}^{-1}$. Of course, to this end we can again use *Cap*. Note, that $\mathcal{R}^{-1}$ often is no TRS but only a generalized TRS which may violate the variable condition. This is the reason why we defined *Cap* also for generalized TRSs.

So the basic idea of the EDG*-estimation is to note that $s \to t$ and $u \to v$ cannot form a chain if $\mathit{Cap}^{\varnothing}_{\mathcal{R}^{-1},\varnothing}(u)$ and t are not unifiable.

The idea of EDG* is then further improved for the innermost case resulting in the graph approximation EIDG* [HM05]. For EIDG*, instead of taking the whole TRS $\mathcal{R}$, one just considers the usable rules $\mathcal{R}'$. To be more precise, instead of checking whether $\mathit{Cap}^{\varnothing}_{\mathcal{R}^{-1},\varnothing}(u)$ and t are unifiable, one checks whether $\mathit{Cap}^{\varnothing}_{\mathcal{R}'^{-1},\varnothing}(u)$ and t are unifiable. Here, $\mathcal{R}'$ is the set of usable rules of t.

Now we use the ideas of EDG* and EIDG* to present a new graph estimation for arbitrary estimations of *Cap*, for arbitrary estimations of $\mathcal{U}$, and for $\mathcal{Q}$-restricted rewriting instead of full- or innermost-rewriting. As our notion of usable rules is independent of the rewrite relation we do not even need two different versions.

Definition 3.31 (Star-Estimation of the Dependency Graph)**.** *Let ECap be an estimated Cap-function, let $\mathcal{EU}$ estimate $\mathcal{U}$. Then the corresponding estimated star-$(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-dependency graph with $\mathcal{P} = (N, E)$ is defined as (N, E') where $(s \to t, u \to v) \in E'$ iff $\mathit{ECap}^{\varnothing}_{\mathcal{R}',\varnothing}(u)$ and t are unifiable by an mgu δ such that $s\delta$ and $u\delta$ are in $\mathcal{Q}$-normal form. Here, $\mathcal{R}'$ are the reversed usable rules of t, i.e., $\mathcal{R}' = (\mathcal{EU}^{\{s,u\}}_{\mathcal{R},\mathcal{Q}}(t))^{-1}$.*

Theorem 3.32 (Soundness of the Star-Estimation of the Dependency Graph)**.** *The estimated star-$(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-dependency graph of Definition 3.31 contains the $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-dependency graph.*

Of course, by using the estimations given in Definitions 3.11 and 3.26, the star-estimation of the dependency graph can easily be computed.

We consider the following example which demonstrates the difference between the dependency graph estimations of Definitions 3.9 and 3.31.

Example 3.33. Let $\mathcal{R}$ be the TRS of Example 3.5 and let $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$. For $\mathcal{P}$ we take the following pairs:

$$\mathsf{G}(\mathsf{a}, \mathsf{0}, x) \rightarrow \mathsf{G}(\mathsf{a}, \mathsf{isZero}(x), x) \tag{31}$$

$$\mathsf{G}(\mathsf{b}, \mathsf{0}, x) \rightarrow \mathsf{G}(\mathsf{b}, \mathsf{a}, \mathsf{plus}(x, x)) \tag{32}$$

We start with deleting edges due to Definition 3.31. For (31) the usable rules are the two rules (9) and (10) for isZero. Hence, the reversed usable rules for this pair are $\mathcal{R}' = \{\mathsf{true} \rightarrow \mathsf{isZero}(\mathsf{0}), \mathsf{false} \rightarrow \mathsf{isZero}(\mathsf{s}(x))\}$. Now, we compute $\mathit{ICap}^{\varnothing}_{\mathcal{R}',\varnothing}(\mathsf{G}(\mathsf{a}, \mathsf{0}, x)) = \mathsf{G}(\mathsf{a}, \mathsf{0}, y)$. As this term is not unifiable with any right-hand side of $\mathcal{P}$, both edges starting in (31) are deleted. The intuitive reason for this deletion is the fact that with Definition 3.31 we can detect that no instance of $\mathsf{isZero}(x)$ can be rewritten to $\mathsf{0}$, as the only possible results of isZero are true and false.

However, this reasoning is not possible with Definition 3.9. In that case we would have detected that an instance $\mathsf{isZero}(x)$ can be reduced and therefore the term $\mathsf{isZero}(x)$ is replaced by a fresh variable. And after this replacement the new term is unifiable with the left-hand side of (31) and thus, due to Definition 3.9 we cannot delete the looping edge from (31) to itself. This clearly shows that there are instances of DP problems where Definition 3.31 improves upon Definition 3.9.

But if we look at the edges starting from (32) we get the complementary result that sometimes Definition 3.9 is better. When applying *ICap* on the right-hand side of (32) we obtain $\mathsf{G}(\mathsf{b}, \mathsf{a}, y)$. As this term is not unifiable with any left-hand side of $\mathcal{P}$ we are able to delete both edges starting in (32) due to Definition 3.9.

This is in contrast to Definition 3.31. The usable rules for the pair (32) contain the collapsing plus-rule (11). This is especially bad, as one can see that whenever the usable rules that are computed for a connection between two pairs contain a collapsing rule, then Definition 3.31 will never be able to delete this edge: if a collapsing rule is usable then reversing these rules results in a generalized TRS $\mathcal{R}''$ containing a rule where the left-hand side is a variable. Obviously, every term can be rewritten by $\mathcal{R}''$. Thus, the term $\mathit{Cap}^{\varnothing}_{\mathcal{R}'',\varnothing}(t)$ is a fresh variable for every term t. And as a fresh variable is unifiable with every right-hand side of $\mathcal{P}$ we cannot delete a single edge starting in (32) due to Definition 3.31.[14] This shows that Definition 3.9 is sometimes a better estimation than Definition 3.31 which finally shows that both estimations are incomparable.

Note that to delete the edges starting in (31) it is essential that usable rules are computed for every right-hand side of $\mathcal{P}$ separately, instead of computing them for the whole DP problem as in Theorem 3.25. Otherwise, the TRS $\mathcal{R}'$ would include the critical generalized rule $y \rightarrow \mathsf{plus}(\mathsf{0}, y)$ and no edge could have been deleted by the star-estimation of the dependency graph.

As both graph estimations are incomparable we get the best result when applying both estimations. If one only uses Theorem 3.3 with one estimation then it is not possible to solve the given DP problem with the processors presented in this chapter. But we can use the dependency graph processor of Theorem 3.3 with both estimations to delete all edges. Then the resulting problem can be solved by Theorem 3.4.

---

[14] For a complete proof that Definition 3.31 cannot delete an edge in case of a collapsing usable rule we also have to look at the normal form conditions. If we are checking the connection between $s \rightarrow t$ and $u \rightarrow v$ then we unify a fresh variable with u by the mgu δ. Thus, $u\delta = u$ and $s\delta = s$. If one of these terms is not in $\mathcal{Q}$-normal form then one can delete the edge. However, then $\mathcal{U}^{\{s,u\}}_{\mathcal{R},\mathcal{Q}}(t)$ is the empty set in contradiction to the requirement that there is collapsing usable rule.

3.5. Reducing $\mathcal{Q}$

In this section we introduce a novel processor which tries to reduce the set $\mathcal{Q}$. The obvious question is why one should want such a processor since it never harms to add terms to $\mathcal{Q}$ (Lemma 2.17). However, the idea is to only remove those terms in $\mathcal{Q}$ that cannot block reductions any more: If a term in $\mathcal{Q}$ contains symbols which do not occur in $\mathcal{P} \cup \mathcal{R}$ any more, as one might have deleted the corresponding pairs and rules, then one can safely remove that term from $\mathcal{Q}$ without losing completeness.

Consider the remaining DP problem $\mathcal{D}_6 = (\{(23), (25)\}, \mathit{lhs}(\mathcal{R}), \mathcal{R}', \mathbf{m})$ for the recursive calls of mul and if in Example 3.30 where $\mathcal{R}'$ is the set of plus- and isZero-rules, i.e., $\mathcal{R}' = \{(9) - (13)\}$. Here, the set $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$ still contains symbols like times and mul which neither occur in the pairs nor in the rules. Thus, removing all the terms containing these symbols from $\mathcal{Q}$ results in the new set $\mathcal{Q}' = \mathit{lhs}(\mathcal{R}')$.

There are two advantages one obtains from reducing $\mathcal{Q}$ in this way. The first one is efficiency. As we have seen in the previous sections, for the estimation of the dependency graph and for the usable rules one has to perform many tests whether some term is in $\mathcal{Q}$-normal form. And of course, these tests become cheaper for smaller sets $\mathcal{Q}$. The other advantage is that certain processors are more powerful when $\mathcal{Q}$ is reduced. For example, proving non-termination is easier for small sets $\mathcal{Q}$, cf. Chapter 8, and the $\mathcal{A}$-transformation and semantic labeling are more powerful if $\mathcal{Q}$ does not contain terms of a certain form, cf. Theorems 6.8 and 6.17 (C) in Chapter 6, and Theorems 7.15, 7.19, and 7.23 in Chapter 7. Moreover, some processors which have only been developed for innermost rewriting may become applicable after the $\mathcal{Q}$-reduction processors have been applied. Examples include the processors of [GTSS07] to prove termination by showing that arguments are increased until they finally reach a bound.

However, it turns out that in general reducing $\mathcal{Q}$ forces us to drop minimality to achieve soundness.

Theorem 3.34 ($\mathcal{Q}$-Reduction Processor)**.** *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ be a DP problem, let $\mathcal{F}$ be the set of symbols occurring in $\mathcal{P} \cup \mathcal{R}$. Then the following processor is sound and complete.*

$$\mathit{Proc}((\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)) = \{(\mathcal{P}, \mathcal{Q} \cap \mathcal{T}(\mathcal{F}, \mathcal{V}), \mathcal{R}, \mathbf{a})\}$$

The following example shows that minimality cannot be carried over, even if $\mathcal{Q} \supseteq \mathit{lhs}(\mathcal{R})$.

Example 3.35. Let $\mathcal{P} = \{\mathsf{F}(x) \to \mathsf{F}(\mathsf{f}(x))\}$, let $\mathcal{Q} = \mathit{lhs}(\mathcal{R}) \cup \{\mathsf{h}(\mathsf{a})\}$, and let $\mathcal{R}$ consist of the following rules.

$$\begin{array}{rcl} \mathsf{f}(x) & \to & x \\ \mathsf{f}(x) & \to & \mathsf{g}(\mathsf{h}(x)) \\ \mathsf{g}(\mathsf{h}(x)) & \to & \mathsf{g}(\mathsf{h}(x)) \end{array}$$

Then there is an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain.

$$\mathsf{F}(\mathsf{a}) \to \mathsf{F}(\mathsf{f}(\mathsf{a})) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \mathsf{F}(\mathsf{a}) \to \dots$$

The minimality is due to the fact that the evaluation $\mathsf{f}(\mathsf{a}) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \mathsf{g}(\mathsf{h}(\mathsf{a}))$ is blocked as now $\mathsf{h}(\mathsf{a})$ is not in $\mathcal{Q}$-normal form. However, applying Theorem 3.34 results in $\mathcal{Q}' = \mathit{lhs}(\mathcal{R})$ as the term $\mathsf{h}(\mathsf{a})$ contains the symbols a which is not present in $\mathcal{P} \cup \mathcal{R}$. Then the above chain still is an infinite $(\mathcal{P}, \mathcal{Q}', \mathcal{R})$-chain, but there does not exist an infinite minimal $(\mathcal{P}, \mathcal{Q}', \mathcal{R})$-chain as no instance of $\mathsf{F}(\mathsf{f}(x))$ is terminating w.r.t. $\xrightarrow{\mathcal{Q}'}_{\mathcal{R}}$.

The problem in the previous example is that terms of $\mathcal{Q}$ have been deleted which contained symbols which are not in $\mathcal{F}$ and which occurred *below the root*. It turns out that if we are more restrictive than innermost rewriting, and if we delete only those terms in $\mathcal{Q}$ where the *root-symbol* is not contained in $\mathcal{F}$, then we can preserve minimality.

Theorem 3.36 ($\mathcal{Q}$-Reduction Processor). *Let* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ *be a DP problem with* $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$, *let* $\mathcal{F}$ *be the set of symbols occurring in* $\mathcal{P} \cup \mathcal{R}$. *Then the following processor is sound and complete.*

$$\mathit{Proc}((\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)) = \{(\mathcal{P}, \{q \in \mathcal{Q} \mid \mathit{root}(q) \in \mathcal{F}\}, \mathcal{R}, f)\}$$

The following example shows that in the previous theorem we really need the condition $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$, even if we only delete those terms from $\mathcal{Q}$ which have a root that is not contained in $\mathcal{F}$.

Example 3.37. Let $\mathcal{P}$ consist of the single rule

$$\begin{array}{ll} \mathsf{F}(x_0, x_1, x_2, y_0, y_0, y_1, y_1, y_2, y_2, z) & \rightarrow \\ \mathsf{F}(x_0, x_1, x_2, x_0, x_1, x_1, x_2, x_2, x_0, \mathsf{h}(x_0, x_1, x_2)) & \end{array}$$

let $\mathcal{Q} = \{\mathsf{g}(\mathsf{c})\}$, and let $\mathcal{R}$ be the set of the following rules.

$$\begin{array}{rcl} \mathsf{a}_0 & \rightarrow & \mathsf{b}_{20} \\ \mathsf{a}_0 & \rightarrow & \mathsf{b}_{01} \\ \mathsf{a}_1 & \rightarrow & \mathsf{b}_{01} \\ \mathsf{a}_1 & \rightarrow & \mathsf{b}_{12} \\ \mathsf{a}_2 & \rightarrow & \mathsf{b}_{12} \\ \mathsf{a}_2 & \rightarrow & \mathsf{b}_{20} \\ \mathsf{a}_0 & \rightarrow & \mathsf{c} \\ \mathsf{a}_1 & \rightarrow & \mathsf{c} \\ \mathsf{a}_2 & \rightarrow & \mathsf{c} \\ \mathsf{h}(x, x, x) & \rightarrow & \mathsf{h}(x, x, x) \end{array}$$

Then there is an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$ chain if one chooses σ with $\sigma(x_i) = \mathsf{g}(\mathsf{a}_i)$, $\sigma(y_i) = \mathsf{g}(\mathsf{b}_{i,i+1 \bmod 3})$, and $\sigma(z) = \mathsf{h}(\mathsf{g}(\mathsf{a}_0), \mathsf{g}(\mathsf{a}_1), \mathsf{g}(\mathsf{a}_2))$. The reason is that $\sigma(x_i)$ and $\sigma(x_{i+1 \bmod 3})$ can be joined into $\sigma(y_i)$, but to join all $\sigma(x_i)$ one has to reduce these terms to $\mathsf{g}(\mathsf{c})$. But then the possible infinite reduction of the term $\mathsf{h}(\mathsf{g}(\mathsf{c}), \mathsf{g}(\mathsf{c}), \mathsf{g}(\mathsf{c}))$ is blocked by $\mathcal{Q}$.

However, note that $\mathit{root}(\mathsf{g}(\mathsf{c})) = \mathsf{g} \notin \mathcal{F}$ and hence, we may delete the only term in $\mathcal{Q}$. As one can prove that there is no infinite minimal $(\mathcal{P}, \varnothing, \mathcal{R})$-chain, this example shows that one cannot drop the condition $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$ from Theorem 3.36.

Note that Theorem 3.34 and Theorem 3.36 do not subsume each other, as Theorem 3.34 can delete more terms from $\mathcal{Q}$ and does not require $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$, but on the other hand only with Theorem 3.36 it is possible to preserve minimality. We conclude this section by continuing our running example.

Example 3.38. As illustrated in the introduction of this section, we are now able to simplify the remaining DP problem $\mathcal{D}_6 = (\{(23), (25)\}, \mathit{lhs}(\mathcal{R}), \mathcal{R}', \mathbf{m})$ into the DP problem $\mathcal{D}_7 = (\{(23), (25)\}, \mathit{lhs}(\mathcal{R}'), \mathcal{R}', \mathbf{m})$ where $\mathcal{R}' = \{(9) - (13)\}$.

Note that at this point we have transformed back a DP problem which had a stronger strategy restriction than innermost into one which has again exactly innermost evaluation strategy. Thus, we can again apply every technique which is only developed for innermost rewriting.

Summary of Chapter 3

The processors of this chapter help us to prove termination of different algorithms independently (Theorems 3.3 and 3.4). Moreover, with Theorem 3.14 we have shown a processor that can be used to switch from full termination to innermost termination. And by Theorem 3.25 there is a way to delete rules which are not usable. Finally with Theorem 3.36 we have a processor which can remove superfluous terms from $\mathcal{Q}$. Since all of these processors can easily be applied – there is no major search problem involved – and since these processors never complicate DP problems, one should always simplify a given DP problem with these processors, before trying more expensive processors of upcoming chapters.

The techniques behind most of these processors are already known in the literature. The only completely new techniques of this chapter are the processors to reduce $\mathcal{Q}$, and the important processor to switch to innermost termination (a simplified version of the latter processor was published by us in [GTS05a] and extends the result of [Gra95]). Nevertheless, this chapter contains some more contributions: For the first time, all the processors are now generalized to $\mathcal{Q}$-restricted rewriting. And even more importantly, we have unified different *syntactic* notions by giving them semantics, where we have developed a *semantic* concept of usable rules and of *Cap* instead of using one of at least three available syntactic variants. Moreover, we have presented syntactic estimations of usable rules, of *Cap*, and of the dependency graph, which improve upon nearly all previous syntactic versions [AG00, HM05, Mid02] (basic version of our improved estimations have already been published by us in [GTS05b, GTSF06]). Currently, the only estimations that are not encompassed by our work are the graph estimations of [Mid01] which are based on tree automata and are incomparable in power.

Nevertheless, we still see at least two possibilities to extend the processors and estimations of this chapter. First, up to now we can only guarantee local confluence of $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ for a non-empty $\mathcal{Q}$ if all critical pairs are trivial, whereas for $\mathcal{Q} = \varnothing$ it suffices if all critical pairs are joinable. Here, it should be possible to develop kind of joinability criterion to increase the power of the processor to switch to innermost.

And second, we strongly conjecture that it should be possible to only consider the usable rules of a subterm when computing the star-estimation of the dependency graph. We illustrate our idea with the pair $\mathsf{F}(\mathsf{true}, x) \to \mathsf{F}(\mathsf{test}(x), \mathsf{id}(x))$ and the TRS $\mathcal{R} = \{\mathsf{test}(x) \to \mathsf{false}, \mathsf{id}(x) \to x\}$. Even in the innermost case our current estimations cannot detect that the pair is not connected to itself. The problem in the star-estimation is that *Cap* will replace every term by a fresh variable due to the reversed usable id-rule. However, only the test-rule can be applied on the first argument of F. Therefore, one should only take the reversed usable rule $\mathsf{false} \to \mathsf{test}(x)$ of the argument $\mathsf{test}(x)$ as generalized TRS when applying *Cap* on the argument true. In this way one can detect that $\mathsf{test}(x)$ cannot be reduced to true and one can delete the edge.

Although the processors of this chapter simplify DP problems considerably, we still need more processors. The reason is that all processors of this chapter can only delete rules that are never used, and they can delete pairs and edges that are used at most once in a chain. However, if we have recursive algorithms like `plus` and `mult`, then the number of recursive calls depends on the starting term and can be arbitrarily high, although not infinite. To prove this we can use well-founded orders to show that every recursion must finally end. This is investigated in more detail in the next chapter.

4. Processors Based on Orders

Classical techniques for automated termination proofs try to find a *reduction order* $\succ$, i.e., an order which is well-founded, monotonic, and stable (closed under contexts and substitutions), such that $\ell \succ r$ holds for all rules $\ell \to r$ of the TRS. In practice, most orders are *simplification orders*, where a term is always greater than its proper subterms [Der87, Ste95]. Examples for such orders are the *lexicographic* or *recursive path order* [Der87, KL80], the *Knuth-Bendix order* [KB70], and many polynomial orders [Lan79]. However, the power of this approach is limited, since termination of many important TRSs cannot be proved with simplification orders. For instance, simplification orders fail on the TRS of Example 3.1, since the left-hand side of rule (18) is embedded in its right-hand side if x is instantiated with $\mathsf{s}(\mathsf{s}(\mathsf{0}))$.

The dependency pair approach was introduced to overcome the limitations of classical simplification orders. For any TRS, it generates a set of inequality constraints and if there exists a well-founded order satisfying the constraints, then termination is proved. Since the well-founded orders need not be monotonic, one can compose more powerful orders from so-called *argument filters* and classical orders. Hence, one can use existing techniques to search for suitable orders and it turns out that in this way, classical simplification orders can prove termination of numerous TRSs where they would have failed otherwise.

In this chapter we formalize this idea in the context of the DP framework. To be more precise, we present, combine, and improve recent techniques of [GTS05a, GTS05b, GTSF06, HM05, HM07, TGS04, Urb01] to reduce the number of constraints that have to be satisfied by a well-founded order. Moreover, in all the corresponding proofs we replace the various syntactic definitions of *Cap* by our semantic definition.

The structure of this chapter is as follows. In Section 4.1 we show the initial use of well-founded orders in combination with argument filters as in [AG00]. Then, in Section 4.2 techniques are presented which considerably reduce the set of generated constraints. Moreover, a processor is developed which can remove unneeded rules from the TRS $\mathcal{R}$ of a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$. A method to remove further rules of $\mathcal{R}$ is described in Section 4.3. How to reduce the set of constraints even more by considering argument filters is the topic of Sections 4.4 and 4.5 which is especially useful for proving termination of TRSs using an accumulator. Finally, in Section 4.6 we will adapt the *subterm criterion* such that it can be applied on arbitrary DP problems. To this end, we will also develop a new processor to delete edges from the pair-graph which are connected in the dependency graph of Definition 3.2.

4.1. Reduction Pairs

To remove pairs from $\mathcal{P}$ in a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, one can generate constraints which should be satisfied by a *reduction pair* [KNT99] $(\succsim, \succ)$ where $\succsim$ is reflexive, transitive, monotonic, and stable and $\succ$ is a stable well-founded order compatible with $\succsim$, i.e.,

$\succsim \circ \succ \circ \succsim \; \subseteq \; \succ$.[15] But $\succ$ does not have to be monotonic. To simplify a DP problem, the constraints require that at least one pair in $\mathcal{P}$ is strictly decreasing (w.r.t. $\succ$) and all remaining pairs in $\mathcal{P}$ and all rules in $\mathcal{R}$ are weakly decreasing (w.r.t. $\succsim$). Requiring $\ell \succsim r$ for all rules $\ell \to r \in \mathcal{R}$ ensures that in chains $s_1 \to t_1, s_2 \to t_2, s_3 \to t_3, \ldots$ with $t_i\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} s_{i+1}\sigma$, we have $t_i\sigma \succsim s_{i+1}\sigma$. Hence, the existence of such a reduction pair implies that there is no chain which contains the strictly decreasing pairs of $\mathcal{P}$ infinitely often. Thus, all of these pairs can be deleted from $\mathcal{P}$.

To generate reduction pairs $(\succsim, \succ)$ automatically, one often uses classical (monotonic) simplification orders. However, $\succ$ does not have to be monotonic. To benefit from this possibility and to build non-monotonic orders from simplification orders, one may preprocess the constraints first and delete certain function symbols and arguments by an argument filter π. We use the notation of [KNT99].

Definition 4.1 (Argument Filter [AG00]). *Let $\mathcal{F}$ be a signature. An* argument filter π *is a mapping that assigns to every function symbol of $f \in \mathcal{F}$ with arity n either a number between 1 and n or a list of numbers from 1 to n in ascending order. An argument filter is lifted to a function from terms of $\mathcal{T}(\mathcal{F}, \mathcal{V})$ to terms as follows.*

- $\pi(x) = x,$ *if x is a variable*
- $\pi(f(t_1, \ldots, t_n)) = \pi(t_i),$ *if $\pi(f) = i$*
- $\pi(f(t_1, \ldots, t_n)) = f(\pi(t_{i_1}), \ldots, \pi(t_{i_k})),$ *if $\pi(f) = [i_1, \ldots, i_k]$*

An argument filter with $\pi(f) = i$ for some f is called collapsing*. We extend π to a mapping from TRSs to generalized TRSs in the obvious way: $\pi(\mathcal{R}) = \{\pi(\ell) \to \pi(r) \mid \ell \to r \in \mathcal{R}\}$.*

As an example consider an argument filter π that eliminates the second argument of a function symbol f with arity 3, i.e., $\pi(f) = [1, 3]$. Then for any term t, $\pi(t)$ results from replacing all subterms $f(t_1, t_2, t_3)$ by $f(\pi(t_1), \pi(t_3))$. Moreover, one can also define an argument filter π' with $\pi'(f) = 3$. Then we replace all subterms of the form $f(t_1, t_2, t_3)$ by $\pi'(t_3)$.

Now instead of a reduction pair $(\succsim, \succ)$, one may use the reduction pair $(\succsim_\pi, \succ_\pi)$ with $s \succsim_\pi t$ iff $\pi(s) \succsim \pi(t)$ and $s \succ_\pi t$ iff $\pi(s) \succ \pi(t)$.

Now we can define a processor which deletes all pairs from $\mathcal{P}$ which are strictly decreasing w.r.t. a reduction pair and an argument filter (i.e., all pairs of $\mathcal{P}$ that are strictly decreasing w.r.t. $\succ_\pi$). The reason is that they cannot occur infinitely often in a chain.

To ease presentation, throughout this thesis we often interpret sets of rules and pairs as binary relations on terms. This allows us to write $\mathcal{P} \setminus \succ_\pi$ instead of $\{s \to t \in \mathcal{P} \mid s \not\succ_\pi t\}$ and we write $\mathcal{R} \subseteq \succsim_\pi$ instead of $\mathcal{R} \subseteq \{\ell \to r \in \mathcal{R} \mid \ell \succsim_\pi r\}$, for example.

Theorem 4.2 (Processors Based on Reduction Pairs). *Let $(\succsim, \succ)$ be a reduction pair and π be an argument filter. Then the following processor Proc is sound and complete. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, Proc returns*

- $\{(\mathcal{P} \setminus \succ_\pi, \mathcal{Q}, \mathcal{R}, f)\}$, *if*
    - $\mathcal{P} \subseteq \succ_\pi \cup \succsim_\pi$ *and*
    - $\mathcal{R} \subseteq \succsim_\pi$.

---

[15] Note that this is equivalent to the requirement $\succsim \circ \succ \; \subseteq \; \succ$ or $\succ \circ \succsim \; \subseteq \; \succ$.

- $\{(\mathcal{P},\mathcal{Q},\mathcal{R},f)\}$, *otherwise*

First note that this processor was already presented in [GTS05a, Theorem 19], and its formulation is inspired by the recursive decomposition algorithm in [HM05, Theorem 22]. Indeed, if one uses this processor in combination with the processors based on the dependency graph then one can completely mimic the recursive algorithm of [HM05]. However, our approach is more flexible, since we can at every time apply arbitrary other processors. That this is sometimes necessary, we will have seen when finally solving the running example of this chapter (Example 4.33). There, a larger part of the proof tree for a difficult DP problem is shown, and it turns out that after the application of the processors based on the dependency graph two other (upcoming) processors are applied, before using a reduction pair processor.

We demonstrate the use of the reduction pair processor of Theorem 4.2 to finally prove termination of the running example of Section 3. As in this chapter we will only rarely exploit the graph structure of a DP problem $(\mathcal{P},\mathcal{Q},\mathcal{R},f)$, we identify the graph $\mathcal{P} = (N,E)$ with its set of nodes N.

Example 4.3. At the end of Example 3.38 we had to solve two remaining DP problems. The DP problem $\mathcal{D}_7 = (\{(23),(25)\}, \mathit{lhs}(\mathcal{R}'), \mathcal{R}', \mathbf{m})$ with $\mathcal{R}' = \{(9),(10),(11),(12),(13)\}$ can easily be handled by Theorem 4.2. First, we build the constraints of the reduction pair processor to delete both pairs (23) and (25) of the DP problem.

$$\begin{aligned}
\mathsf{MUL}(x,y,z) &\succ_\pi \mathsf{IF}(\mathsf{isZero}(x),x,y,z) && (23)\\
\mathsf{IF}(\mathsf{false},\mathsf{s}(x),y,z) &\succ_\pi \mathsf{MUL}(x,y,\mathsf{plus}(y,z)) && (25)\\
\mathsf{isZero}(\mathsf{0}) &\succsim_\pi \mathsf{true} && (9)\\
\mathsf{isZero}(\mathsf{s}(x)) &\succsim_\pi \mathsf{false} && (10)\\
\mathsf{plus}(\mathsf{0},y) &\succsim_\pi y && (11)\\
\mathsf{plus}(\mathsf{s}(x),y) &\succsim_\pi \mathsf{s}(\mathsf{plus}(x,y)) && (12)\\
\mathsf{plus}(\mathsf{plus}(x,y),z) &\succsim_\pi \mathsf{plus}(x,\mathsf{plus}(y,z)) && (13)
\end{aligned}$$

By choosing $\pi(\mathsf{IF}) = 2$, $\pi(\mathsf{MUL}) = \pi(\mathsf{s}) = [1]$, $\pi(\mathsf{isZero}) = \pi(\mathsf{true}) = \pi(\mathsf{false}) = \pi(\mathsf{0}) = [\,]$, and $\pi(\mathsf{plus}) = [1,2]$ we obtain the following filtered constraints.

$$\begin{aligned}
\mathsf{MUL}(x) &\succ x && (23)\\
\mathsf{s}(x) &\succ \mathsf{MUL}(x) && (25)\\
\mathsf{isZero} &\succsim \mathsf{true} && (9)\\
\mathsf{isZero} &\succsim \mathsf{false} && (10)\\
\mathsf{plus}(\mathsf{0},y) &\succsim y && (11)\\
\mathsf{plus}(\mathsf{s}(x),y) &\succsim \mathsf{s}(\mathsf{plus}(x,y)) && (12)\\
\mathsf{plus}(\mathsf{plus}(x,y),z) &\succsim \mathsf{plus}(x,\mathsf{plus}(y,z)) && (13)
\end{aligned}$$

These constraints are all satisfied if we choose for $\succ$ an LPO with precedence $\mathsf{plus} > \mathsf{s} > \mathsf{MUL}$, $\mathsf{isZero} > \mathsf{true}$, and $\mathsf{isZero} > \mathsf{false}$. Thus, the result of the reduction pair processor returns a DP problem where $\mathcal{P}$ is the empty set. This new DP problem is then easily solved by the processor of Theorem 3.4.

For the other DP problem $\mathcal{D}_1 = (\{(19),(20),(21)\}, \varnothing, \mathcal{R}, \mathbf{m})$ containing the dependency pairs of the plus-rules we get even more constraints. Note that we are not able to simplify this problem by the usable rules processor of Theorem 3.25 since $\mathcal{Q} = \varnothing$ and since there are variables in right-hand sides of all pairs in $\mathcal{D}_1$. Hence, we have to build constraints for the whole TRS $\mathcal{R}$. It turns out that due to the additional constraints for the rules the constraints of the reduction pair processor are unsatisfiable for reduction pairs based on RPO in combination with an argument filter. Nevertheless, using the polynomial order $\mathcal{P}ol$ defined by

$$\begin{array}{rcl}
\mathcal{P}ol(\mathsf{PLUS}(x,y)) & = & x \\
\mathcal{P}ol(\mathsf{plus}(x,y)) & = & x+y+1 \\
\mathcal{P}ol(\mathsf{s}(x)) & = & x+1 \\
\mathcal{P}ol(\mathsf{0}) & = & 0 \\
\mathcal{P}ol(\mathsf{times}(x,y)) & = & (x+1)y \\
\mathcal{P}ol(\mathsf{mul}(x,y,z)) & = & (x+1)y+z \\
\mathcal{P}ol(\mathsf{isZero}(x)) & = & 0 \\
\mathcal{P}ol(\mathsf{true}) & = & 0 \\
\mathcal{P}ol(\mathsf{false}) & = & 0 \\
\mathcal{P}ol(\mathsf{f}(x,y,z)) & = & 0
\end{array}$$

satisfies all constraints and we can delete all pairs in $\mathcal{D}_1$.[16] The resulting DP problem can again be solved by the dependency graph processor. Thus, we have now proven termination of the running example of Chapter 3.

Note that in the previous example we have provided suitable argument filters and orders. But this search can also be automated. Techniques to search for argument filters and orders efficiently have been developed in [CLS06, CMTU05, CSL$^+$06, FGM$^+$07, GTSF03, GTSF06, HM05, STA$^+$07, ZHM07]. Some of these techniques describe stand-alone constraint solvers whereas the (currently) more efficient approaches encode the constraints into SAT and then apply modern SAT solver.

As we have already mentioned in the previous example there is a major difference between proving termination and proving innermost termination. In the innermost case we can usually delete many rules of the TRS that are not usable. And having less rules implies having less constraints for the reduction pair processor. In Example 4.3 even the constraints for the whole TRS are solvable by standard orders but sometimes it may be the case that the difference between the whole TRS in the termination case and the set of usable rules in the innermost case is essential.

This is demonstrates in the following new running example of this chapter to compute the division function.

Example 4.4. In the following TRS $\mathcal{R}$ the function quot computes the division function on integers. We represent the integers by s, $\mathsf{0}$, and p where $\mathsf{p}(x)$ is the predecessor of a number x. To avoid duplicate representations of numbers ($\mathsf{0} = \mathsf{s}(\mathsf{p}(\mathsf{0})) = \mathsf{p}(\mathsf{s}(\mathsf{0})) = \dots$), we use rules (33) and (34) and furthermore, we set $\mathcal{Q} = \{\mathsf{s}(\mathsf{p}(x)), \mathsf{p}(\mathsf{s}(x))\}$ to ensure that numbers are simplified whenever possible. Here we see an example where the flexibility of

---

[16] Note that when using a polynomial order we do not preprocess the constraints by an argument filter as a polynomial order can ignore arguments on its own. However, one can identify the implicit argument filter of a polynomial order and use it for the improved reduction pair processors which use the argument filter to reduce the set of constraints, cf. Sections 4.4 and 4.5. For example, the implicit argument filter for $\mathcal{P}ol$ is defined by $\pi(\mathsf{PLUS}) = 1$, $\pi(\mathsf{isZero}) = \pi(\mathsf{f}) = [\,]$, and $\pi(f) = [1, \dots, ar(f)]$ for all remaining function symbols f.

$\mathcal{Q}$ allows a natural modeling of the integers. Setting $\mathcal{Q} = lhs(\mathcal{R})$ would be to restrictive because then the rules (40) and (41) become useless.

The computation of the division is mainly performed using the div-rules which only work for natural numbers. Therefore, the function quot which is defined also for negative numbers ensures that div is only called with natural numbers by corresponding negations.

$$\begin{aligned}
\mathsf{s}(\mathsf{p}(x)) &\to x && (33)\\
\mathsf{p}(\mathsf{s}(x)) &\to x && (34)\\
\mathsf{minus}(x,\mathsf{0}) &\to x && (35)\\
\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{minus}(x,y) && (36)\\
\mathsf{negate}(\mathsf{0}) &\to \mathsf{0} && (37)\\
\mathsf{negate}(\mathsf{p}(x)) &\to \mathsf{s}(\mathsf{negate}(x)) && (38)\\
\mathsf{negate}(\mathsf{s}(x)) &\to \mathsf{p}(\mathsf{negate}(x)) && (39)\\
\mathsf{negate}(\mathsf{negate}(x)) &\to x && (40)\\
\mathsf{negate}(\mathsf{minus}(\mathsf{0},x)) &\to x && (41)\\
\mathsf{div}(\mathsf{0},\mathsf{s}(x)) &\to \mathsf{0} && (42)\\
\mathsf{div}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),\mathsf{s}(y))) && (43)\\
\mathsf{quot}(x,\mathsf{0}) &\to \mathsf{error} && (44)\\
\mathsf{quot}(x,\mathsf{p}(y)) &\to \mathsf{negate}(\mathsf{quot}(x,\mathsf{negate}(\mathsf{p}(y)))) && (45)\\
\mathsf{quot}(\mathsf{p}(x),y) &\to \mathsf{negate}(\mathsf{quot}(\mathsf{negate}(\mathsf{p}(x)),y)) && (46)\\
\mathsf{quot}(\mathsf{0},\mathsf{s}(x)) &\to \mathsf{0} && (47)\\
\mathsf{quot}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{div}(\mathsf{s}(x),\mathsf{s}(y)) && (48)
\end{aligned}$$

When applying the processors based on the dependency graph on the initial DP problem, we obtain four DP problems $(\mathcal{P}_i, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ with the following pairs.

$$\begin{aligned}
\mathsf{MINUS}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{MINUS}(x,y) && (49)\\
\mathsf{NEGATE}(\mathsf{p}(x)) &\to \mathsf{NEGATE}(x) && (50)\\
\mathsf{NEGATE}(\mathsf{s}(x)) &\to \mathsf{NEGATE}(x) && (51)\\
\mathsf{DIV}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{DIV}(\mathsf{minus}(x,y),\mathsf{s}(y)) && (52)\\
\mathsf{QUOT}(x,\mathsf{p}(y)) &\to \mathsf{QUOT}(x,\mathsf{negate}(\mathsf{p}(y))) && (53)\\
\mathsf{QUOT}(\mathsf{p}(x),y) &\to \mathsf{QUOT}(\mathsf{negate}(\mathsf{p}(x)),y) && (54)
\end{aligned}$$

Here, $\mathcal{P}_1 = \{(49)\}$, $\mathcal{P}_2 = \{(50),(51)\}$, $\mathcal{P}_3 = \{(52)\}$, and $\mathcal{P}_4 = \{(53),(54)\}$. None of these four DP problems can be handled by the reduction pair processor of Theorem 4.2 if $\succsim$ is a *quasi-simplification order* (i.e., a quasi-order containing the embedding order). We prove this claim only for the first DP problem $(\mathcal{P}_1, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ since the claim can be shown for the other three DP problems in a similar way.

If we want to apply Theorem 4.2 successfully then we need at least one strict decrease of a pair in $\mathcal{P}_1$. Thus, the constraints ensure both, that the left-hand side of pair (49) is in relation to the right-hand side w.r.t. $\succ_\pi$, and that the rewrite relation of $\mathcal{R}$ is contained in $\succsim_\pi$. Then due to rules (35) and (41) we must have $\pi(\mathsf{minus}) = [1,2]$. From (49) we obtain $\mathsf{MINUS}(\mathsf{s}(x),\mathsf{s}(y)) \succ_\pi \mathsf{MINUS}(x,y)$ and as $\succ_\pi$ is stable we instantiate x and y by $t = \mathsf{div}(\mathsf{s}(z),\mathsf{s}(\mathsf{s}(z)))$ to get the inequality $\mathsf{MINUS}(\mathsf{s}(t),\mathsf{s}(t)) \succ_\pi \mathsf{MINUS}(t,t)$. As one can rewrite t to the term $t' = \mathsf{s}(\mathsf{div}(\mathsf{minus}(z,\mathsf{s}(z)),\mathsf{s}(\mathsf{s}(z))))$ by (43), we can use $\to_\mathcal{R} \subseteq \succsim_\pi$

to obtain the inequality $t \succsim_\pi t'$. Moreover, as $\succsim$ is a quasi-simplification order and as π does not drop the second argument of minus we know $\mathsf{minus}(z, \mathsf{s}(z)) \succsim_\pi \mathsf{s}(z)$. By monotonicity of $\succsim_\pi$ we conclude $t' \succsim_\pi \mathsf{s}(t)$. Using the compatibility of $\succsim_\pi$ and $\succ_\pi$ we end in a contradiction to the well-foundedness of $\succ_\pi$.

$$\mathsf{MINUS}(\mathsf{s}(t), \mathsf{s}(t)) \succ_\pi \mathsf{MINUS}(t, t) \succsim_\pi \mathsf{MINUS}(t', t') \succsim_\pi \mathsf{MINUS}(\mathsf{s}(t), \mathsf{s}(t))$$

Note that $\mathcal{R}$ does not belong to a class of rewrite systems where innermost termination implies termination. The first two rules (33) and (34) are overlapping with many other rules at non-root positions. These two rules are overlapping with the left-hand sides of each dependency pair of $\mathcal{P}_1 \cup \cdots \cup \mathcal{P}_4$, too. Thus, these two rules also prohibit the use of the processor of Theorem 3.14 to switch to innermost termination for the four DP problems.

And as we are not in the innermost case the usable rules processor of Theorem 3.25 cannot delete any rule of $\mathcal{R}$. The problem with the improved estimation of Definition 3.26 is that due to condition (iv) every rule is usable. It is even more problematic that all rules are usable w.r.t. the semantic definition of usable rules in Definition 3.24. Hence, the problem is not that the improved estimation of usable rules in Definition 3.26 is not good enough in this example.

To conclude, we are currently stuck in the termination proof of this example using the previous processors.

4.2. Needed Rules

To handle the DP problems of Example 4.4 we will now show how one can drop condition (iv) of Definition 3.26 even if one is not in the innermost case, i.e., even if $\mathit{NF}(\mathcal{Q}) \not\subseteq \mathit{NF}(\mathcal{R})$. That this is not possible in general was already shown in Example 3.29. Note that when dropping condition (iv) one does not estimate the usable rules function $\mathcal{U}$ any more. To distinguish the two versions of usable rules we call the new version *needed rules.*[17]

Definition 4.5 (Needed Rules). *Let $\mathcal{Q}$ and $\mathcal{S}$ be set of terms, let $\mathcal{R}$ be a TRS, and let ECap be an estimated Cap-function. The* needed rules *of a term t are defined as the smallest set $\mathcal{N}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{R}$ such that*

(i) If $t = f(t_1, \ldots, t_n)$, $\ell = f(\ell_1, \ldots, \ell_n) \to r \in \mathcal{R}$, and if the terms $f(\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1), \ldots, \mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n))$ and ℓ are unifiable with the mgu δ such that all terms in $(\{\ell_1, \ldots, \ell_n\} \cup \mathcal{S})\delta$ are in $\mathcal{Q}$-normal form, then $\ell \to r \in \mathcal{N}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$.

(ii) If $t = f(t_1, \ldots, t_n)$ then $\mathcal{N}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_i) \subseteq \mathcal{N}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$.

(iii) If $\ell = f(\ell_1, \ldots, \ell_n) \to r \in \mathcal{N}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$ then $\mathcal{N}^{\{\ell_1, \ldots, \ell_n\}}_{\mathcal{R},\mathcal{Q}}(r) \subseteq \mathcal{N}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$.

As before, $\mathcal{N}(\mathcal{P}, \mathcal{Q}, \mathcal{R}) = \bigcup_{s \to t \in \mathcal{P}} \mathcal{N}^{\{s\}}_{\mathcal{R},\mathcal{Q}}(t)$.

---

[17] In the literature even the needed rules were called usable rules. This was possible as there was no semantic notion of usable rules before and one could just limit the use of usable rules in certain processors. For example, in previous versions of the usable rules processor of Theorem 3.25 and the star-estimation of the dependency graph of Definition 3.31 it was only allowed to integrate the usable rules in the innermost case. This is in contrast to this thesis where usable rules can be used without the restriction to the innermost case.

As the main result of this section we will now show that in minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chains the TRS $\mathcal{R}$ can be replaced by the needed rules together with the TRS $\mathcal{C}_\varepsilon = \{\mathsf{c}(x, y) \to x, \mathsf{c}(x, y) \to y\}$. Here, c is some new function symbol not occurring in the DP problem.

Note that for every DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ with $NF(\mathcal{Q}) \subseteq NF(\mathcal{R})$ the improved usable rules Definition 3.26 coincide with the needed rules, i.e., $\mathcal{IU}(\mathcal{P}, \mathcal{Q}, \mathcal{R}) = \mathcal{N}(\mathcal{P}, \mathcal{Q}, \mathcal{R})$. Hence, with the main result we will be almost as powerful in the termination case as in the innermost termination case when solving the constraints of the reduction pair processor.

To obtain the main result, we use the following idea to simulate every reduction step with $\mathcal{R}$ with the set of needed rules $\mathcal{N}$ together with $\mathcal{C}_\varepsilon$. In case a term t can be reduced with an unneeded rule, we store all possible reducts of t in a set of terms. This set will be finite as the terms are terminating due to minimality. Hence, the set can be encoded in a single term representing a list where the new binary function c is used as list constructor. This encoding is performed using the *Comp* function.

Definition 4.6 (*Comp*)**.** *We assume a fixed total order $>$ on terms. Let c be a new function symbol and $\perp$ be a new variable. We define the function Comp from finite sets of terms to terms as follows.*

- $\mathit{Comp}(\varnothing) = \perp$
- $\mathit{Comp}(\{t\} \uplus M) = \mathsf{c}(t, \mathit{Comp}(M))$*, where* $t < s$ *for every* $s \in M$*.*

The total order is needed to ensure that *Comp* is well defined. It enforces that *Comp* always visits the terms in M in the same order. The TRS $\mathcal{C}_\varepsilon$ can now be used to extract every term of M that is encoded in $\mathit{Comp}(M)$.

Lemma 4.7 (Properties of *Comp*)**.** $\mathit{Comp}(M) \xrightarrow{\mathcal{Q}}{}^{+}_{\mathcal{C}_\varepsilon} t$ *if* $t \in M$ *and* $\mathit{Comp}(M) \in NF(\mathcal{Q})$*.*

Now the only missing step is to apply a transformation that replaces those terms t which can be rewritten by an unneeded rule by the corresponding term that encodes the set of reducts of t. This is done in one of the cases of the upcoming transformation $\mathcal{I}$. However, if t can only be rewritten by needed rules then $\mathcal{I}$ does not replace anything and we can simulate the reduction with the corresponding needed rule. Thus, any $\mathcal{R}$-reduction of t can be simulated by an $\mathcal{N} \cup \mathcal{C}_\varepsilon$-reduction of $\mathcal{I}(t)$. This behavior is indicated in the middle rows of Figure 4.8.

However, what we want to do is to transform a minimal chain $s_1 \to t_1, s_2 \to t_2, \ldots$ with σ as substitution into a chain where $\mathcal{I}(\sigma)$ is used as substitution. Therefore, we also have to take care that $\mathcal{I}(s_i\sigma)$ can be related to $s_i\mathcal{I}(\sigma)$ and there should also be a relation between $\mathcal{I}(t_i\sigma)$ and $t_i\mathcal{I}(\sigma)$. This is depicted in the last two rows of Figure 4.8. This will finally allow us to replace $\mathcal{R}$ by $\mathcal{N} \cup \mathcal{C}_\varepsilon$ when building chains.

The following definition introduces $\mathcal{I}$ formally. Similar versions have been developed for weaker variants of needed rules [GTSF06, HM07, TGS04, Urb01]. All these previous approaches define needed rules by just looking at the root symbols instead of unification, they do not allow an arbitrary estimation of *Cap*, they do not integrate checks on $\mathcal{Q}$-normal forms, and they do not handle the strategy given by $\mathcal{Q}$. Moreover, for the first time we also give sufficient conditions that the resulting chain is also minimal. Hence, our result extends the previous work substantially. Nevertheless, all these extensions require a more complex definition of $\mathcal{I}$. After the exact definition we will explain the additional elements σ and $\mathcal{S}_{\mathit{all}}$ which have not been discussed so far.

$$\begin{tikzcd}
s_1\sigma \arrow[r] \arrow[d, equal] & t_1\sigma \arrow[d, equal] & & s_2\sigma \arrow[r] \arrow[d, equal] & t_2\sigma \arrow[d, equal] & \cdots & (\mathcal{P},\mathcal{Q},\mathcal{R})\text{-chain} \\
s_1\sigma \arrow[d, dotted, "\mathcal{I}"] & t_1\sigma \arrow[rr, "*" near end, "\mathcal{R}"' near end, ""{name=A}] \arrow[d, dotted, "\mathcal{I}"] & & s_1\sigma \arrow[d, dotted, "\mathcal{I}"] & t_1\sigma \arrow[r, "*" near end, "\mathcal{R}"' near end, ""{name=B}] \arrow[d, dotted, "\mathcal{I}"] & \cdots & \\
\mathcal{I}(s_1\sigma) \arrow[d, "\mathcal{C}_\varepsilon"', "*" near end] & \mathcal{I}(t_1\sigma) \arrow[rr, "*" near end, "\mathcal{N}\cup\mathcal{C}_\varepsilon"' near end, ""{name=C}] \arrow[d, equal] & & \mathcal{I}(s_2\sigma) \arrow[d, "\mathcal{C}_\varepsilon"', "*" near end] & \mathcal{I}(t_2\sigma) \arrow[r, "*" near end, "\mathcal{N}\cup\mathcal{C}_\varepsilon"' near end, ""{name=D}] \arrow[d, equal] & \cdots & \\
s_1\mathcal{I}(\sigma) \arrow[r] & t_1\mathcal{I}(\sigma) & & s_2\mathcal{I}(\sigma) \arrow[r] & t_2\mathcal{I}(\sigma) & \cdots & (\mathcal{P},\mathcal{Q},\mathcal{N}\cup\mathcal{C}_\varepsilon)\text{-chain}
\arrow[from=A, to=C, dotted, "\mathcal{I}"]
\arrow[from=B, to=D, dotted, "\mathcal{I}"]
\end{tikzcd}$$

Figure 4.8.: Transformation of chains

Definition 4.9 ($\mathcal{I}$)**.** *Let $u_1 \to v_1, u_2 \to v_2, \ldots$ be an infinite minimal $(\mathcal{P},\mathcal{Q},\mathcal{R})$-chain and let $\mathcal{N} = \mathcal{N}(\mathcal{P},\mathcal{Q},\mathcal{R})$ be the set of needed rules of the given DP problem. W.l.o.g. we assume that σ is the substitution used for instantiating every $u_i \to v_i$. Moreover, whenever in the reduction of $v_i\sigma$ a rule $\ell_j \to r_j \in \mathcal{R}$ is applied, then by renaming the variables in the rule we again assume that the rule is instantiated by σ in that rewrite step. Let $\mathcal{S}_{all}$ contain all u_i and all direct subterms of each ℓ_j. Let c be the new constant and let $\bot$ be the new variable which are introduced by Comp. We define the mapping $\mathcal{I}$ from terms of $\mathcal{T}(\mathcal{F},\mathcal{V})$ that terminate w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ to terms of $\mathcal{T}(\mathcal{F} \uplus \{\mathsf{c}\}, \mathcal{V} \uplus \{\bot\})$ as follows.*

- *$\mathcal{I}(x) = x$ for every variable x*
- *$\mathcal{I}(f(t_1,\ldots,t_n)) = f(\mathcal{I}(t_1),\ldots,\mathcal{I}(t_n))$, if there is no rule $\ell \to r \in \mathcal{R} \setminus \mathcal{N}$ such that $f(Cap^{\mathcal{S}_{all}\sigma}_{\mathcal{R},\mathcal{Q}}(t_1),\ldots Cap^{\mathcal{S}_{all}\sigma}_{\mathcal{R},\mathcal{Q}}(t_n))$ unifies with ℓ by some mgu μ where $(\mathcal{S}_{all}\sigma \cup \{\ell|_1,\ldots,\ell|_n\})\mu \subseteq NF(\mathcal{Q})$*
- *$\mathcal{I}(f(t_1,\ldots,t_n)) = \mathsf{c}(f(\mathcal{I}(t_1),\ldots,\mathcal{I}(t_n)), Comp(Red(f(t_1,\ldots,t_n))))$, otherwise.*

Here, $Red(t) = \{\mathcal{I}(s) \mid t \xrightarrow{\mathcal{Q}}{}^{+}_{\mathcal{R}} s, s \in NF(\mathcal{Q})\}$.
We extend $\mathcal{I}$ to substitutions by defining the substitution $\mathcal{I}(\sigma)$ as $\mathcal{I}(\sigma)(x) = \mathcal{I}(x\sigma)$.

To prove the main result in Lemma 4.11 that minimal $(\mathcal{P},\mathcal{Q},\mathcal{R})$-chains are $(\mathcal{P},\mathcal{Q},\mathcal{N} \cup \mathcal{C}_\varepsilon)$-chains with the help of $\mathcal{I}$, we need to establish a connection between $\mathcal{I}$ and the definition of needed rules: whenever the unification with a left-hand side ℓ succeeds in the definition of $\mathcal{I}$, then the corresponding unification with ℓ must also succeed in case (i) of Definition 4.5.

However, there are three differences in the unifications performed in $\mathcal{I}$ and those performed in Definition 4.5. First, the needed rules are computed by looking at rules and pairs whereas $\mathcal{I}$ works on terms that are *instances* of left- or right-hand sides of rules and pairs. Second, in Definition 4.5 we apply an *estimated Cap*-function whereas $\mathcal{I}$ uses *Cap* itself. And third, the parameter $\mathcal{S}$ for *ECap* is chosen *locally* for each pair and rule in Definition 4.5 whereas we need a *global* set $\mathcal{S}$ for $\mathcal{I}$.

To obtain the required connection we first show the following additional property of *Cap*. It states that applying *Cap* on an instantiated term $t\sigma$ is more precise than if we first apply an estimated *Cap*-function *ECap* on t and then instantiate the resulting term by σ. Moreover, the set $\mathcal{S}$ that is used for *ECap* has to be smaller than the corresponding set that is used for *Cap*. This is the reason we why defined $\mathcal{S}_{all}$ to contain all direct subterms of left-hand sides of rules and all left-hand sides of pairs that are used in the chain.[18] Then indeed all three differences are handled and we can state the main lemma of this section after this auxiliary lemma about *Cap*.

Lemma 4.10 (Properties of *Cap*)**.** *Let $\mathcal{Q}$, $\mathcal{S}$, and $\mathcal{T}$ be sets of terms, let $\mathcal{R}$ be a TRS, and let σ be a substitution. If $\mathcal{S}\sigma \subseteq \mathcal{T}$ then for every term t there is a substitution δ such that $\mathit{Cap}^{\mathcal{T}}_{\mathcal{R},\mathcal{Q}}(t\sigma) = \mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)\sigma\delta$ where δ only instantiates the variables that are introduced by $\mathit{Cap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$.*

Note that Lemma 4.10 is not valid if one replaces *Cap* by *ECap*. This is the reason for using *Cap* in the definition of $\mathcal{I}$ instead of an estimation.

Lemma 4.11 (Properties of $\mathcal{I}$)**.** *Let $\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{N}, u_1 \to v_1, u_2, \to v_2, \ldots, \mathcal{S}_{all}, \sigma$ be as in Definition 4.9. Let t be terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$. Let $\mathit{Inv}(s)$ be the property, that all subterms s' of s with $\mathit{root}(s') = \mathsf{c}$ are in $\mathcal{Q}$-normal form.*

(i) If $t \in \mathit{NF}(\mathcal{Q})$ then $\mathcal{I}(t) \in \mathit{NF}(\mathcal{Q})$.

(ii) If $\mathcal{N}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{N}$ and $\mathcal{S} \subseteq \mathcal{S}_{all}$ then $\mathcal{I}(t\sigma) = t\mathcal{I}(\sigma)$.

(iii) If $t\sigma \in \mathit{NF}(\mathcal{Q})$ then $\mathcal{I}(t\sigma) \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{C}_\varepsilon} t\mathcal{I}(\sigma)$ and $t\mathcal{I}(\sigma) \in \mathit{NF}(\mathcal{Q})$.

(iv) If $t \xrightarrow{\mathcal{Q}}{}^{+}_{\mathcal{R}} s$, $\mathit{Inv}(\mathcal{I}(t))$, and $\mathcal{I}(t)$ is built by the third case then $\mathcal{I}(t) \xrightarrow{\mathcal{Q}}{}^{+}_{\mathcal{C}_\varepsilon} \mathcal{I}(s)$.

(v) If $t \xrightarrow{\mathcal{Q}}_{\mathcal{R}} s$ is a reduction at the root position and $\mathcal{I}(t)$ is built by the second case then $\mathcal{I}(t) \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{C}_\varepsilon} \xrightarrow{\mathcal{Q}}_{\mathcal{N}} \mathcal{I}(s)$ and $\mathit{Inv}(\mathcal{I}(s))$.

(vi) If $t \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} s$, $s \in \mathit{NF}(\mathcal{Q})$, and $\mathit{Inv}(\mathcal{I}(t))$ then $\mathcal{I}(t) \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{N}\cup\mathcal{C}_\varepsilon} \mathcal{I}(s)$.

(vii) If $\mathcal{Q} = \varnothing$ and if $\mathcal{M} \subseteq \mathcal{R}$ is left-linear then $\mathcal{I}(t)$ is terminating w.r.t. $\mathcal{M} \cup \mathcal{C}_\varepsilon$.

(viii) If $\mathcal{N} \subseteq \mathcal{M} \subseteq \mathcal{R}$ then $u_1 \to v_1, u_2 \to v_2, \ldots$ is a $(\mathcal{P}, \mathcal{Q}, \mathcal{M} \cup \mathcal{C}_\varepsilon)$-chain. Moreover, if $\mathcal{Q} = \varnothing$ and if $\mathcal{M}$ is left-linear then the chain is minimal.

Using this important lemma we will formulate three different processors. The first processor returns quite similar results compared to the usable rules processor of Theorem 3.25 in the innermost case.

Theorem 4.12 (Processors Based on Needed Rules)**.** *Let ECap be an estimated Cap-function that is used to compute the needed rules in Definition 4.5. The following processor Proc is sound. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ with $\mathcal{N}(\mathcal{P}, \mathcal{Q}, \mathcal{R}) \subseteq \mathcal{M} \subseteq \mathcal{R}$, Proc returns*

- *$\{(\mathcal{P}, \mathcal{Q}, \mathcal{M} \cup \mathcal{C}_\varepsilon, f')\}$, if $f = \mathbf{m}$.*
  Here, $f' = \mathbf{m}$ if $\mathcal{Q} = \varnothing$ and $\mathcal{M}$ is left-linear, and $f' = \mathbf{a}$, otherwise.

---

[18] Putting all direct subterms of left-hand sides of $\mathcal{R}$ and all left-hand sides of pairs of $\mathcal{P}$ into $\mathcal{S}_{all}$ would be unsound as some pairs and rules may contain subterms which are not in $\mathcal{Q}$-normal form. And if $\mathcal{S}_{all}$ contains a term which is not in $\mathcal{Q}$-normal form then the third case of $\mathcal{I}$ is never used. In that case we do not obtain a chain if we replace σ by $\mathcal{I}(\sigma)$. As an example consider $\mathcal{Q} = \{\mathsf{a}\}$, $\mathcal{P} = \{\mathsf{F}(\mathsf{a},\mathsf{a},\mathsf{a}) \to \mathsf{F}(\mathsf{b},\mathsf{b},\mathsf{b}), \mathsf{F}(\mathsf{0},\mathsf{1},x) \to \mathsf{F}(x,x,x)\}$, and $\mathcal{R} = \{\mathsf{d} \to \mathsf{0}, \mathsf{d} \to \mathsf{1}\}$. Then there obviously is a minimal infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain using only the second pair of $\mathcal{P}$ if we instantiate x by d. However, if $\mathsf{F}(\mathsf{a},\mathsf{a},\mathsf{a}) \in \mathcal{S}_{all}$ then $\mathcal{I}(\sigma)(x) = \mathsf{d}$ and it is not possible to reduce $\mathsf{F}(x,x,x)\mathcal{I}(\sigma) = \mathsf{F}(\mathsf{d},\mathsf{d},\mathsf{d})$ to an instance of $\mathsf{F}(\mathsf{0},\mathsf{1},x')$ using the $\mathcal{C}_\varepsilon$-rules.

- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$, *otherwise.*

Note that we introduced the additional set $\mathcal{M}$ in the needed rules processor to be able to apply the result for arbitrary estimations of the needed rules. For example it is possible to drop the normal form conditions in Definition 4.5 or one can just compare the outermost symbols instead of performing the unifications in Definition 4.5. Due to Lemma 2.4 it might seem obvious that one can replace $\mathcal{N}$ by a larger set $\mathcal{M}$, however it is not obvious that then minimality can be obtained.

The needed rules processor has two weaknesses that do not occur in the similar usable rules processor of Theorem 3.25. Therefore, one should always prefer the usable rules processor in the innermost case as then there are always less usable rules than needed rules.

The first weakness of the needed rules processor is its incompleteness. This is a disadvantage regardless of whether the original DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ is finite or infinite. If $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ is finite then it may happen that the resulting DP problems are not finite any more and one fails in proving termination. On the other hand, if $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ is infinite then even if one can prove that one of the resulting DP problems is infinite one cannot conclude infiniteness of $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$. Thus, applying incomplete processors is nearly useless to prove non-termination.

By the TRS of Toyama [Toy87] it can be shown that completeness cannot be achieved for the needed rules processor. If $\mathcal{R} = \{\mathsf{f}(\mathsf{0}, \mathsf{1}, x) \to \mathsf{f}(x, x, x)\}$ then the initial DP problem $(\mathit{DP}(\mathcal{R}), \varnothing, \mathcal{R}, \mathbf{m})$ is not infinite. However, an application of Theorem 4.12 results in the DP problem $(\mathit{DP}(\mathcal{R}), \varnothing, \mathcal{C}_\varepsilon, \mathbf{m})$ which is infinite:

$$\mathsf{F}(\mathsf{0}, \mathsf{1}, \mathsf{c}(\mathsf{0}, \mathsf{1})) \to \mathsf{F}(\mathsf{c}(\mathsf{0}, \mathsf{1}), \mathsf{c}(\mathsf{0}, \mathsf{1}), \mathsf{c}(\mathsf{0}, \mathsf{1})) \to^2_{\mathcal{C}_\varepsilon} \mathsf{F}(\mathsf{0}, \mathsf{1}, \mathsf{c}(\mathsf{0}, \mathsf{1})) \to \dots$$

Nevertheless, sometimes it is useful to apply incomplete processors as they may be the missing step to obtain a termination proof. We will see in the following example that the first three DP problems can be solved with the help of the (incomplete) needed rules processor.

Example 4.13. We continue to prove termination of Example 4.4 with our new needed rules processor of Theorem 4.12. First recall the DP problems $(\{(49)\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ and $(\{(50), (51)\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ that correspond to the minus and negate-functions, respectively. As neither of them does have any needed rules, we get the new problems $(\{(49)\}, \mathcal{Q}, \mathcal{C}_\varepsilon, \mathbf{a})$ and $(\{(50), (51)\}, \mathcal{Q}, \mathcal{C}_\varepsilon, \mathbf{a})$. Both new problems can be solved by the reduction pair processor of Theorem 4.2 by the embedding order. Here, it is not even necessary to use an argument filter, i.e., one can choose the identity argument filter with $\pi(t) = t$ for every term t. This is done by defining $\pi(f) = [1, \dots, n]$ for every symbol f with arity n.

In the DP problem $(\{(52)\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ originating from the div-function both minus-rules (35) and (36) are needed rules, and the s-rule (33) is needed, too. Therefore, the new DP problem $(\{(52)\}, \mathcal{Q}, \{(33), (35), (36)\} \cup \mathcal{C}_\varepsilon, \mathbf{a})$ is obtained by an application of Theorem 4.12. Again this can be solved by the reduction pair processor of Theorem 4.2 by the embedding order. Here, we need an argument filter with $\pi(\mathsf{minus}) = 1$. Then the constraint

$$\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) \succ_\pi \mathsf{DIV}(\mathsf{minus}(x, y), \mathsf{s}(y))$$

for the dependency pair (52) can be simplified to

$$\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) \succ \mathsf{DIV}(x, \mathsf{s}(y))$$

which is easy to satisfy. Remember that this argument filter would not be possible if we had to satisfy the constraint $\mathsf{negate}(\mathsf{minus}(0,x)) \succsim_\pi x$ for the unneeded rule (41).

Finally, for the DP problem $(\{(53),(54)\},\mathcal{Q},\mathcal{R},\mathbf{m})$ one obtains the needed rules $\{(33),(34),(37)-(41)\}$ for s, p, and negate. Unfortunately, the constraints of Theorem 4.2 for the corresponding new DP problem are not satisfiable if $\succsim$ is a quasi-simplification order. The reason is that π may not drop the argument of negate due to the collapsing negate-rule (40). In this case we obtain $\mathsf{negate}(\mathsf{p}(x)) \succsim_\pi \mathsf{p}(x)$ which clearly shows by monotonicity that none of the constraints for the dependency pairs (53) and (54) can be satisfied for the strict relation $\succ_\pi$.

The second weakness of the needed rules processor is the possible loss of the minimality flag. The processor requires that for the original DP problem $(\mathcal{P},\mathcal{Q},\mathcal{R},f)$ the flag f is set to $\mathbf{m}$, but it sometimes returns a DP problem where the flag is $\mathbf{a}$. Note that if the flag is $\mathbf{a}$ then it is not possible to switch to the innermost case by Theorem 3.14. Thus, for the resulting DP problem we are neither in the innermost case nor do we have to consider only minimal chains. DP problems of these kind are often hard to handle. When using orders one always has to build constraints for at least the set of usable rules which is most likely the whole TRS $\mathcal{R}$. (One cannot use the needed rules processor any more and also upcoming powerful processors in this chapter require an enabled minimality flag or the innermost case. For a complete list of processors that cannot be applied any more, we refer to the discussion in Section 2.2 on page 13 about why losing minimality is bad.)

The following examples show that Theorem 4.12 cannot be strengthened by changing the requirements on the minimality flags. One needs minimality of the input DP problem and minimality of the resulting DP problem cannot be guaranteed if the requirements of Theorem 4.12 are not satisfied.

Example 4.14. This example shows that the requirement $f = \mathbf{m}$ in Theorem 4.12 is essential. If $\mathcal{P} = \{\mathsf{F}(\mathsf{s}(x)) \to \mathsf{F}(x)\}$, $\mathcal{Q} = \varnothing$, and $\mathcal{R} = \{\mathsf{a} \to \mathsf{s}(\mathsf{a})\}$ then $\mathsf{F}(\mathsf{s}(x)) \to \mathsf{F}(x), \mathsf{F}(\mathsf{s}(x)) \to \mathsf{F}(x), \dots$ is an infinite chain: if we instantiate x by a then we obtain the infinite reduction $\mathsf{F}(\mathsf{s}(\mathsf{a})) \to_{\mathcal{P}} \mathsf{F}(\mathsf{a}) \to_{\mathcal{R}} \mathsf{F}(\mathsf{s}(\mathsf{a})) \to_{\mathcal{P}} \dots$. However, it is not a minimal chain.

Note that there are no needed rules. Thus, any processor that transforms the non-finite DP problem $(\mathcal{P},\mathcal{Q},\mathcal{R},\mathbf{a})$ into $(\mathcal{P},\mathcal{Q},\mathcal{C}_\varepsilon,\mathbf{a})$ would be unsound as clearly there is no infinite $(\mathcal{P},\mathcal{Q},\mathcal{C}_\varepsilon)$-chain.

Example 4.15. This example illustrates that for a non-empty set $\mathcal{Q}$, the resulting DP problem of the needed rules processor must have the minimality flag $\mathbf{a}$, even if $\mathcal{R}$ is left-linear. Let $\mathcal{P} = \{\mathsf{F}(x,0,1,y) \to \mathsf{F}(x,x,x,\mathsf{f}(\mathsf{g}(x)))\}$, let $\mathcal{Q} = \{\mathsf{g}(\mathsf{a})\}$, and let $\mathcal{R}$ consist of the following four rules.

$$\begin{array}{rcl@{\qquad\qquad}rcl} \mathsf{a} & \to & 0 & \mathsf{f}(\mathsf{g}(x)) & \to & \mathsf{h}(x,x) \\ \mathsf{a} & \to & 1 & \mathsf{h}(0,1) & \to & \mathsf{h}(0,1) \end{array}$$

There is an infinite minimal $(\mathcal{P},\mathcal{Q},\mathcal{R})$-chain, as can be shown using the substitution σ with $\sigma(x) = \mathsf{a}$ and $\sigma(y) = \mathsf{f}(\mathsf{g}(0))$. The reason is that the instantiated right-hand side of $\mathcal{P}$'s only pair now reduces to its instantiated left-hand side:

$$\begin{array}{rl} & \mathsf{F}(x,x,x,\mathsf{f}(\mathsf{g}(x)))\sigma \\ = & \mathsf{F}(\mathsf{a},\mathsf{a},\mathsf{a},\mathsf{f}(\mathsf{g}(\mathsf{a}))) \\ \xrightarrow{\mathcal{Q}}{}^{3}_{\mathcal{R}} & \mathsf{F}(\mathsf{a},0,1,\mathsf{f}(\mathsf{g}(0))) \\ = & \mathsf{F}(x,0,1,y)\sigma \end{array}$$

Note that $\mathsf{f}(\mathsf{g}(\mathsf{a}))$ is terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ as whenever a subterm of the form $\mathsf{h}(t_1, t_2)$ is reached then both t_i are either $\mathsf{0}$ or $\mathsf{1}$.

However, for $\mathcal{N} = \mathcal{N}(\mathcal{P}, \mathcal{Q}, \mathcal{R})$ the DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{N} \cup \mathcal{C}_\varepsilon, \mathbf{m})$ is finite and therefore, any processor which transforms $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ into $(\mathcal{P}, \mathcal{Q}, \mathcal{N} \cup \mathcal{C}_\varepsilon, \mathbf{m})$ is unsound. Here, the needed rules $\mathcal{N} = \mathcal{N}(\mathcal{P}, \mathcal{Q}, \mathcal{R})$ consists of the f-rule and the h-rule. To prove that $(\mathcal{P}, \mathcal{Q}, \mathcal{N} \cup \mathcal{C}_\varepsilon, \mathbf{m})$ is finite we have to show that there is no infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{N} \cup \mathcal{C}_\varepsilon)$-chain.

In any chain of length greater than 1, the right-hand side of $\mathcal{P}$'s rule has to be instantiated by a substitution σ such that all $x\sigma$ can be reduced to both $\mathsf{0}$ and $\mathsf{1}$ with $\mathcal{N} \cup \mathcal{C}_\varepsilon$. It is easy to see that then $x\sigma$ can also be reduced to $\mathsf{c}(\mathsf{0}, \mathsf{1})$ or to $\mathsf{c}(\mathsf{1}, \mathsf{0})$. Hence, the subterm $\mathsf{f}(\mathsf{g}(x))\sigma = \mathsf{f}(\mathsf{g}(x\sigma))$ of the instantiated right-hand side can be reduced to $\mathsf{f}(\mathsf{g}(\mathsf{c}(\mathsf{0}, \mathsf{1})))$ or to $\mathsf{f}(\mathsf{g}(\mathsf{c}(\mathsf{1}, \mathsf{0})))$ and further to the non-terminating term $\mathsf{h}(\mathsf{0}, \mathsf{1})$. So the instantiated right-hand side of $\mathcal{P}$'s pair is not terminating and thus, there is no minimal chain of length greater than 1.

Example 4.16. This example shows that minimality cannot be preserved by the needed rules processor in case that $\mathcal{R}$ is not left-linear even if $\mathcal{Q} = \varnothing$. Let $\mathcal{P}$ consist of the single pair

$$\mathsf{F}(\mathsf{0}, \mathsf{1}, x_1, \mathsf{0}, \mathsf{1}, x_2, \mathsf{0}, \mathsf{1}, x_3, y) \to \mathsf{F}(x_1, x_1, x_1, x_2, x_2, x_2, x_3, x_3, x_3, \mathsf{g}(x_1, x_2, x_3))$$

and let $\mathcal{R}$ consist of the following ten rules:

$$\begin{array}{rcl@{\qquad\qquad}rcl}
\mathsf{a} & \to & \mathsf{0} & \mathsf{g}(x, x, y) & \to & \mathsf{h}(x, x) \\
\mathsf{a} & \to & \mathsf{1} & \mathsf{g}(x, y, x) & \to & \mathsf{h}(x, x) \\
\mathsf{b} & \to & \mathsf{0} & \mathsf{g}(y, x, x) & \to & \mathsf{h}(x, x) \\
\mathsf{b} & \to & \mathsf{1} & \mathsf{h}(\mathsf{0}, \mathsf{1}) & \to & \mathsf{h}(\mathsf{0}, \mathsf{1}) \\
\mathsf{d} & \to & \mathsf{0} & & & \\
\mathsf{d} & \to & \mathsf{1} & & &
\end{array}$$

As in the previous example there is an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain. We choose the substitution σ with $\sigma(x_1) = \mathsf{a}$, $\sigma(x_2) = \mathsf{b}$, $\sigma(x_3) = \mathsf{d}$, and $\sigma(y) = \mathsf{g}(\mathsf{a}, \mathsf{b}, \mathsf{d})$. Then the instantiated right-hand side of the pair in $\mathcal{P}$ can be reduced to its instantiated left-hand side:

$$\begin{array}{rl}
 & \mathsf{F}(x_1, x_1, x_1, x_2, x_2, x_2, x_3, x_3, x_3, \mathsf{g}(x_1, x_2, x_3))\sigma \\
= & \mathsf{F}(\mathsf{a}, \mathsf{a}, \mathsf{a}, \mathsf{b}, \mathsf{b}, \mathsf{b}, \mathsf{d}, \mathsf{d}, \mathsf{d}, \mathsf{g}(\mathsf{a}, \mathsf{b}, \mathsf{d})) \\
\to_{\mathcal{R}}^{6} & \mathsf{F}(\mathsf{0}, \mathsf{1}, \mathsf{a}, \mathsf{0}, \mathsf{1}, \mathsf{b}, \mathsf{0}, \mathsf{1}, \mathsf{d}, \mathsf{g}(\mathsf{a}, \mathsf{b}, \mathsf{d})) \\
= & \mathsf{F}(\mathsf{0}, \mathsf{1}, x_1, \mathsf{0}, \mathsf{1}, x_2, \mathsf{0}, \mathsf{1}, x_3, y)\sigma
\end{array}$$

This chain is minimal since the subterm $\mathsf{g}(\mathsf{a}, \mathsf{b}, \mathsf{d})$ of the instantiated right-hand side is terminating. The reason is that this subterm can be reduced to $\mathsf{h}(\mathsf{0}, \mathsf{0})$ or to $\mathsf{h}(\mathsf{1}, \mathsf{1})$ but not to $\mathsf{h}(\mathsf{0}, \mathsf{1})$. Thus, the DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ is not finite.

But for $\mathcal{N} = \mathcal{N}(\mathcal{P}, \mathcal{Q}, \mathcal{R})$ the DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{N} \cup \mathcal{C}_\varepsilon, \mathbf{m})$ is finite and therefore, any processor which transforms $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ into $(\mathcal{P}, \mathcal{Q}, \mathcal{N} \cup \mathcal{C}_\varepsilon, \mathbf{m})$ is unsound. Here, the needed rules $\mathcal{N}$ consist of all g-rules and the h-rule, As before, we prove that there is no infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{N} \cup \mathcal{C}_\varepsilon)$-chain.

In any chain of length greater than 1 we must use a substitution σ such that all $x_i\sigma$ can be reduced to both $\mathsf{0}$ and $\mathsf{1}$ with $\mathcal{N} \cup \mathcal{C}_\varepsilon$. As in Example 4.15 each $x_i\sigma$ must be reducible to $\mathsf{c}(\mathsf{0}, \mathsf{1})$ or to $\mathsf{c}(\mathsf{1}, \mathsf{0})$. So there are at least two $x_i\sigma$ and $x_j\sigma$ with $i \neq j$ which can be reduced to the same term $\mathsf{c}(\mathsf{0}, \mathsf{1})$ or $\mathsf{c}(\mathsf{1}, \mathsf{0})$. Hence, the subterm $\mathsf{g}(x_1, x_2, x_3)\sigma$ of $\mathcal{P}$'s instantiated right-hand side can be reduced to $\mathsf{h}(\mathsf{c}(\mathsf{0}, \mathsf{1}), \mathsf{c}(\mathsf{0}, \mathsf{1}))$ or to $\mathsf{h}(\mathsf{c}(\mathsf{1}, \mathsf{0}), \mathsf{c}(\mathsf{1}, \mathsf{0}))$ and further to the non-terminating term $\mathsf{h}(\mathsf{0}, \mathsf{1})$.

Example 4.17. One might argue that the previous counterexamples do not show that really something can go wrong. If one starts with a TRS including the non-terminating rule $\mathsf{h}(0,1) \to \mathsf{h}(0,1)$ then one will always get a dependency pair $\mathsf{H}(0,1) \to \mathsf{H}(0,1)$ and the resulting DP problem cannot be solved by a wrong application of Theorem 4.12.

However, in the following example a wrong application would transform the *only* non-finite DP problem into a finite one. To demonstrate this effect the g-rules and h-rules of Example 4.16 have to be combined. We consider the following TRS $\mathcal{R}$ where $r = \mathsf{g}(x_1,x_1,x_1,x_1,x_2,x_2,x_2,x_2,x_3,x_3,x_3,x_3,x_1,x_2,x_3,y,y)$.

$$\begin{array}{lcl@{\qquad}lcl}
\mathsf{a} & \to & 0 & \mathsf{f}(0,1,2,x_1,0,1,3,x_2,0,1,4,x_3,z) & & \to \\
\mathsf{a} & \to & 1 & \multicolumn{3}{l}{\mathsf{f}(x_1,x_1,x_1,x_1,x_2,x_2,x_2,x_2,x_3,x_3,x_3,x_3,r[y/x_1])} \\
\mathsf{a} & \to & 2 & & & \\
\mathsf{b} & \to & 0 & \mathsf{g}(0,1,2,x_1,0,1,3,x_2,0,1,4,x_3,y,y,z,0,1) & \to & r \\
\mathsf{b} & \to & 1 & \mathsf{g}(0,1,2,x_1,0,1,3,x_2,0,1,4,x_3,y,z,y,0,1) & \to & r \\
\mathsf{b} & \to & 3 & \mathsf{g}(0,1,2,x_1,0,1,3,x_2,0,1,4,x_3,z,y,y,0,1) & \to & r \\
\mathsf{d} & \to & 0 & & & \\
\mathsf{d} & \to & 1 & & & \\
\mathsf{d} & \to & 4 & & &
\end{array}$$

First note that the TRS without the f-rule is terminating. For an infinite reduction with one of the g-rules we obviously must instantiate x_1 by a, x_2 by b, and x_3 by d. However, the only possibility to join any two terms of a, b, and d is to rewrite them to either 0 or 1. Hence, y must be instantiated by 0 or 1. In any case the instance of y cannot match both 0 and 1. Thus, one cannot apply any g-rule more than once.

Now we consider the whole TRS $\mathcal{R}$. As argued before the only possible non-finite DP problem is the one containing the dependency pair $\mathsf{F}(\dots) \to \mathsf{F}(\dots)$ for the f-rule. Similarly to Example 4.16 there is an infinite chain when instantiating x_1 by a, x_2 by b, x_3 by d, and z by $\mathsf{g}(\mathsf{a},\mathsf{a},\mathsf{a},\mathsf{a},\mathsf{b},\mathsf{b},\mathsf{b},\mathsf{b},\mathsf{d},\mathsf{d},\mathsf{d},\mathsf{d},\mathsf{a},\mathsf{b},\mathsf{d},\mathsf{a},\mathsf{a})$. Note that this is also a minimal chain as the TRS without the f-rule is terminating.

The needed rules $\mathcal{N}$ are just the three g-rules. However, there is no infinite minimal chain if we consider the TRS $\mathcal{N} \cup \mathcal{C}_\varepsilon$ instead of $\mathcal{R}$. As in Example 4.16 we conclude that in any chain of length greater than one we must use a substitution σ where each of the terms $x_i\sigma$ must be reducible to $\mathsf{c}(0,1)$ or $\mathsf{c}(1,0)$. Moreover, $x_1\sigma$ must be reducible to 2, $x_2\sigma$ to 3, and $x_3\sigma$ to 4. We only consider the case that $x_2\sigma$ and $x_3\sigma$ are both reducible to $\mathsf{c}(0,1)$ to show that the term $r[y/x_1]\sigma$ is non-terminating. The remaining cases are completely analogous.

$$\begin{array}{ll}
 & r[y/x_1]\sigma \\
= & \mathsf{g}(x_1,x_1,x_1,x_1,x_2,x_2,x_2,x_2,x_3,x_3,x_3,x_3,x_1,x_2,x_3,x_1,x_1)\sigma \\
\to^*_{\mathcal{N}\cup\mathcal{C}_\varepsilon} & \mathsf{g}(x_1,x_1,x_1,x_1,x_2,x_2,x_2,x_2,x_3,x_3,x_3,x_3,x_1,x_2,x_3,0,1)\sigma = t \\
\to^*_{\mathcal{N}\cup\mathcal{C}_\varepsilon} & \mathsf{g}(0,1,2,x_1,0,1,3,x_2,0,1,4,x_3,x_1,x_2,x_3,0,1)\sigma \\
\to^*_{\mathcal{N}\cup\mathcal{C}_\varepsilon} & \mathsf{g}(0,1,2,x_1,0,1,3,x_2,0,1,4,x_3,x_1,\mathsf{c}(0,1),\mathsf{c}(0,1),0,1)\sigma \\
\to_{\mathcal{N}} & \mathsf{g}(x_1,x_1,x_1,x_1,x_2,x_2,x_2,x_2,x_3,x_3,x_3,x_3,x_1,x_2,x_3,\mathsf{c}(0,1),\mathsf{c}(0,1))\sigma \\
\to^2_{\mathcal{C}_\varepsilon} & \mathsf{g}(x_1,x_1,x_1,x_1,x_2,x_2,x_2,x_2,x_3,x_3,x_3,x_3,x_1,x_2,x_3,0,1)\sigma = t
\end{array}$$

Thus, transforming the non-finite DP problem $(\{\mathsf{F}(\dots) \to \mathsf{F}(\dots)\}, \varnothing, \mathcal{R}, \mathbf{m})$ into the problem $(\{\mathsf{F}(\dots) \to \mathsf{F}(\dots)\}, \varnothing, \mathcal{N} \cup \mathcal{C}_\varepsilon, \mathbf{m})$ is unsound. This is especially critical as $\mathsf{F}(\dots) \to \mathsf{F}(\dots)$ is the only dependency pair of the non-terminating TRS $\mathcal{R}$ that can be used to build infinite chains.

We shortly recapitulate the situation. An application of the needed rules processor in Theorem 4.12 can simplify the constraints generated by the reduction pair processor of Theorem 4.2 considerably. However, it does not preserve the minimality flag and it is incomplete. It was explained that both of these effects are not desirable at all and examples were given to show that Theorem 4.12 cannot be improved.

As the main use of the needed rules processor is to reduce the number of the constraints for the reduction pair processor we now introduce the second processor based on Lemma 4.11 (viii) which combines the idea of needed rules into the reduction pair processor. The idea is to just satisfy the constraints for the needed rules and to delete all strictly decreasing pairs in the resulting DP problem. However, the TRS in the resulting TRS problem will still be the whole TRS. It turns out that with this idea we obtain a complete processor which does not lose minimality.

Theorem 4.18 (Processors Based on Reduction Pairs and Needed Rules)**.** *Let* $(\succsim, \succ)$ *be a reduction pair where* $\succsim$ *is* $\mathcal{C}_\varepsilon$*-compatible, and let* π *be an argument filter. Let ECap be an arbitrary estimated Cap-function that is used to compute the needed rules in Definition 4.5. Then the following processor Proc is sound and complete. For a DP problem* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$*, Proc returns*

- $\{(\mathcal{P} \setminus \succ_\pi, \mathcal{Q}, \mathcal{R}, f)\}$*, if*
  - $\mathcal{P} \subseteq \succ_\pi \cup \succsim_\pi$*,*
  - $\mathcal{N}(\mathcal{P}, \mathcal{Q}, \mathcal{R}) \subseteq \succsim_\pi$*, and*
  - $f = \mathbf{m}$.
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$*, otherwise*

Note that in contrast to the needed rules processor of Theorem 4.12 we do not use a superset $\mathcal{M}$ of the needed rules. The reason is that whenever we satisfy the constraints for $\mathcal{M} \supseteq \mathcal{N}(\mathcal{P}, \mathcal{Q}, \mathcal{R})$ then the constraints of Theorem 4.18 are also satisfied.

To compare the sequential application of the needed rules processor and the reduction pair processor of Theorem 4.2 with the new reduction pair processor of Theorem 4.18 we use the following abstract example.

Example 4.19. Let $\mathcal{P} = \{\mathsf{F}(\dots) \to \mathsf{F}(\dots, \mathsf{g}_1(\dots), \dots), \mathsf{F}(\dots) \to \mathsf{F}(\dots, \mathsf{g}_2(\dots), \dots)\}$, let $\mathcal{Q} \neq \varnothing$, and let $\mathcal{R} = \{\mathsf{g}_1(\dots) \to \dots, \mathsf{g}_2(\dots) \to \dots, \mathsf{h}(\dots) \to \dots\}$. Here, it might be the case that it is hard to satisfy the constraint for the h-rule. The needed rules of the DP problem are $\mathcal{N}_1 = \{\mathsf{g}_1(\dots) \to \dots, \mathsf{g}_2(\dots) \to \dots\}$ which do not contain the h-rule. Thus, the needed rule processor can transforms $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ to $(\mathcal{P}, \mathcal{Q}, \mathcal{N}_1 \cup \mathcal{C}_\varepsilon, \mathbf{a})$. Now as the h-rule is missing it might be possible to solve the constraints of the reduction pair processor of Theorem 4.2 and to delete the first pair of $\mathcal{P}$. Then we are left with the DP problem $(\{\mathsf{F}(\dots) \to \mathsf{F}(\dots, \mathsf{g}_2(\dots), \dots)\}, \mathcal{Q}, \mathcal{N}_1 \cup \mathcal{C}_\varepsilon, \mathbf{a})$ which has the needed rules $\mathcal{N}_2 = \{\mathsf{g}_2(\dots) \to \dots\}$. At this point we are not able to reapply the needed rules processor to delete the now unneeded g_1-rule. Hence, in another application of the reduction pair processor of Theorem 4.2 we will have to satisfy the constraint $\mathsf{g}_1(\dots) \succsim_\pi \dots$ for the unneeded g_1-rule.

This is in contrast to the new reduction pair processor of Theorem 4.18. Here, we have to satisfy the same constraints as above to delete the first rule of $\mathcal{P}$. But the result is the DP problem $(\{\mathsf{F}(\dots) \to \mathsf{F}(\dots, \mathsf{g}_2(\dots), \dots)\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ which has the needed rules $\mathcal{N}_2$ as above. However, in a second application of Theorem 4.18 there will be no constraint for the unneeded g_1-rule. In this aspect the new reduction pair processor is always better than the combination of the needed rules processor and the reduction pair processor.

Although the reduction pair processor always seems to be better than the needed rules processor there are some disadvantages. With Theorem 4.18 one cannot reduce the TRS $\mathcal{R}$. Removing rules of $\mathcal{R}$ might be needed for a successful application of the processor of Theorem 3.14 to switch to the innermost-case if the unneeded rules are not confluent or are overlapping with $\mathcal{P}$. And removing rules is also beneficial when using other techniques like semantic labeling, cf. Chapter 7 and Example 7.32.

For this second application the needed rules processor is sometimes required. However, after we have applied the needed rules processor minimality can be destroyed and the non-confluent rules of $\mathcal{C}_\varepsilon$ are added. Thus, the needed rules processor does not help at all to transform a DP problem into one where the processor of Theorem 3.14 to switch to innermost is applicable. For that reason, we introduce the following third and last processor which is based on Lemma 4.11 (viii). It removes unneeded rules, does not add the additional rules of $\mathcal{C}_\varepsilon$, and it keeps minimality. The removal of non-usable rules is often crucial, since these rules often block the application of other important processors, as will be shown in Example 4.23. The costs for these benefits are constraints that have to be satisfied by a *monotonic reduction pair* $(\succsim, \succ)$ which is a reduction pair where additionally $\succ$ is monotonic and $\mathcal{C}_\varepsilon$-compatible. (This ensures that the $\mathcal{C}_\varepsilon$-rules are only applied finitely often.) Thus, we cannot preprocess the constraints with an argument filter π as $\succ_\pi$ in general is not monotonic even if $\succ$ is monotonic.

Theorem 4.20 (Processors Based on Needed Rules and Reduction Pairs). *Let $(\succsim, \succ)$ be a monotonic reduction pair where $\succ$ is $\mathcal{C}_\varepsilon$-compatible. Let ECap be an estimated Cap-function that parameterizes the needed rules $\mathcal{N}$ in Definition 4.5. Then the following processor Proc is sound and complete. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, Proc returns*

- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{N}, f)\}$, *if*
  - $\mathcal{N} = \mathcal{N}(\mathcal{P}, \mathcal{Q}, \mathcal{R})$,
  - $\mathcal{P} \cup \mathcal{N} \subseteq \succsim$, *and*
  - $f = \mathbf{m}$.
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$, *otherwise*

This processor is similar to the usable rules processor of Theorem 3.25 but one additionally has to satisfy a set of constraints and one can only use it if one tries to prove absence of *minimal* chains. However, in the case that a DP problem is not in the innermost case, we usually get far less needed rules than usable rules.

Compared to the needed rules processor of Theorem 4.12 in Theorem 4.20 it is neither required to add the rules of $\mathcal{C}_\varepsilon$, nor is minimality lost. However, these benefits are paid by having to satisfy constraints which are similar to the ones of Theorem 4.18. But a major difference to Theorem 4.18 is that in Theorem 4.20 the strict order has to be monotonic which prohibits the use of argument filters. We will investigate the difference by continuing to prove termination of the running example of Example 4.13.

Example 4.21. We recapitulate some rules and pairs.

$$\begin{aligned} \mathsf{s}(\mathsf{p}(x)) &\to x && (33)\\ \mathsf{p}(\mathsf{s}(x)) &\to x && (34)\\ \mathsf{negate}(\mathsf{0}) &\to \mathsf{0} && (37)\\ \mathsf{negate}(\mathsf{p}(x)) &\to \mathsf{s}(\mathsf{negate}(x)) && (38)\\ \mathsf{negate}(\mathsf{s}(x)) &\to \mathsf{p}(\mathsf{negate}(x)) && (39)\\ \mathsf{negate}(\mathsf{negate}(x)) &\to x && (40)\\ \mathsf{negate}(\mathsf{minus}(\mathsf{0}, x)) &\to x && (41) \end{aligned}$$

$$\begin{aligned} \mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{DIV}(\mathsf{minus}(x, y), \mathsf{s}(y)) && (52)\\ \mathsf{QUOT}(x, \mathsf{p}(y)) &\to \mathsf{QUOT}(x, \mathsf{negate}(\mathsf{p}(y))) && (53)\\ \mathsf{QUOT}(\mathsf{p}(x), y) &\to \mathsf{QUOT}(\mathsf{negate}(\mathsf{p}(x)), y) && (54) \end{aligned}$$

With the reduction pair processor of Theorem 4.18 we can solve the same three examples with the same orders as we did in Example 4.13. But as before, it is not possible to simplify the remaining DP problem $(\{(53), (54)\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ using quasi-simplification orders. The reason is that the needed rule (40) enforces to use an argument filter which does not drop the argument of negate.

We obtain another result by the needed rules processor of Theorem 4.20. Here, the constraint for pair (52) is not satisfiable if $\succ$ is monotonic and if $\succsim$ is a quasi-simplification order. The reason is that we must not filter the second argument of minus due to the required monotonicity. Thus, the reduction pair processor of Theorem 4.18 can be more powerful than the needed rules processor.

However, we can simplify the remaining DP problem $(\{(53), (54)\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ by the needed rules processor of Theorem 4.20. As reduction pair we choose a polynomial order with $\mathcal{P}ol(\mathsf{c}(x, y)) = 1 + x + y$ and $\mathcal{P}ol(f(x_1, \ldots, x_n)) = x_1 + \cdots + x_n$ for all remaining symbols f. Then the requirements are satisfied and we obtain the simpler DP problem $(\{(53), (54)\}, \mathcal{Q}, \{(33), (34), (37) - (41)\}, \mathbf{m})$.

Thus, although the constraints of Theorem 4.18 and Theorem 4.20 look similar they are incomparable: Theorem 4.18 requires to have at least one strict decrease in the constraints for $\mathcal{P}$ and Theorem 4.20 requires that the strict order in the reduction pair is monotonic. However, neither of them is able to simplify our new remaining DP problem further. For this we will need processors of the remainder of this chapter.

An important observation about Theorem 4.20 has already been made in [GTS05a, Page 18]: Theorem 4.20 can always be applied if $\mathcal{P}$ and the needed rules are non-duplicating. In this case the polynomial order of Example 4.21 always satisfies the constraints. Thus, for DP problems arising from string rewrite systems one can always replace $\mathcal{R}$ by the needed rules.

Relating Theorem 4.20 to the previous version in the literature [GTS05a, Theorem 28] we see two differences. First, the new version in this thesis is more powerful, since we use an improved version of needed rules. But on the other hand, in [GTS05a] it is allowed to delete all rules and pairs that contain unneeded symbols in their left-hand side.

There are two reasons that we did not *combine* these two theorems into one large theorem which has the improved version of needed rules and the additional deletion possibility of [GTS05a, Theorem 28]. First, it suffices to apply both existing theorems *separately*.

Whenever we can satisfy the constraints of Theorem 4.20 using some reduction pair then we already can change to the improved version of needed rules. Moreover, the same reduction pair satisfies the constraints of [GTS05a, Theorem 28] for the resulting DP problem. Thus, we can use the additional deletion possibility of [GTS05a, Theorem 28]. With this sequential application we have the same effect as if we had applied a combined theorem. The second reason is that the combination would require an even more complex and hence, less understandable version of the transformation $\mathcal{I}$.

Nevertheless, for the automation of these processors it is useful to combine both theorems. Once one has found the reduction pair for Theorem 4.20 one should not forget it, but directly apply [GTS05a, Theorem 28] afterwards.

4.3. Rule Removal

Now we present the processor of [GTS05a] to remove further rules from $\mathcal{R}$. Our work extends the idea in [Zan05b, Theorems 1 and 4] to simplify string rewrite systems by repeatedly removing rules by using polynomial orders.

As in Theorem 4.2, for a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, all rules in $\mathcal{P}$ and $\mathcal{R}$ are oriented with a reduction pair $(\succsim, \succ)$. The processor in Theorem 4.2 was used to remove pairs from $\mathcal{P}$ which could be oriented with $\succ$. In contrast, the present processor removes rules from both $\mathcal{P}$ and $\mathcal{R}$ if they can be oriented with $\succ$. The disadvantage is that here we are again restricted to monotonic reduction pairs where $\succ$ is monotonic and where we may not use argument filters.

Theorem 4.22 (Processors Based on Rule Removal)**.** *Let $(\succsim, \succ)$ be a reduction pair where $\succ$ is monotonic. The following processor Proc is sound and complete. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, Proc returns*

- $\{(\mathcal{P} \setminus \succ, \mathcal{Q}, \mathcal{R} \setminus \succ, f)\}$, *if*
  - $\mathcal{P} \subseteq \succ \cup \succsim$ *and*
  - $\mathcal{R} \subseteq \succ \cup \succsim$
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$, *otherwise*

Note that in case of $\mathcal{C}_\varepsilon$-compatible reduction pairs, the constraints of Theorem 4.22 are harder to satisfy than those of Theorem 4.20. Therefore, it is always a good idea to apply the needed rules processor of Theorem 4.20 first. However, the needed rules processor is applicable once only. After it has deleted all unneeded rules, all rules in the resulting TRS are needed. Therefore, a second application of the needed rules processor will not simplify the DP problem any further. In contrast, the rule removal processor can delete any remaining rule provided that the constraints are satisfied.

We continue to prove termination of Example 4.21.

Example 4.23. As discussed at the end of Section 4.2, the remaining DP problem $(\{(53), (54)\}, \mathcal{Q}, \{(33), (34), (37) - (41)\}, \mathbf{m})$ cannot be simplified by the processors that are known at that point. However, the new rule removal processor of Theorem 4.22 is applicable. We choose the linear polynomial order with $\mathcal{P}ol(\mathsf{s}(x)) = \mathcal{P}ol(\mathsf{p}(x)) = 1 + x$ and $\mathcal{P}ol(f(x_1, \ldots, x_n)) = x_1 + \cdots + x_n$ for the remaining symbols f, i.e., $\mathcal{P}ol$ counts the number of s- and p-symbols. Then the constraints of rules (33) and (34) are strictly decreasing and we can delete these rules. We obtain the remaining DP problem

$(\{(53), (54)\}, \mathcal{Q}, \{(37) - (41)\}, \mathbf{m})$ which cannot be simplified further by our current set of processors if we only use reduction pairs based on quasi-simplification orders.

As a final remark, note that the rule removal processor cannot handle the DP problem $(\{(53), (54)\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ using a quasi-simplification order due to the division rule (43). Hence, the previous application of the needed rules processor of Theorem 4.20 was urgently required.

4.4. Usable Rules w.r.t. an Argument Filter

In the following two sections we will show how one can reduce the set of constraints of the reduction pair processors even further. For the main idea, we recapitulate the situation in Example 4.3. We had to solve the constraints

$$\begin{aligned} \mathsf{MUL}(x, y, z) &\succ_\pi \mathsf{IF}(\mathsf{isZero}(x), x, y, z) && (23) \\ \mathsf{IF}(\mathsf{false}, \mathsf{s}(x), y, z) &\succ_\pi \mathsf{MUL}(x, y, \mathsf{plus}(y, z)) && (25) \end{aligned}$$

together with the constraints for the remaining usable rules for computing isZero and plus. Note that both the test isZero and the accumulator plus do not contribute to the termination argument: we used an argument filter which only considered the first argument of MUL and the second argument of IF. Hence, if we apply the filter we do not see the function symbols isZero and plus any more. So the question is why we should have to consider the isZero- and plus-rules as usable rules for this specific argument filter.

Thus, the idea is to make usable rules dependent on the argument filter to obtain less usable rules. To this end, we introduce the notion of regarded positions and make minor adjustments to Definition 3.24. After an example we also refine the improved estimation of usable rules in Definition 3.26 in order to respect the argument filter.

Definition 4.24 (Regarded Positions)**.** *Let π be an argument filter. Then for every term we define its* regarded positions *w.r.t. π as the smallest set $RegPos_\pi(t)$ such that*

- *$\varepsilon \in RegPos_\pi(t)$ and*
- *if $t = f(t_1, \ldots, t_n)$, $p \in RegPos_\pi(t_i)$, and $\pi(f) = i$ or $\pi(f) = [\ldots, i, \ldots]$ then $ip \in RegPos_\pi(t)$.*

So in essence, the regarded positions of a term t are all those positions of t that are not dropped by the filter π. Now we can define the usable rules w.r.t. an argument filter.

Definition 4.25 (Usable Rules w.r.t. an Argument Filter)**.** *Let $\mathcal{Q}$ and $\mathcal{S}$ be sets of terms, let $\mathcal{R}$ be a TRS, and let π be an argument filter. We define the usable rules w.r.t. π of a term t as the smallest subset $\mathcal{U}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$ of $\mathcal{R}$ such that whenever there is a substitution σ such that $\mathcal{S}\sigma \subseteq NF(\mathcal{Q})$ and $t\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} v \xrightarrow{\mathcal{Q}}_{\ell \to r, p} u$ for some rule $\ell \to r \in \mathcal{R}$ and position $p \in RegPos_\pi(v)$ then $\ell \to r \in \mathcal{U}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$.*

A function $\mathcal{EU}$ estimates *the usable rules w.r.t. π iff $\mathcal{U}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{EU}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$ for all possible inputs $\mathcal{R}$, $\mathcal{Q}$, $\mathcal{S}$, t, and π. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ the estimated usable rules w.r.t. π are defined as $\mathcal{EU}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \pi) = \bigcup_{s \to t \in \mathcal{P}} \mathcal{EU}^{\{s\},\pi}_{\mathcal{R},\mathcal{Q}}(t)$.*

Using this definition one immediately obtains the following lemma.

Lemma 4.26 (Properties of Usable Rules w.r.t. to an Argument Filter)**.** *Let π be an argument filter and let $\mathcal{EU}$ be an estimation of the usable rules w.r.t. π. If $s_1 \to t_1, s_2 \to t_2, \ldots$ is an infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain then $\pi(s_1) \to \pi(t_1), \pi(s_2) \to \pi(t_2), \ldots$ is an infinite $(\pi(\mathcal{P}), \varnothing, \pi(\mathcal{EU}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \pi)))$-chain.*[19]

Now we can improve the reduction pair processor of Theorem 4.2 considerably by integrating the argument filter.

Theorem 4.27 (Processors Based on Reduction Pairs and Usable Rules w.r.t. an Argument Filter)**.** *Let $(\succsim, \succ)$ be a reduction pair and π be an argument filter. Let $\mathcal{EU}$ estimate the usable rules w.r.t. π. Then the following processor Proc is sound and complete. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, Proc returns*

- $\{(\mathcal{P} \setminus \succ_\pi, \mathcal{Q}, \mathcal{R}, f)\}$, *if*
  - $\mathcal{P} \subseteq \succ_\pi \cup \succsim_\pi$ *and*
  - $\mathcal{EU}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \pi) \subseteq \succsim_\pi$.
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$, *otherwise*

Example 4.28. We now show that the DP problem $\mathcal{D}_6$ with pairs (23) and (25) of Example 4.3 can even be solved by the embedding order.

$$\mathsf{MUL}(x, y, z) \succsim \mathsf{IF}(\mathsf{isZero}(x), x, y, z) \tag{23}$$
$$\mathsf{IF}(\mathsf{false}, \mathsf{s}(x), y, z) \succ \mathsf{MUL}(x, y, \mathsf{plus}(y, z)) \tag{25}$$

We choose $\pi(\mathsf{MUL}) = 1$, $\pi(\mathsf{IF}) = 2$, and $\pi(\mathsf{s}) = [1]$. Then there are no usable rules w.r.t. π since π drops both subterms $\mathsf{isZero}(x)$ and $\mathsf{plus}(y, z)$ which contain defined symbols. Thus, the constraints for the DP problem are simplified to $x \succsim x$ and $\mathsf{s}(x) \succ x$. These are obviously satisfied by the embedding order. Hence, by Theorem 4.27 we can delete (23) from $\mathcal{P}$. The remaining DP problem can then by solved by the processors based on the dependency graph.

Note that Theorem 4.27 does neither help in solving the other DP problem $\mathcal{D}_1$ of Example 4.3 nor does it help for the remaining DP problem Example 4.23. As both DP problems are not in the innermost case, all rules are usable w.r.t. π for any filter with $\pi(\mathsf{PLUS}) \neq []$ or $\pi(\mathsf{QUOT}) \neq []$, respectively. Thus, it is strongly required to carry over the idea of usable rules w.r.t. an argument filter to define the needed rules w.r.t. an argument filter. However, the needed rules in Definition 4.5 are based on the approximation of usable rules where one only dropped the variable case. So, before we define the needed rules w.r.t. an argument filter in the next section, let us adapt the improved estimation of usable rules of Definition 3.26 to regard a given argument filter. This is also required to mechanize Theorem 4.27 as the usable rules w.r.t. an argument filter are – like the usable rules – not computable. To obtain the following definition one just has to replace condition (*ii*) in Definition 3.26 accordingly.

Definition 4.29 (Improved Estimated Usable Rules w.r.t. an Argument Filter)**.** *Let $\mathcal{Q}$ and $\mathcal{S}$ be set of terms, let $\mathcal{R}$ be a TRS, and let π be an argument filter. Let ECap be an estimated Cap-function. The* improved estimated usable rules *of a term t w.r.t. π are defined as the smallest set $\mathcal{IU}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{R}$ such that*

---

[19] Note that we introduced chains only for DP problems where $\mathcal{P}$ is a set of pairs (or a graph over pairs) and where $\mathcal{R}$ is a TRS. However, in general $\pi(\mathcal{P})$ and $\pi(\mathcal{EU}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \pi))$ are only generalized TRSs. But this is no problem as one can easily extend the definition of a chain to generalized TRSs.

(i) *If* $t = f(t_1, \ldots, t_n)$, $\ell = f(\ell_1, \ldots, \ell_n) \to r \in \mathcal{R}$, *and if the terms* $f(\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1), \ldots, \mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n))$ *and* ℓ *are unifiable with the mgu* δ *such that all terms in* $(\{\ell_1, \ldots, \ell_n\} \cup \mathcal{S})\delta$ *are in* $\mathcal{Q}$*-normal form, then* $\ell \to r \in \mathcal{IU}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$.

(ii) *If* $t = f(t_1, \ldots, t_n)$ *and* $i \in \mathit{RegPos}_\pi(t)$ *then* $\mathcal{IU}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t_i) \subseteq \mathcal{IU}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$.

(iii) *If* $\ell = f(\ell_1, \ldots, \ell_n) \to r \in \mathcal{IU}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$ *then* $\mathcal{IU}^{\{\ell_1,\ldots,\ell_n\},\pi}_{\mathcal{R},\mathcal{Q}}(r) \subseteq \mathcal{IU}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$.

(iv) *If* $t = x$ *and* x *is not a subterm of any term in* $\mathcal{S}$ *or* $\mathit{NF}(\mathcal{Q}) \not\subseteq \mathit{NF}(\mathcal{R})$ *then* $\mathcal{IU}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t) = \mathcal{R}$.

The following theorem states the desired result that indeed the improved estimated usable rules w.r.t. π estimate the usable rules w.r.t. π.

Theorem 4.30 (Soundness of the Improved Usable Rules w.r.t. an Argument Filter)**.** *The function* $\mathcal{IU}$ *to compute the improved estimated usable rules w.r.t. an argument filter estimates* $\mathcal{U}$*.*

Comparing the usable rules w.r.t. an argument filter with previous approaches in the literature [AG00, GTS05b, GTSF06, TGS04], we see that only our approach uses unification, an arbitrary *Cap*-function, the argument filter, and checks on $\mathcal{Q}$-normal forms to estimate the usable rules w.r.t. an argument filter. Even a combination of unification and argument filters has not been done before. All previous approaches essentially only compare the root symbols [AG00, GTSF06, TGS04] or do not integrate the argument filter [AG00, GTS05b]. That sometimes both is required is shown in our running example (Example 4.33) where we use the combination of unification and argument filters for the needed rules.

4.5. Needed Rules w.r.t. an Argument Filter

In Section 3.3 we have seen that if DP problems are not in the innermost case then the set of usable rules often contains all rules. This remains valid for the usable rules w.r.t. an argument filter, but not for the needed rules, cf. Section 4.2. Now we further reduce the set of needed rules by regarding an argument filter. As in Section 4.2 one can show that only these rules have to be considered when adding the rules of the TRS $\mathcal{C}_\varepsilon$. As this result is obtained in a completely similar way as in Section 4.2, we will first present the main theorem and illustrate it before we present the necessary transformation $\mathcal{I}_\pi$ at the end of this section.

We obtain the needed rules w.r.t. an argument filter from the improved estimation of usable rules w.r.t. an argument filter in the way that we obtained the needed rules from the improved estimated rules: We drop condition (iv) for the variable case of Definition 4.29.

Definition 4.31 (Needed Rules w.r.t. an Argument Filter)**.** *Let* $\mathcal{Q}$ *and* $\mathcal{S}$ *be set of terms, let* $\mathcal{R}$ *be a TRS, let* π *be an argument filter, and let ECap be an estimated Cap-function. The* needed rules *of a term* t *w.r.t.* π *are defined as the smallest set* $\mathcal{N}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{R}$ *such that*

(i) *If* $t = f(t_1, \ldots, t_n)$, $\ell = f(\ell_1, \ldots, \ell_n) \to r \in \mathcal{R}$, *and if the terms* $f(\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1), \ldots, \mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n))$ *and* ℓ *are unifiable with the mgu* δ *such that all terms in* $(\{\ell_1, \ldots, \ell_n\} \cup \mathcal{S})\delta$ *are in* $\mathcal{Q}$*-normal form, then* $\ell \to r \in \mathcal{N}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$.

(ii) If $t = f(t_1, \dots, t_n)$ *and* $i \in \mathit{RegPos}_\pi(t)$ *then* $\mathcal{N}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t_i) \subseteq \mathcal{N}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$*.*

(iii) If $\ell = f(\ell_1, \dots, \ell_n) \to r \in \mathcal{N}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$ *then* $\mathcal{N}^{\{\ell_1,\dots,\ell_n\},\pi}_{\mathcal{R},\mathcal{Q}}(r) \subseteq \mathcal{N}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$*.*

As usual, $\mathcal{N}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \pi) = \bigcup_{s \to t \in \mathcal{P}} \mathcal{N}^{\{s\},\pi}_{\mathcal{R},\mathcal{Q}}(t)$*.*

Theorem 4.32 (Processors Based on Reduction Pairs and Needed Rules w.r.t. an Argument Filter)**.** *Let* $(\succsim, \succ)$ *be a reduction pair where* $\succsim$ *is* $\mathcal{C}_\varepsilon$*-compatible and let* π *be an argument filter. Let ECap be an estimated Cap-function that parameterizes the needed rules* $\mathcal{N}$ *in Definition 4.31. Then the following processor Proc is sound and complete. For a DP problem* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$*, Proc returns*

- $\{(\mathcal{P} \setminus \succ_\pi, \mathcal{Q}, \mathcal{R}, f)\}$*, if*
  - $\mathcal{P} \subseteq \succ_\pi \cup \succsim_\pi$*,*
  - $\mathcal{N}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \pi) \subseteq \succsim_\pi$*, and*
  - $f = \mathbf{m}$*.*
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$*, otherwise*

Note that Theorem 4.18 is subsumed by Theorem 4.32: disregarding the argument filter in the computation of needed rules will only produce more needed rules and therefore more constraints, but the other conditions to apply the reduction pair processor are identical in Theorem 4.32 and Theorem 4.18. That the processor of Theorem 4.32 is strictly more powerful is shown in the running example of this chapter.

Example 4.33. We recapitulate some rules and pairs of the remaining DP problem $(\{(53), (54)\}, \mathcal{Q}, \{(37) - (41)\}, \mathbf{m})$ of Example 4.23.

$$\mathsf{negate}(\mathsf{0}) \to \mathsf{0} \tag{37}$$
$$\mathsf{negate}(\mathsf{p}(x)) \to \mathsf{s}(\mathsf{negate}(x)) \tag{38}$$
$$\mathsf{negate}(\mathsf{s}(x)) \to \mathsf{p}(\mathsf{negate}(x)) \tag{39}$$
$$\mathsf{negate}(\mathsf{negate}(x)) \to x \tag{40}$$
$$\mathsf{negate}(\mathsf{minus}(\mathsf{0}, x)) \to x \tag{41}$$

$$\mathsf{QUOT}(x, \mathsf{p}(y)) \to \mathsf{QUOT}(x, \mathsf{negate}(\mathsf{p}(y))) \tag{53}$$
$$\mathsf{QUOT}(\mathsf{p}(x), y) \to \mathsf{QUOT}(\mathsf{negate}(\mathsf{p}(x)), y) \tag{54}$$

Consider the argument filter π with $\pi(\mathsf{QUOT}) = [1, 2]$, $\pi(\mathsf{p}) = [1]$, and $\pi(\mathsf{s}) = \pi(\mathsf{negate}) = [\,]$. For this argument filter only rule (38) is needed: since we have deleted the p-rule, $\mathit{ICap}^{\{\mathsf{QUOT}(\mathsf{p}(x),y)\}}_{\mathcal{R},\mathcal{Q}}(\mathsf{p}(x)) = \mathsf{p}(z)$. Therefore, from the pair (54) we directly see that (38) is needed, but none of the left-hand sides of the other rules is unifiable with $\mathsf{negate}(\mathsf{p}(z))$. Note that at this point previous definitions of needed rules [GTSF06, TGS04, Urb01] mark all rules as needed as they just look at the root symbols. Moreover, as the right-hand side of (38) contains the subterm $\mathsf{negate}(x)$ all rules are needed if we disregard the argument filter. But as π drops the argument of s this subterm does not have to be considered when computing the needed rules w.r.t. π. And as there is no s-rule any more, (38) remains the only needed rule w.r.t. π.

Now, the resulting constraints

$$\begin{aligned} \mathsf{QUOT}(x, \mathsf{p}(y)) &\succ \mathsf{QUOT}(x, \mathsf{negate}) && (53)\\ \mathsf{QUOT}(\mathsf{p}(x), y) &\succ \mathsf{QUOT}(\mathsf{negate}, y) && (54)\\ \mathsf{negate} &\succsim \mathsf{s} && (38) \end{aligned}$$

are easily satisfied by an LPO with precedence $\mathsf{p} > \mathsf{negate} > \mathsf{s}$. Thus, all pairs of the DP problem can be removed by the processor of Theorem 4.32 and we have finished the termination proof of our running example to compute division on integers.

To perform this proof with quasi-simplification orders for the most difficult DP problem containing the dependency pairs for the quot-rules, we required three processors that have been presented in this chapter:

(i) The needed rules processor of Theorem 4.20 deleted all unneeded rules.

(ii) Then the rule removal processor of Theorem 4.22 was able to delete the s- and p-rules.

(iii) Finally the reduction pair processor of Theorem 4.32 removed all pairs.

It can be shown that in this proof none of these three processors can be replaced by any other processor of this chapter, even if one uses larger classes of orders. For example even with negative polynomial interpretations of [HM07] one needs all three processors. In this example one can also see why losing minimality is bad. If one had started with the needed rules processor of Theorem 4.12 then it would not be possible to apply the reduction pair processor of Theorem 4.32 in the third step any more.

Note that by Theorem 4.32 it is possible to solve the DP problem $\mathcal{D}_1$ of Example 4.3 using only the embedding order. This is in contrast to the previous reduction pair processors where we had to use non-linear polynomial orders as in Example 4.3.

Now we adapt the proof of Theorem 4.18 to prove Theorem 4.32. To this end we define a transformation $\mathcal{I}_\pi$ in a similar way to $\mathcal{I}$. But $\mathcal{I}_\pi$ differs from $\mathcal{I}$ as it returns terms that are filtered with π.

Definition 4.34 ($\mathcal{I}_\pi$)**.** *Let $\mathcal{N} = \mathcal{N}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \pi)$ be the set of needed rules of the given DP problem, and let $u_1 \to v_1, u_2 \to v_2, \ldots$ be an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain. W.l.o.g. we assume that σ is the substitution used for instantiating every $u_i \to v_i$. Moreover, whenever in the reduction of $v_i\sigma$ a rule $\ell_j \to r_j \in \mathcal{R}$ is applied, then by renaming the variables in the rule we again assume that the rule is instantiated by σ in that rewrite step. Let $\mathcal{S}_{all}$ contain all u_i and all direct subterms of each ℓ_j. Let c be the new constant and let $\bot$ be the new variable which are introduced by Comp. We define the mapping $\mathcal{I}_\pi$ from terms of $\mathcal{T}(\mathcal{F}, \mathcal{V})$ that terminate w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ to terms of $\mathcal{T}(\mathcal{F} \uplus \{\mathsf{c}\}, \mathcal{V} \uplus \{\bot\})$ as follows.*

- $\mathcal{I}_\pi(x) = x$ *for every variable* x
- $\mathcal{I}_\pi(f(t_1, \ldots, t_n)) = \overline{f(t_1, \ldots, t_n)}$, *if there is no rule $\ell \to r \in \mathcal{R} \setminus \mathcal{N}$ such that $f(Cap^{\mathcal{S}_{all}\sigma}_{\mathcal{R},\mathcal{Q}}(t_1), \ldots Cap^{\mathcal{S}_{all}\sigma}_{\mathcal{R},\mathcal{Q}}(t_n))$ unifies with ℓ by some mgu μ where $(\{\ell|_1, \ldots, \ell|_n\} \cup \mathcal{S}_{all}\sigma)\mu \subseteq NF(\mathcal{Q})$.*
- $\mathcal{I}_\pi(f(t_1, \ldots, t_n)) = \mathsf{c}(\overline{f(t_1, \ldots, t_n)}, Comp(Red(f(t_1, \ldots, t_n))))$, *otherwise.*

Here, $Red_\pi(t) = \{\mathcal{I}_\pi(s) \mid t \overset{\mathcal{Q}}{\to}_{\mathcal{R}} s\}$ *and* $\overline{f(t_1,\dots,t_n)} = \mathcal{I}_\pi(t_i)$*, if* $\pi(f) = i$*, and* $\overline{f(t_1,\dots,t_n)} = f(\mathcal{I}_\pi(t_{i_1}),\dots,\mathcal{I}_\pi(t_{i_k}))$*, if* $\pi(f) = [i_1,\dots,i_k]$*.*

We extend $\mathcal{I}_\pi$ *to substitutions by defining* $\mathcal{I}_\pi(\sigma)$ *as the substitution with* $x\mathcal{I}_\pi(\sigma) = \mathcal{I}_\pi(x\sigma)$*.*

The transformation $\mathcal{I}_\pi$ has similar properties compared to $\mathcal{I}$. In correspondence to Lemma 4.11 we will show in Lemma 4.35 how to transform a minimal $(\mathcal{P},\mathcal{Q},\mathcal{R})$-chain into a $(\pi(\mathcal{P}),\varnothing,\pi(\mathcal{N}) \cup \mathcal{C}_\varepsilon)$-chain. One difference is that due to the argument filter the evaluation strategy given by $\mathcal{Q}$ cannot be preserved. This is not a severe problem for proving Theorem 4.32. since it suffices to use the filtered chain in order to show properties of the original chain.

To be more precise, if $s_1 \to t_1, s_2 \to t_2, \dots$ is an infinite minimal $(\mathcal{P},\mathcal{Q},\mathcal{R})$-chain then this chain will be transformed into the infinite filtered chain $\pi(s_1) \to \pi(t_1), \pi(s_2) \to \pi(t_2), \dots$ where only rules of $\pi(\mathcal{N}) \cup \mathcal{C}_\varepsilon$ are needed. Then the constraints of Theorem 4.32 ensure that certain pairs $\pi(s) \to \pi(t)$ in the filtered chain can only occur finitely often. But then even in the original chain the corresponding pairs $s \to t$ can occur only finitely often, and hence they can be removed from $\mathcal{P}$.

Dropping the strategy will make the proof simpler as we do not require the invariant *Inv* that every term below a c is in $\mathcal{Q}$-normal form. Moreover, as we drop the evaluation strategy we can simulate the reduction step by step and we do not have to combine multiple reduction steps which lead to a $\mathcal{Q}$-normal form. (This was required in Lemma 4.11 (vi).) On the other hand integrating the argument filter will complicate the proof a little bit.

Lemma 4.35 (Properties of $\mathcal{I}_\pi$). *Let* $\mathcal{P},\mathcal{Q},\mathcal{R},\mathcal{N}, u_1 \to v_1, u_2, \to v_2, \dots, \mathcal{S}_{all}, \sigma$ *be as in Definition 4.34. Let* t *be terminating w.r.t.* $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$*.*

(i) If $\mathcal{N}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{N}$ *and* $\mathcal{S} \subseteq \mathcal{S}_{all}$ *then* $\mathcal{I}_\pi(t\sigma) = \pi(t)\mathcal{I}_\pi(\sigma)$*.*

(ii) $\mathcal{I}_\pi(t\sigma) \to^*_{\mathcal{C}_\varepsilon} \pi(t)\mathcal{I}_\pi(\sigma)$*.*

(iii) If $t \overset{\mathcal{Q}}{\to}_{\mathcal{R}} s$*, and* $\mathcal{I}_\pi(t)$ *is built by the third case then* $\mathcal{I}_\pi(t) \to^+_{\mathcal{C}_\varepsilon} \mathcal{I}_\pi(s)$*.*

(iv) If $t \overset{\mathcal{Q}}{\to}_{\mathcal{R}} s$ *is a reduction at the root position and* $\mathcal{I}_\pi(t)$ *is built by the second case then* $\mathcal{I}_\pi(t) \to^*_{\mathcal{C}_\varepsilon}\to_{\pi(\mathcal{N})} \mathcal{I}_\pi(s)$*.*

(v) If $t \overset{\mathcal{Q}}{\to}_{\mathcal{R}} s$ *then* $\mathcal{I}_\pi(t) \to^*_{\pi(\mathcal{N})\cup\mathcal{C}_\varepsilon} \mathcal{I}_\pi(s)$*.*

(vi) $\pi(u_1 \to v_1), \pi(u_2 \to v_2), \dots$ *is a* $(\pi(\mathcal{P}),\varnothing,\pi(\mathcal{N}) \cup \mathcal{C}_\varepsilon)$*-chain.*

With the help of Lemma 4.35(vi) it is now possible to prove Theorem 4.32 in the illustrated way.

Remember that we introduced three different processors based on the transformation $\mathcal{I}$. It turned out that the reduction pair processor of Theorem 4.18 is completely subsumed by the new reduction pair processor of Theorem 4.32 which is based on $\mathcal{I}_\pi$. Hence, the obvious question is whether we can improve the other two processors which are based on $\mathcal{I}$. We answer this question in the remainder of this section.

We first consider the needed rules processor of Theorem 4.20. As this processor requires to solve constraints for $\mathcal{P} \cup \mathcal{N}(\mathcal{P},\mathcal{Q},\mathcal{R})$ it might be tempting to allow an argument filter π and just require constraints for $\mathcal{P} \cup \mathcal{N}(\mathcal{P},\mathcal{Q},\mathcal{R},\pi)$. Unfortunately, this is unsound. Consider the following counter-example with $\mathcal{P} = \{\mathsf{F}(\mathsf{a}) \to \mathsf{F}(\mathsf{b})\}$, $\mathcal{R} = \{\mathsf{b} \to \mathsf{a}\}$, and

$\mathcal{Q} = \varnothing$. If we choose the argument filter π with $\pi(\mathsf{F}) = [\,]$ then there are no needed rules w.r.t. π. Hence, the resulting constraint of the only pair in $\mathcal{P}$ can be satisfied by a monotonic $\mathcal{C}_\varepsilon$-compatible order regardless of whether we require $\pi(\mathsf{F}(\mathsf{a})) \succsim \pi(\mathsf{F}(\mathsf{b}))$ or $\mathsf{F}(\mathsf{a}) \succsim \mathsf{F}(\mathsf{b})$. The resulting DP problem where we have replaced $\mathcal{R}$ by $\varnothing$ is obviously finite. However, as $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$ is not finite, any processor which deletes the rule of $\mathcal{R}$ is unsound.

Finally, we consider a processor similar to the needed rules processor of Theorem 4.12 which directly applies Lemma 4.35 (vi) and replaces a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ by $(\pi(\mathcal{P}), \varnothing, \pi(\mathcal{N}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \pi)) \cup \mathcal{C}_\varepsilon, \mathbf{a})$, provided that the filtered pairs and rules form two TRSs. Obviously, this processor is sound, but it does not subsume Theorem 4.12. There are two reasons for this. First, the strategy $\mathcal{Q}$ is not carried over and second minimality is always lost, even if the requirements of Theorem 4.12 to carry over minimality ($\mathcal{Q} = \varnothing$ and left-linearity of $\mathcal{N}$) are satisfied. With the following two examples we show that both of the weaknesses are not due to a weakness of the transformation $\mathcal{I}_\pi$, but that any processor without these disadvantages is unsound.

To show that minimality cannot be preserved, even if $\mathcal{Q} = \varnothing$ and if $\mathcal{P} \cup \mathcal{R}$ is ground, we consider the non-finite DP problem $\mathcal{P} = \{\mathsf{F}(\mathsf{g}(\mathsf{b})) \to \mathsf{F}(\mathsf{g}(\mathsf{a}))\}$ and $\mathcal{R} = \{\mathsf{g}(\mathsf{a}) \to \mathsf{g}(\mathsf{b})\}$. If we choose the argument filter π with $\pi(\mathsf{F}) = [1]$ and $\pi(\mathsf{g}) = [\,]$ then the resulting DP problem is $(\{\mathsf{F}(\mathsf{g}) \to \mathsf{F}(\mathsf{g})\}, \varnothing, \{\mathsf{g} \to \mathsf{g}\} \cup \mathcal{C}_\varepsilon, f)$. Now, if $f = \mathbf{m}$ then this DP problem is finite since the right-hand side $\mathsf{F}(\mathsf{g})$ of the only pair is obviously not terminating w.r.t. the filtered rule $\mathsf{g} \to \mathsf{g}$.

The second example shows that it is not possible to carry over the reduction strategy given by $\mathcal{Q}$. We consider the non-finite DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ with $\mathcal{P} = \{\mathsf{F}(x) \to \mathsf{F}(\mathsf{g}(\mathsf{a}, \mathsf{s}(\mathsf{a})))\}$, $\mathcal{Q} = \{\mathsf{g}(x, x)\}$, and $\mathcal{R} = \varnothing$. If we choose $\pi(\mathsf{s}) = 1$, $\pi(\mathsf{F}) = [1]$, and $\pi(\mathsf{g}) = [1, 2]$ then we obtain the new DP problem $(\{\mathsf{F}(x) \to \mathsf{F}(\mathsf{g}(\mathsf{a}, \mathsf{a}))\}, \mathcal{Q}', \mathcal{C}_\varepsilon, \mathbf{a})$. Regardless of whether we choose $\mathcal{Q}' = \mathcal{Q}$ or $\mathcal{Q}' = \pi(\mathcal{Q})$ there is no infinite chain for this new DP problem as the right-hand side of the filtered pair cannot be reduced to a $\mathcal{Q}$-normal form.

Although minimality and strategy are lost, it can sometimes be useful to apply the processor which replaces $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$ by $(\pi(\mathcal{P}), \varnothing, \pi(\mathcal{N}) \cup \mathcal{C}_\varepsilon, \mathbf{a})$ where $\mathcal{N} = \mathcal{N}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \pi)$. In this way one can apply techniques on arbitrary DP problems which are only available for string rewriting. If one chooses π in such a way that for every function symbol at most one argument is kept, then $(\pi(\mathcal{P}), \varnothing, \pi(\mathcal{N}) \cup \mathcal{C}_\varepsilon, \mathbf{a})$ is a DP problem where every function symbol except c has at most arity 1. Since c does not occur in $\pi(\mathcal{P}) \cup \pi(\mathcal{N})$ and since this TRS is non-duplicating we can apply Theorem 4.22 to remove the $\mathcal{C}_\varepsilon$-rules. The polynomial order with $\mathcal{P}ol(\mathsf{c}(x, y)) = 1 + x + y$ and $\mathcal{P}ol(f(x_1, \ldots, x_n)) = x_1 + \cdots + x_n$ can always be used. Hence, for the resulting DP problem $(\pi(\mathcal{P}), \varnothing, \pi(\mathcal{N}), \mathbf{a})$ all symbols have at most arity 1. And to remove all constants one can use the technique of [TZGS07] such that one obtains DP problems with only unary symbols. Hence, the resulting DP problem consists of string rewrite systems.

The only problem is the automation of this approach. Since one has to perform a complete termination proof of the resulting "string DP problem", and since argument filtering is inherently incomplete, it might be a problem to detect a suitable argument filter from the exponentially large set of all possible argument filters.

4.6. Subterm Criterion

With the techniques of usable and needed rules w.r.t. an argument filter in the previous two sections it is possible to reduce the number of constraints for the rewrite system $\mathcal{R}$ in a reduction pair processor considerably. However, sometimes even these sets can get large and make the search for a suitable reduction pair in Theorem 4.27 and Theorem 4.32 hard. In [HM07] Hirokawa and Middeldorp have shown that by using a special class of orders, one does not need to consider $\mathcal{R}$ at all. Using their *subterm criterion* no constraints are generated for $\mathcal{R}$ but the relations $\succ$ and $\succsim$ are fixed to the proper and the non-proper subterm relations, respectively. Moreover, the argument filter is also restricted. For tuple symbols – the symbols that have been introduced when computing the DPs of a TRS – one may use collapsing argument filters and for every other symbol f of arity n the argument filter of f is fixed to $[1, \dots, n]$.

Here we encounter a problem in the DP framework. In a DP problem all pairs in $\mathcal{P}$ are arbitrary rules which are possibly collapsing and there is no notion of tuple symbols. The following list contains some of the characteristics of tuple symbols in the initial DP problem $DP(\mathcal{R})$.

- Every pair in $DP(\mathcal{R})$ is of the form $f(\dots) \to g(\dots)$ where f and g are tuple symbols.
- Tuple symbols do not occur in $\mathcal{R}$.
- Tuple symbols only occur at the roots of pairs in $\mathcal{P}$.

In order to obtain a subterm criterion for the DP framework we introduce a new notion of *head symbols* to describe which symbols may be filtered in a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$ if $\mathcal{P}$ is a set of arbitrary rules. Particularly, if $\mathcal{P} = DP(\mathcal{R})$ for a TRS $\mathcal{R}$, then the head symbols are exactly the tuple symbols.

Definition 4.36 (Head Symbol)**.** *For any DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$, we define its head symbols $\mathcal{H}(\mathcal{P}, \mathcal{R})$ as the set of all those function symbols on root positions in $\mathcal{P}$ which do not occur below the root in $\mathcal{P}$ and which do not occur in $\mathcal{R}$ at all.*

$$\mathcal{H}(\mathcal{P}, \mathcal{R}) = \mathcal{F}(\mathcal{P}) \setminus (\mathcal{F}(\mathcal{R}) \cup \mathcal{F}_{>\varepsilon}(\mathcal{P}))$$

Here, $\mathcal{F}(\mathcal{P})$ and $\mathcal{F}(\mathcal{R})$ denote all function symbols occurring in $\mathcal{P}$ and $\mathcal{R}$, respectively, and $\mathcal{F}_{>\varepsilon}(\mathcal{P})$ denotes all function symbols occurring in $\mathcal{P}$ below the root position.

What makes head symbols interesting is that in every infinite chain, eventually for all connections between $s \to t$ and the following pair $u \to v$ either both t and u have head symbols as root, or both do not. This property is formulated in the following lemma and is needed to prove the subterm criterion that works with head symbols instead of tuple symbols. Moreover, the lemma can be used for a processor that deletes all edges between $s \to t$ and $u \to v$ if exactly one of the terms t and u has a head symbol as root. Finally, it is used to prove that an *argument filter* processor preserves both minimality and strategy if only head symbols are filtered. This is contrast to the argument filter processor of [GTS05a, Theorem 37]. There it was necessary to drop both minimality and strategy, since arbitrary filters where allowed in the argument filter processor. However, it was already indicated that one can define an argument filter processor as we do in the upcoming Theorem 4.38, but the details were omitted.

Lemma 4.37 (Properties of Head Symbols)**.** *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ be a DP problem and let $\mathcal{H} = \mathcal{H}(\mathcal{P}, \mathcal{R})$ be its head symbols. Then in every infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain $s_1 \to t_1, s_2 \to t_2, \dots$ there is an $n \in \mathbb{N}$ such that for every $i \geq n$ the equivalence $root(t_i) \in \mathcal{H} \Leftrightarrow root(s_{i+1}) \in \mathcal{H}$ is satisfied.*[20]

---

[20] Note that the root of a variable is not defined. Nevertheless, we say that $root(t_i)$ is not contained in $\mathcal{H}$ whenever t_i is a variable.

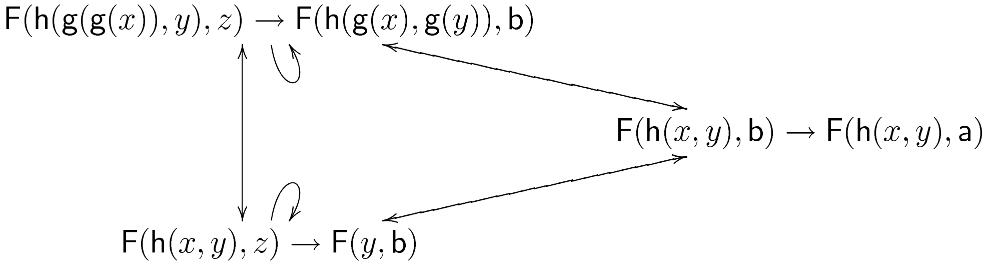
Theorem 4.38 (Processors Based on Argument Filters)**.** *Let π be an argument filter. The following processor Proc is sound. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ with $\mathcal{H} = \mathcal{H}(\mathcal{P}, \mathcal{R})$, Proc returns*

- $\{(\pi(\mathcal{P}), \mathcal{Q}, \mathcal{R}, f)\}$, *if $\pi(\mathcal{P})$ is a TRS and $\pi(f) = [1, \ldots, ar(f)]$ for all $f \notin \mathcal{H}$*
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$, *otherwise*

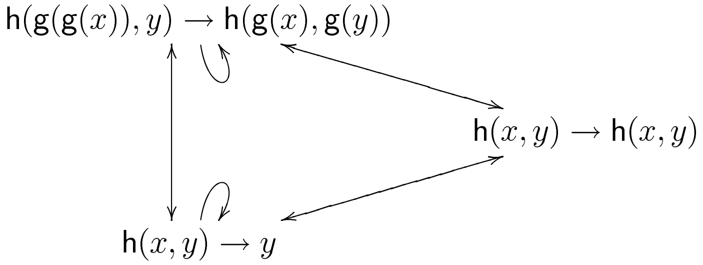
Theorem 4.39 (Edge Deletion by Head Symbols Processor)**.** *The following processor Proc is sound and complete. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ with $\mathcal{P} = (N, E)$ and $\mathcal{H} = \mathcal{H}(\mathcal{P}, \mathcal{R})$, Proc returns $\{((N, E'), \mathcal{Q}, \mathcal{R}, f)\}$ where $E' = \{(s \to t, u \to v) \in E \mid root(t) \in \mathcal{H} \Leftrightarrow root(u) \in \mathcal{H}\}$.*

Note that Theorem 4.39 is not subsumed by the edge deletion processor of Theorem 3.3 which is based on the dependency graph. The reason is that this processor can delete edges that are connected in the dependency graph. We use a small example to illustrate the previous two processors.

Example 4.40. Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ be given with $\mathcal{Q} = \varnothing$, $\mathcal{R} = \{\mathsf{g}(x) \to \ldots\}$ where h does not occur in $\mathcal{R}$, and where $\mathcal{P}$ is the following pair-graph.


$\mathsf{F}(\mathsf{h}(\mathsf{g}(\mathsf{g}(x)), y), z) \to \mathsf{F}(\mathsf{h}(\mathsf{g}(x), \mathsf{g}(y)), \mathsf{b})$
$\mathsf{F}(\mathsf{h}(x, y), \mathsf{b}) \to \mathsf{F}(\mathsf{h}(x, y), \mathsf{a})$
$\mathsf{F}(\mathsf{h}(x, y), z) \to \mathsf{F}(y, \mathsf{b})$

Since F is a head symbol, the argument filter processor of Theorem 4.38 is applicable with $\pi(\mathsf{F}) = 1$. The resulting DP problem has the following pair-graph.


$\mathsf{h}(\mathsf{g}(\mathsf{g}(x)), y) \to \mathsf{h}(\mathsf{g}(x), \mathsf{g}(y))$
$\mathsf{h}(x, y) \to \mathsf{h}(x, y)$
$\mathsf{h}(x, y) \to y$

Note that only due to carrying over the graph structure we have not introduced an infinite chain, as otherwise there clearly would be an infinite chain using only the pair $\mathsf{h}(x, y) \to \mathsf{h}(x, y)$.

For this new DP problem the only head symbol is h. Hence, we can now apply Theorem 4.39 to delete all outgoing edges of the pair $\mathsf{h}(x, y) \to y$ and then afterwards delete that pair itself by Theorem 3.4. Note that we cannot exchange Theorem 4.39 by Theorem 3.3, since the dependency graph still contains all outgoing edges of that pair. We show how to finish the termination proof later in this section in Example 4.42.

Further uses of the argument filter processor are demonstrated in Example 6.25, in Example 7.32, and in [GTS05a, Example 41]. Moreover, [GTS05a, Examples 38–40] illustrate why it is only allowed to filter the head symbols of a DP problem, and why the argument filter processor is incomplete.

Coming back to the main aim of this section we can now formulate and prove the following generalized theorem about the subterm criterion of [HM07]. It can be applied for arbitrary DP problems and it does not necessarily require minimality as in [HM07].

Theorem 4.41 (Processors Based on the Subterm Criterion). *Let π be an argument filter. Then the following processor Proc is sound and complete. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, Proc returns*

- $\{(\mathcal{P} \setminus \rhd_\pi, \mathcal{Q}, \mathcal{R}, f)\}$, *if*
  - $\mathcal{P} \subseteq \unrhd_\pi$,
  - $\pi(g) \in \{1, \ldots, ar(g)\}$ *for all* $g \in \mathcal{H}(\mathcal{P}, \mathcal{R})$,
  - $\pi(g) = [1, \ldots, ar(g)]$ *for all* $g \notin \mathcal{H}(\mathcal{P}, \mathcal{R})$, *and*
  - $f = \mathbf{m}$ *or* $NF(\mathcal{Q}) \subseteq NF(\mathcal{R})$.
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$, *otherwise*

Note that the subterm criterion is easy to automate and it is able to prove termination of every TRS in primitive recursive form. However, for more complex DP problems like the DP problems containing dependency pairs of the div- and quot-rules in Example 4.4, often more powerful methods like the reduction pair processors of Theorems 4.27 and 4.32 are needed. Note that the proof of the subterm processor reveals that in the innermost case, Theorem 4.27 completely subsumes the subterm criterion even if one does not use the precise usable rules but if one estimates the usable rules by Definitions 3.11 and 3.26. As the needed rules are identical to the improved estimated usable rules in the innermost case we also know that the reduction pair processor of Theorem 4.32 encompasses the subterm processor.

Even if one is not in the innermost case, Theorem 4.32 is more powerful than the subterm processor for a large class of DP problems, namely those DP problems where no left-hand sides of $\mathcal{P}$ contains a defined symbols of $\mathcal{R}$. This includes all DP problems that arise from functional programs. As in the proof of Theorem 4.41 one can show that for these DP problems there are no needed rules w.r.t. the argument filter whenever the constraints of the subterm processor are satisfied. However, continuing the proof of Example 4.40 in Example 4.42 demonstrates that there are DP problems where the subterm processor is applicable but the reduction pair processor of Theorem 4.32 is not.

Example 4.42. For the remaining pairs in Example 4.40, the g-rule and possibly other rules that depend on g are needed. The corresponding constraints may prohibit a successful application of Theorem 4.32. However, we can choose $\pi(\mathsf{h}) = 1$ and delete the pair $\mathsf{h}(\mathsf{g}(\mathsf{g}(x)), y) \to \mathsf{h}(\mathsf{g}(x), \mathsf{g}(y))$ by the subterm processor. A final application of Theorem 3.4 finishes the proof.

Here, the preprocessing with the argument filter processor was required for the success of the subterm processor, since none of arguments $\mathsf{h}(\mathsf{g}(x), \mathsf{g}(y))$ and b of the right-hand side $\mathsf{F}(\mathsf{h}(\mathsf{g}(x), \mathsf{g}(y)), \mathsf{b})$ is a subterm of the corresponding left-hand side.

Additionally, the preprocessing with Theorem 4.39 was important. Without the deletion of the third pair $\mathsf{h}(x, y) \to y$, the resulting constraints would be unsatisfiable.

The following example illustrates that the subterm processor cannot be extended in a way that we replace the subterm relation by the embedding order.

Example 4.43. Let $\mathcal{P} = \{\mathsf{F}(\mathsf{g}(\mathsf{h}(\mathsf{g}(x)))) \to \mathsf{F}(\mathsf{g}(\mathsf{g}(x)))\}$, let $\mathcal{Q} = \varnothing$, and let $\mathcal{R} = \{\mathsf{g}(\mathsf{g}(x)) \to \mathsf{g}(\mathsf{h}(\mathsf{g}(x)))\}$. There obviously is an infinite chain and since $\mathcal{R}$ is terminating, this chain is also minimal. Thus, no sound processor may remove the pair of $\mathcal{P}$. However, if we choose $\pi(\mathsf{F}) = 1$ then the filtered right-hand side $\mathsf{g}(\mathsf{g}(x))$ of the pair in $\mathcal{P}$ is embedded in the filtered left-hand side $\mathsf{g}(\mathsf{h}(\mathsf{g}(x)))$. Therefore, it is not allowed to change the subterm relation in Theorem 4.41 to the embedding order.

Summary of Chapter 4

In this chapter we have seen how one can use reduction pairs to delete pairs and rules which can occur only finitely often in chains. The five most important processors – starting with the most efficient one, and ending with the most powerful one – are the processor of Theorem 4.41 based on the subterm criterion (restricted power since the order is fixed to the subterm relation), the two processors of Theorems 4.20 and 4.22 to remove rules (medium power since the order is not fixed but argument filters are not allowed), and the two reduction pair processors of Theorems 4.27 and 4.32 to delete pairs (most powerful since the order is not fixed, argument filters are allowed, and less constraints need to be considered due to the argument filter). Like the processors of the previous chapter, these five processors should be applied as often as possible, since they always simplify a given DP problem. However, the automation of these processors is more difficult, since finding a suitable reduction pair imposes a major search problem.

Nevertheless, efficient techniques to search for reduction pairs are available in our work [CSL+06, FGM+07, GTSF03, GTSF06, STA+07] and in the work of [CLS06, CMTU05, HM05, ZHM07]. Whereas the older techniques of [CMTU05, GTSF03, GTSF06, HM05] describe stand-alone approaches, the current techniques encode the search as a SAT problem and benefit from modern SAT solvers which turns out to be a more efficient approach.

The other two reduction pair processors of Theorems 4.2 and 4.18 are not that important since they are subsumed by Theorems 4.27 and 4.32, respectively. They have only been introduced in this thesis in order to illustrate the basic ideas of the reduction pair processor, before integrating the more complex concepts of usable (resp. needed) rules w.r.t. an argument filter.

Both the needed rules processor of Theorem 4.12 and the argument filter processor of Theorem 4.38 are also useful, but there is one severe problem when integrating them in an automated tool. As they easily introduce non-termination, one usually has to perform a complete termination proof to detect whether the application was helpful. Especially for the argument filter processor this is critical, since there are potentially exponentially many resulting DP problems that have to be investigated. Nevertheless, in combination with other techniques – as illustrated in the upcoming Example 6.25 – the automation becomes feasible.

Basic versions of the techniques of Theorems 4.20, 4.22, 4.27, and 4.32 have already been published by us in [GTS05a, GTS05b, GTSF03, GTSF06, TGS04] and in [Urb01, HM07, Zan05b]. However, in all these papers only certain aspects to reduce the set of constraints have been considered. For example, in these papers there is no variant of needed rules which is based on unification, and there is no variant of usable rules which is based on unification that also integrates argument filters. In contrast, here we have combined needed and usable rules with argument filters, we have estimated them by using unification, we have integrated normal-form checks, we have allowed arbitrary estimations of *Cap*, and we have considered $\mathcal{Q}$-restricted rewriting.

Moreover, we have solved the question under which conditions minimality can be preserved by the needed rules processor. Here, only the negative result was already published by us in [GTSF06].

The last contribution of this chapter is the formulation of the subterm criterion of Hirokawa and Middeldorp [HM07] as a processor, where we replace tuple symbols by the new notion of head symbols. This is required, as in a DP problem there are no tuple-symbols any more, since the pairs and rules are two arbitrary TRSs.

With our current set of processors we can already prove termination of many TRSs. Nevertheless, there are classes of TRSs where our techniques are not yet sufficient (if one is limited to standard orders). In the following two chapters we will see two of these classes and, of course, powerful processors are presented to tackle these problems.

We end this chapter with a discussion on future work in the area of constraint solving. We see two possible directions. Although the generated constraints can often be satisfied by the embedding due to our contributions, one sometimes needs powerful orders beyond the class of the well-known simplification orders. That progress can be made in this direction has been impressively demonstrated in the recent work of [EWZ06, HM07, HW06]. The two new classes of orders which are based on negative polynomial interpretations and matrix interpretations turn out to be very useful. Both approaches can be efficiently mechanized and help to solve previously unsolvable termination problems.

The other direction is to develop new techniques that generate constraints which are simpler to satisfy than those that are generated with the current processors. That progress can be achieved has been illustrated by us in [GTSS07]. There we have shown how to generate *conditional* constraints which can prove termination of algorithms that are terminating by increasing an argument until a bound is reached – a previously hard problem. Moreover, in that work there are *less restrictions* on the orders, e.g., the order $\succsim$ does not have to be monotonic any more. It would be useful to extend these ideas further. Moreover, we think that it is even possible to improve the processors that have been presented in this chapter. For example under certain conditions it should be possible to remove rules (and not only pairs) although one uses argument filters.

It remains to state that for both directions one also needs to design efficient algorithms for the automation.

5. Processors Based on Pair Transformations

As shown in [AG00, GA01], to increase the power of the DP approach, a dependency pair may be transformed by *rewriting*, *narrowing*, or *instantiation*. These transformations are useful if one has to prove termination of TRSs which do not use pattern matching, but use tests and selectors instead. We illustrate the difference with the following TRS.

$$\begin{aligned}
\mathsf{p}(\mathsf{s}(x)) &\to x && (55)\\
\mathsf{isNonZero}(\mathsf{s}(x)) &\to \mathsf{true} && (56)\\
\mathsf{isNonZero}(\mathsf{0}) &\to \mathsf{false} && (57)\\
\mathsf{f}(\mathsf{s}(x)) &\to C[\mathsf{f}(x)] && (58)\\
\mathsf{f}(x) &\to \mathsf{if}(\mathsf{isNonZero}(x), x) && (59)\\
\mathsf{if}(\mathsf{true}, x) &\to C[\mathsf{f}(\mathsf{p}(x))] && (60)
\end{aligned}$$

Essentially, rule (58) does the same as rules (59) and (60). But while rule (58) is defined via pattern matching, one uses the test isNonZero and the selector p in the rules (59) and (60). As a consequence, it is easy to find a reduction pair $(\succsim, \succ)$ which satisfies $\mathsf{s}(x) \succ x$, but it is impossible to find a reduction pair which solves $x \succ \mathsf{p}(x)$. The reason for the latter is that one also has to take care of the possibility that x is instantiated by 0, since there is no information available in the constraint of rule (60) that one tested $x \neq 0$ with rule (59). However, using the transformations will essentially pass this information to rule (60). Then if one replaces (60) by the resulting new rule

$$\mathsf{if}(\mathsf{true}, \mathsf{s}(x)) \to C[\mathsf{f}(x)]$$

it is again easy to solve the constraints of the reduction pair processor.

As in [GTS05a, GTSF06] we adapt these transformations from the DP approach to the DP framework. Given a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, they replace one of the pairs $s \to t$ in $\mathcal{P}$ by several new ones which result from rewriting, narrowing, or instantiating $s \to t$.

Compared to the original versions of these transformations in the DP approach, they are now improved and modularized considerably. One reason is that now these transformations can be applied at any time during the proof, and moreover, the conditions for their applicability only have to take the pairs and rules in the current DP problem into account. In this way, these conditions are satisfied much more often than in the original DP approach, where such transformations were only permitted in the very beginning. But even compared to the newer versions of these transformation in the DP framework (including the forward instantiation processor of [GTSF06]) they are improved considerably.

The first improvement is that our transformation processors contain more *checks on normal forms* than previous versions. The benefits are weaker application conditions, and less new pairs will be generated.

A second reason for the improvement is that for the first time a pair-*graph* $\mathcal{P}$ instead of a *set* of pairs $\mathcal{P}$ is used. This always improves the efficiency of the transformations and sometimes the power of the transformations is increased, cf. Example 5.4 and Example 5.12.

A third improvement is the integration of $\mathcal{Q}$. In [GTSF06] only full- or innermost rewriting is considered. And even in [GTS05a] where $\mathcal{Q}$-restricted rewriting is considered, only the cases $\mathcal{Q} = \varnothing$ and $\mathcal{Q} \supseteq \mathit{lhs}(\mathcal{R})$ are investigated. However, here there is no restriction on $\mathcal{Q}$ and if a processor requires a specific property of $\mathcal{Q}$ then we provide counterexamples to show the necessity of that property.

The fourth improvement is due to our *semantic* versions of the *Cap*-function and of the usable rules. Whereas previous transformations are formulated for specific versions of *Cap* and of usable rules, we now present the transformations for arbitrary estimations. Hence, future improvements of usable rules (i.e., better estimations of the semantic usable rules of Definition 3.24) can be integrated into the transformation processors without rechecking the proofs. That this is not completely trivial can be seen in Example 5.14. There it is demonstrated that the rewriting processor as formulated in [GTS05a] and [GTSF06] is unsound for certain estimations of usable rules.

The last and most powerful improvement is the development of a completely new idea. In the narrowing processor one can choose a *position* and then only those narrowings have to be considered that correspond to rewrite steps below that position. This will be essential to handle problems where tests and selectors are used in combination with an accumulator, see Section 5.4 for more details and examples.

We will demonstrate all transformations with the following running example where some of our new improvements are crucial, e.g. the forward instantiation processor in the generalized case of $\mathcal{Q}$-restricted rewriting, and the integration of the position in the narrowing processor.

Example 5.1. Consider the TRS $\mathcal{R}$ to compute division on natural numbers:

$$\mathsf{p}(\mathsf{s}(x)) \to x \tag{61}$$
$$\mathsf{p}(\mathsf{0}) \to \mathsf{s}(\mathsf{0}) \tag{62}$$
$$\mathsf{plus}(\mathsf{s}(x), y) \to \mathsf{plus1}(\mathsf{s}(x), \mathsf{s}(y)) \tag{63}$$
$$\mathsf{plus1}(x, y) \to \mathsf{plus2}(\mathsf{p}(x), y) \tag{64}$$
$$\mathsf{plus2}(x, y) \to \mathsf{plus}(x, y) \tag{65}$$
$$\mathsf{plus}(\mathsf{0}, y) \to y \tag{66}$$
$$\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{minus}(x, y) \tag{67}$$
$$\mathsf{minus}(\mathsf{0}, x) \to \mathsf{0} \tag{68}$$
$$\mathsf{minus}(x, \mathsf{0}) \to x \tag{69}$$
$$\mathsf{minus}(x, x) \to \mathsf{0} \tag{70}$$
$$\mathsf{ge}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{ge}(x, y) \tag{71}$$
$$\mathsf{ge}(\mathsf{0}, \mathsf{s}(x)) \to \mathsf{ff} \tag{72}$$
$$\mathsf{ge}(x, \mathsf{0}) \to \mathsf{tt} \tag{73}$$
$$\mathsf{div}(x, y) \to \mathsf{quot}(x, y, \mathsf{0}) \tag{74}$$
$$\mathsf{quot}(x, y, z) \to \mathsf{if}(\mathsf{ge}(\mathsf{0}, y), \mathsf{ge}(x, y), \mathsf{minus}(x, y), y, \mathsf{plus}(z, \mathsf{s}(\mathsf{0}))) \tag{75}$$
$$\mathsf{if}(\mathsf{tt}, b, x, y, z) \to \mathsf{divByZeroError} \tag{76}$$
$$\mathsf{if}(\mathsf{ff}, b, x, y, z) \to \mathsf{if2}(b, x, y, z) \tag{77}$$
$$\mathsf{if2}(\mathsf{tt}, x, y, z) \to \mathsf{quot}(x, y, z) \tag{78}$$
$$\mathsf{if2}(\mathsf{ff}, x, y, z) \to \mathsf{p}(z) \tag{79}$$

As this TRS belongs to a class where innermost termination is equivalent to termination we set $\mathcal{Q} = lhs(\mathcal{R})$. We simplify the initial DP problem with the processors of Section 3 and Section 4 using the embedding order. This yields two remaining DP problems which cannot be simplified further. The first one for the plus-function is $\mathcal{D}_8 = (\{(80),(81),(82)\},\mathcal{Q},\{(61),(62)\},\mathbf{m})$ and the second DP problem originates from the quot-function: $\mathcal{D}_9 = (\{(83),(84),(85)\},\mathcal{Q},\{(61)-(73)\},\mathbf{m})$.

$$\mathsf{PLUS}(\mathsf{s}(x),y) \to \mathsf{PLUS1}(\mathsf{s}(x),\mathsf{s}(y)) \tag{80}$$
$$\mathsf{PLUS1}(x,y) \to \mathsf{PLUS2}(\mathsf{p}(x),y) \tag{81}$$
$$\mathsf{PLUS2}(x,y) \to \mathsf{PLUS}(x,y) \tag{82}$$
$$\mathsf{QUOT}(x,y,z) \to \mathsf{IF}(\mathsf{ge}(\mathsf{0},y),\mathsf{ge}(x,y),\mathsf{minus}(x,y),y,\mathsf{plus}(z,\mathsf{s}(\mathsf{0}))) \tag{83}$$
$$\mathsf{IF}(\mathsf{ff},b,x,y,z) \to \mathsf{IF2}(b,x,y,z) \tag{84}$$
$$\mathsf{IF2}(\mathsf{tt},x,y,z) \to \mathsf{QUOT}(x,y,z) \tag{85}$$

Note that $\mathcal{D}_8$ cannot be simplified any further using the processors of Section 4 if one is restricted to quasi-simplification orders. Even reduction pair processors using negative polynomial orders of [HM07] are not applicable on this DP problem. For example a polynomial interpretation which maps $\mathsf{p}(x)$ to $x-1$ is prohibited by rule (62).

The situation for $\mathcal{D}_9$ is even worse. The constraints of any reduction pair processor cannot be satisfied by any $\mathcal{C}_\varepsilon$-compatible reduction pair. The reason is that $\mathcal{R} \cup \mathcal{C}_\varepsilon$ is not terminating and $(\{(83),(84),(85)\},\mathcal{Q},\{(61)-(73)\} \cup \mathcal{C}_\varepsilon,\mathbf{m})$ is not finite which is shown by the following infinite reduction.

$$\begin{array}{ll}
 & \mathsf{QUOT}(\mathsf{0},\mathsf{c}(\mathsf{0},\mathsf{s}(\mathsf{0})),\mathsf{0}) \\
\to & \mathsf{IF}(\mathsf{ge}(\mathsf{0},\mathsf{c}(\mathsf{0},\mathsf{s}(\mathsf{0}))),\mathsf{ge}(\mathsf{0},\mathsf{c}(\mathsf{0},\mathsf{s}(\mathsf{0}))),\mathsf{minus}(\mathsf{0},\mathsf{c}(\mathsf{0},\mathsf{s}(\mathsf{0}))),\mathsf{c}(\mathsf{0},\mathsf{s}(\mathsf{0})),\mathsf{plus}(\mathsf{0},\mathsf{s}(\mathsf{0}))) \\
\to^+_{\mathcal{C}_\varepsilon} & \mathsf{IF}(\mathsf{ge}(\mathsf{0},\mathsf{s}(\mathsf{0})),\mathsf{ge}(\mathsf{0},\mathsf{0}),\mathsf{minus}(\mathsf{0},\mathsf{0}),\mathsf{c}(\mathsf{0},\mathsf{s}(\mathsf{0})),\mathsf{plus}(\mathsf{0},\mathsf{s}(\mathsf{0}))) \\
\to^*_{\mathcal{R}} & \mathsf{IF}(\mathsf{ff},\mathsf{tt},\mathsf{0},\mathsf{c}(\mathsf{0},\mathsf{s}(\mathsf{0})),\mathsf{s}(\mathsf{0})) \\
\to & \mathsf{IF2}(\mathsf{tt},\mathsf{0},\mathsf{c}(\mathsf{0},\mathsf{s}(\mathsf{0})),\mathsf{s}(\mathsf{0})) \\
\to & \mathsf{QUOT}(\mathsf{0},\mathsf{c}(\mathsf{0},\mathsf{s}(\mathsf{0})),\mathsf{s}(\mathsf{0})) \\
\to^+ & \mathsf{QUOT}(\mathsf{0},\mathsf{c}(\mathsf{0},\mathsf{s}(\mathsf{0})),\mathsf{s}(\mathsf{s}(\mathsf{0}))) \\
\to^+ & \ldots
\end{array}$$

The class of $\mathcal{C}_\varepsilon$-compatible reduction pairs includes negative polynomial orders with negative constants [HM07], polynomial orders over the non-negative reals [Luc05], and the orders that are based on matrix interpretations [EWZ06, HW06]. And even negative polynomial orders with negative coefficients [HM07] fail on this example. Hence, it is unlikely that an automated termination proof of $\mathcal{D}_9$ can be performed using only the processors of the previous chapters. However, with the help of the processors of this chapter, in the end we can solve all problems using simple orders like polynomial orders where every function is interpreted by a linear polynomial with 0 and 1 as only coefficients.

Before we present all four transformations *instantiation*, *forward instantiation*, *rewriting*, and *narrowing* in the corresponding four Sections 5.1–5.4, we will need a new graph operation. For $\mathcal{P} = (N \uplus \{m\}, E)$ we define $\mathcal{P}[m/N']$ as the graph where we replace the node m by the set of nodes N'. More precisely, $\mathcal{P}[m/N'] = (N \cup N', E')$ where the set E' of new edges is defined as follows:

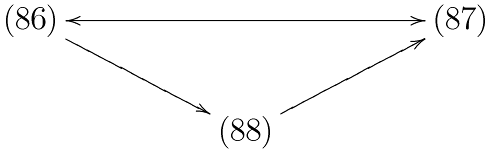
$$\begin{array}{rcl}
E' & = & (E \cap (N \times N)) \\
 & \cup & \{(n,n') \mid n \in N, n' \in N', (n,m) \in E\} \\
 & \cup & \{(n',n) \mid n \in N, n' \in N', (m,n) \in E\} \\
 & \cup & \{(n',n'') \mid n',n'' \in N', (m,m) \in E\}
\end{array}$$

Note that it is assumed that N is disjoint from N', because otherwise we would have the problem that one can introduce new edges between nodes of N. However, this requirement is not satisfied by the upcoming processors. Therefore, we allow multiple nodes for the same pair, i.e., $\mathcal{P}$ is not a graph with a set of pairs any more, but it is a graph where the nodes are labeled with pairs. As this new layer would make the presentation unnecessary complicated we still use a pair-graph, where the nodes are the pairs, and we omit the details on how to handle different occurrences of the same pair in a pair-graph. We just illustrate the difference in the following example.

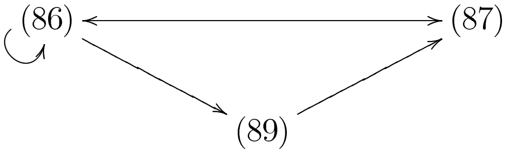
Example 5.2. Let the following pairs be given.

$$\begin{array}{rcl} \mathsf{F}(\mathsf{s}(x)) & \to & \mathsf{F}(\mathsf{s}(x)) \qquad (86) \\ \mathsf{F}(\mathsf{s}(x)) & \to & \mathsf{F}(x) \qquad (87) \\ \mathsf{F}(x) & \to & \mathsf{F}(\mathsf{g}(x)) \qquad (88) \\ \mathsf{F}(\mathsf{0}) & \to & \mathsf{F}(\mathsf{0}) \qquad (89) \end{array}$$

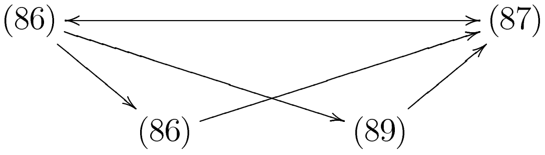
If $\mathcal{R} = \{\mathsf{g}(\mathsf{s}(x)) \to \mathsf{s}(x), \mathsf{g}(\mathsf{0}) \to \mathsf{0}\}$ and $\mathcal{P}$ has the following graph structure which uses the first three pairs,[21]



then there is no infinite chain. Now consider the replacement of (88) by the pairs (86) and (89). Notice that this will be possible by the narrowing processor, and in the corresponding Theorem 5.19 it is stated that this application of the narrowing processor is complete. But if one identifies the old and the new pair (86), the graph $\mathcal{P}[(88)/\{(86),(89)\}]$ has a graph structure which allows an infinite chain with pair (86).



However, we distinguish the old and the new pair (86). Then there is still no infinite chain possible.



Nevertheless, if some nodes are in the graph are labeled with the same pair and if they have the same incoming and outgoing edges, then one can safely merge these nodes.

---

[21] A graph structure like this can arise after applying the argument filter processor of Theorem 4.38, cf. Example 4.40.

5.1. Instantiation

As first transformation we present the *instantiation* processor. The idea is to consider the predecessors $u \to v$ of a pair $s \to t$. If certain parts of v remain unchanged when rewriting $v\sigma$ to $s\sigma$ then one can instantiate the pair $s \to t$ by those parts that remain unchanged. (One can use (an estimation of) the *Cap*-function to detect the unchanged parts.) For example if we look at the pairs (80) and (81) in the running example then we see that the s's of the right-hand side of (80) will remain unchanged. Hence, one should be able to instantiate the variables x and y in (81) by $\mathsf{s}(x)$ and $\mathsf{s}(y)$.

The difference of our instantiation processor to previous versions is the use of the graph structure and the use of arbitrary estimations of *Cap*. Moreover, we do not need two different versions for termination and innermost-termination any more, but we can formulate one generalized version.

Theorem 5.3 (Instantiation Processors)**.** *Let ECap be an estimated Cap-function. Let Proc be an* instantiation *processor which transforms a DP problem* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ *with* $\mathcal{P} = (N \uplus \{s \to t\}, E)$ *into* $\{(\mathcal{P}[s \to t/N'], \mathcal{Q}, \mathcal{R}, f)\}$ *where* N' *is the following set of new pairs.*

$$\{s\delta \to t\delta \mid (u \to v, s \to t) \in E,\ \delta = mgu(ECap^{\{u,s\}}_{\mathcal{R},\mathcal{Q}}(v), s),\ \{u\delta, s\delta\} \subseteq NF(\mathcal{Q})\}$$

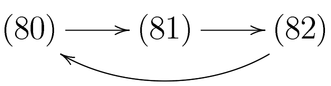
Then the processor Proc is sound and complete.

Note that there is a strong correspondence between the instantiation processor and the dependency graph estimation of Definition 3.9 as we perform the same unification and the same checks on $\mathcal{Q}$-normal forms. Therefore, to improve efficiency in the automation one can memorize the unifiers δ when estimating the dependency graph in Definition 3.9 and can reuse them for the instantiation processor.

Example 5.4. We continue the termination proof of the running example.

$$\begin{align*} \mathsf{PLUS}(\mathsf{s}(x), y) &\to \mathsf{PLUS1}(\mathsf{s}(x), \mathsf{s}(y)) \tag{80}\\ \mathsf{PLUS1}(x, y) &\to \mathsf{PLUS2}(\mathsf{p}(x), y) \tag{81}\\ \mathsf{PLUS2}(x, y) &\to \mathsf{PLUS}(x, y) \tag{82} \end{align*}$$

To this end we apply the instantiation processor on the DP problem $\mathcal{D}_8$ where the pair-graph has the following structure.



To instantiate the pair (81) we only have to look at the only predecessor (80). As $ECap^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(\mathsf{PLUS}(\mathsf{s}(x), \mathsf{s}(y))) = \mathsf{PLUS}(\mathsf{s}(x), \mathsf{s}(y))$ we can instantiate the variables in (81) by $\mathsf{s}(x)$ and $\mathsf{s}(y)$. Thus, the instantiation processor replaces (81) by (90).

$$\mathsf{PLUS1}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{PLUS2}(\mathsf{p}(\mathsf{s}(x)), \mathsf{s}(y)) \tag{90}$$

Note that if the DP problems only contained a set of pairs without the graph structure then we would have to examine all three pairs as predecessors which is clearly less efficient. In the same way, in the resulting DP problem we only obtain the edges (80) → (90), (90) → (82), and (82) → (80). Hence, by applying the dependency graph processor of Theorem 3.3 it suffices to check whether the two new edges (80) → (90) and (90) → (82)

can be deleted, but one does not have to check whether one can delete the remaining three possible edges (90) → (80), (90) → (90), and (82) → (90). Thus, the graph structure of DP problems again helps to obtain efficiency but sometimes even power is affected. This is demonstrated in more detail with an example application of the rewriting processor in Example 5.12.

After the instantiation, the usable rules processor of Theorem 3.25 is applicable and deletes the unused rule (62). And afterwards, every reduction pair processor of Chapter 4 can delete pair (90) in the resulting DP problem $\mathcal{D}_{10} = (\{(80), (90), (82)\}, \mathcal{Q}, \{(61)\}, \mathbf{m})$ with the negative polynomial order with $\mathcal{P}ol(\mathsf{s}(x)) = x + 1$, $\mathcal{P}ol(\mathsf{p}(x)) = \mathit{max}\{x - 1, 0\}$, and $\mathcal{P}ol(\mathsf{PLUS}(x, y)) = \mathcal{P}ol(\mathsf{PLUS1}(x, y)) = \mathcal{P}ol(\mathsf{PLUS2}(x, y)) = x$. Then the remaining problem is solved by the processors based on the dependency graph.

However, if one uses reduction pairs based on quasi-simplification orders it is still not possible to simplify $\mathcal{D}_{10}$. To this end, we will need the rewriting processor of Section 5.3.

Note that to simplify the other DP problem $\mathcal{D}_9$ we cannot use the instantiation processor, but a similar technique which is presented in the next section.

5.2. Forward Instantiation

We now present the technique of forward instantiation [GTSF06]. While the instantiation processor was based on the dependency graph estimation given in Definition 3.9 where one examines for two pairs $s \to t$ and $u \to v$ whether it is possible to rewrite $t\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} u\sigma$, the forward instantiation processor takes an approach similar to the dependency graph estimation of Definition 3.31 where one examines whether it is possible to rewrite $u\sigma \to^*_{\mathcal{R}^{-1}} t\sigma$ with the reversed TRS $\mathcal{R}^{-1}$. Moreover, we will also integrate the results on usable rules, i.e., it suffices to consider $u\sigma \to^*_{\mathcal{R}'} t\sigma$ where $\mathcal{R}' = (\mathcal{U}^{\{s,u\}}_{\mathcal{R},\mathcal{Q}}(t))^{-1}$. Then we use the same idea as in the instantiation processor, i.e., we detect the unchanged parts by applying (an estimation of) the *Cap*-function and then instantiate $s \to t$ accordingly.

Theorem 5.5 (Forward Instantiation Processors)**.** *Let ECap be an estimated Cap-function and let $\mathcal{EU}$ estimate the usable rules. Let Proc be a* forward instantiation *processor which transforms a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ with $\mathcal{P} = (N \uplus \{s \to t\}, E)$ into $\{(\mathcal{P}[s \to t/N'], \mathcal{Q}, \mathcal{R}, f)\}$ where N' is the following set of new pairs.*

$$\left\{ s\delta \to t\delta \;\middle|\; \begin{array}{l} (s \to t, u \to v) \in E,\ \mathcal{R}' = (\mathcal{EU}^{\{s,u\}}_{\mathcal{R},\mathcal{Q}}(t))^{-1}, \\ \delta = \mathit{mgu}(\mathit{ECap}^{\varnothing}_{\mathcal{R}',\varnothing}(u), t),\ \text{and } \{s\delta, u\delta\} \subseteq \mathit{NF}(\mathcal{Q}) \end{array} \right\}$$

Then the processor Proc is sound and complete.

As for the instantiation processor, to improve efficiency one can memorize each unifier δ in the computation of the star-estimation of the dependency graph in Definition 3.9 to reuse it now for the forward instantiation processor.

Example 5.6.

$$\mathsf{QUOT}(x, y, z) \to \mathsf{IF}(\mathsf{ge}(0, y), \mathsf{ge}(x, y), \mathsf{minus}(x, y), y, \mathsf{plus}(z, \mathsf{s}(0))) \tag{83}$$

$$\mathsf{IF}(\mathsf{ff}, b, x, y, z) \to \mathsf{IF2}(b, x, y, z) \tag{84}$$

$$\mathsf{IF2}(\mathsf{tt}, x, y, z) \to \mathsf{QUOT}(x, y, z) \tag{85}$$

We continue in the termination proof of Example 5.11 where we still have to solve the DP problem $(\{(83), (84), (85)\}, \mathcal{Q}, \{(61)\text{–}(73)\}, \mathbf{m})$. Here we can use the forward instantiation processor to instantiate the pair (84). The only succeeding pair $u \to v$ in the graph is (85). Now we have to build the reversed usable rules $\mathcal{R}'$. As (84) does not possess usable rules we obtain $\mathcal{R}' = \varnothing$. Then we have to unify $\mathit{ECap}^{\varnothing}_{\mathcal{R}',\varnothing}(\mathsf{IF2}(\mathsf{tt}, x', y', z')) = \mathsf{IF2}(\mathsf{tt}, x', y', z')$ with the right-hand side $\mathsf{IF2}(b, x, y, z)$ of (84). Instantiating (84) with the mgu $\delta = \{x'/x, y'/y, z'/z, b/\mathsf{tt}\}$ yields the new pair

$$\mathsf{IF}(\mathsf{ff}, \mathsf{tt}, x, y, z) \;\; \to \;\; \mathsf{IF2}(\mathsf{tt}, x, y, z) \tag{91}$$

The resulting DP problem $\mathcal{D}_{11} = (\{(83), (91), (85)\}, \mathcal{Q}, \{(61) - (73)\}, \mathbf{m})$ can then be simplified further by the narrowing processor which is demonstrated in Example 5.22. There it is also explained why this application of the forward instantiation processor is strongly required for a successful termination proof.

5.3. Rewriting

While adapting instantiation to the DP framework is straightforward, the adaption of the *rewriting* transformation is more problematic. In the classical dependency pair approach, rewriting is only applicable for innermost termination proofs (i.e., if $\mathit{NF}(\mathcal{Q}) = \mathit{NF}(\mathcal{R})$). The problem is that the original proofs for its soundness and completeness [GA01, Theorems 14, 15, 18, and 19] do not extend to the case where $\mathit{NF}(\mathcal{Q}) \subset \mathit{NF}(\mathcal{R})$. The soundness proof relies on the result that weak innermost termination and non-overlappingness imply confluence and termination [Gra95], and the completeness proof relies on the result that innermost termination implies normalization. Obviously, these results do not extend to $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$, i.e., in general $\mathcal{Q}$-termination and non-overlappingness do not imply confluence, termination, or normalization. As a counterexample, consider $\mathcal{R} = \{\mathsf{f}(\mathsf{a}) \to \mathsf{s}(\mathsf{f}(\mathsf{a})), \mathsf{g}(x, x) \to \mathsf{c}, \mathsf{g}(x, \mathsf{s}(x)) \to \mathsf{d}\}$ and $\mathcal{Q} = \mathit{lhs}(\mathcal{R}) \cup \{\mathsf{a}\}$. Now $\mathcal{R}$ is $\mathcal{Q}$-terminating and non-overlapping, but neither terminating nor confluent nor normalizing. For example, the term $\mathsf{f}(\mathsf{a})$ does not have an $\mathcal{R}$-normal form, and the term $\mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{a}))$ can be reduced by $\mathcal{R}$ to the normal forms c and d.

However, such an extension is urgently required as with the previous processors we will have to handle DP problems where $\mathit{NF}(\mathcal{Q}) \subset \mathit{NF}(\mathcal{R})$ even if in the beginning $\mathit{NF}(\mathcal{Q}) = \mathit{NF}(\mathcal{R})$. Now it would be desirable if one could still apply the rewriting transformation (this will be demonstrated when handling the DP problem $\mathcal{D}_{10}$ of our running example in Example 5.11). For the soundness of the rewriting transformation we need at least confluence of the $\mathcal{Q}$-restricted rewrite relation. However, our previous confluence criterion of Lemma 3.19 could only be used for $\mathit{NF}(\mathcal{R}) \subseteq \mathit{NF}(\mathcal{Q})$ but now we are in the case $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$. The following lemma states that if all critical pairs at the root level are trivial, then we even have the *diamond property* (i.e., if $t \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t_1$ and $t \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t_2$, then $t_1 = t_2$ or there exists a t' such that $t_1 \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t'$ and $t_2 \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t'$).

Lemma 5.7 (Confluence of $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$)**.** *Let $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$. Let all critical pairs of $\mathcal{R}$ at the root level be trivial. (For all rules $\ell_1 \to r_1$ and $\ell_2 \to r_2 \in \mathcal{R}$ where ℓ_1 and ℓ_2 are unifiable, the mgu δ of ℓ_1 and ℓ_2 instantiates the right-hand sides of the rules to the same term, i.e., $r_1\delta = r_2\delta$.) Then $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ has the diamond property and hence, is confluent.*

We now give a new proof to show that the rewriting transformation is sound whenever $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$ is satisfied. A similar proof was already presented in [GTS05a], however there are three differences.

First, in [GTS05a] certain properties of (syntactically defined) usable rules are used in the proof which are not satisfied for every estimation of the (semantically defined) usable rules of Definition 3.24. In Example 5.14 it is shown that the rewriting processor of [GTS05a] even is unsound if one allows arbitrary estimations of usable rules. To this end, we added a new requirement in the rewriting processor.

The second difference is that we allow rewriting more often. In [GTS05a] the usable rules had to be non-overlapping whereas here we only require trivial critical pairs.

And the third difference is in the completeness condition. In [GTS05a] one demands for all proper subterms t of the redex that $\mathcal{U}^{\varnothing}_{\mathcal{Q},\varnothing}(t) \subseteq \mathcal{R}$. The problem is that $\mathcal{Q}$ is a rewrite system in [GTS05a] whereas here $\mathcal{Q}$ is a set of terms. So, it is unclear what the usable *rules* should be, if $\mathcal{Q}$ only contains *terms*. Moreover, again certain properties of the syntactic usable rules of [GTS05a] are used in the proof. So, we have to replace the property $\mathcal{U}^{\varnothing}_{\mathcal{Q},\varnothing}(t) \subseteq \mathcal{R}$ by something different.

In essence, the property $\mathcal{U}^{\varnothing}_{\mathcal{Q},\varnothing}(t) \subseteq \mathcal{R}$ is demanded to ensure that whenever $t\sigma$ can be reduced to a normal form u w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$, then u is in $\mathcal{Q}$-normal form, too. This is important whenever we want to normalize the subterm $t\sigma$ w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ in order to perform a reduction above $t\sigma$ afterwards. Only if $t\sigma$ can be rewritten to a normal form w.r.t. $\mathcal{Q}$ then the reduction above $t\sigma$ respects the evaluation strategy. This condition on the normal forms can be used to obtain completeness of both the rewriting and the narrowing processors.

Definition 5.8 ($\mathcal{Q} - \mathcal{R}$ Normal Form Condition). *Let $\mathcal{R}$ be a TRS and $\mathcal{Q}$ and $\mathcal{S}$ be sets of terms. A term t satisfies the $\mathcal{Q} - \mathcal{R}$* normal form condition w.r.t. *$\mathcal{S}$, $nfc^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$ for short, iff for every substitution σ with $\mathcal{S}\sigma \subseteq NF(\mathcal{Q})$ each $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$-normal form of $t\sigma$ is in $\mathcal{Q}$-normal form.*

Example 5.9. We shortly illustrate the $\mathcal{Q} - \mathcal{R}$ normal form condition with the following TRS $\mathcal{R}$ where $\mathcal{Q} = lhs(\mathcal{R}) \cup \{\mathsf{h}(\mathsf{a})\}$.

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

Then $nfc^{\{x\}}_{\mathcal{R},\mathcal{Q}}(\mathsf{f}(x))$ is not satisfied. The reason is that one can instantiate x by a and obtains $\mathsf{f}(\mathsf{a}) \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} \mathsf{h}(\mathsf{a})$ where the latter term is in normal form w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$, but not w.r.t. $\mathcal{Q}$.

However, if one removes the third rule from $\mathcal{R}$ then $\mathsf{f}(x)$ satisfies the normal form condition. One reason is that a reduction to the critical term $\mathsf{h}(\mathsf{a})$ is not possible since then the subterm $\mathsf{h}(x)$ of the left-hand side of the second rule would be instantiated to a term which is not in $\mathcal{Q}$-normal form.

Unfortunately, it is undecidable whether a term satisfies the $\mathcal{Q} - \mathcal{R}$ normal form condition. We will present estimations of $nfc^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$ later in this section. One obvious result is that if $NF(\mathcal{Q}) = NF(\mathcal{R})$ then every term satisfies $nfc^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$, since then every term is in $\mathcal{R}$-normal form iff it is in normal form w.r.t. $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$. As one can see in the upcoming definition of the rewriting processor, this will show that our results encompass the completeness results for innermost termination from [GA01, Theorem 19].

Now we introduce the rewriting processor. It states that in a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ with $NF(\mathcal{Q}) \subseteq NF(\mathcal{R})$, any $s \to t \in \mathcal{P}$ can be replaced by $s \to t'$ if t rewrites to t' at some position p. The only applicability conditions are that the $\mathcal{Q}$-restricted rewrite relation of the usable rules must be confluent and that there must only be trivial critical pairs between the rule used for rewriting t and the usable rules. In contrast to the (forward) instantiation transformations, which perform *all* possible instantiations, here one may replace $s \to t$ by *any* pair resulting from rewriting t.

Theorem 5.10 (Rewriting Processors)**.** *Let Proc be a processor which transforms a DP problem* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ *into* $\{(\mathcal{P}[s \to t/\{s \to t'\}], \mathcal{Q}, \mathcal{R}, f)\}$ *or into* $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$*. In the former case, all following conditions must be satisfied:*

- $s \to t \in \mathcal{P}$
- $t \to_{\mathcal{R},p} t'$ *by a rule* $\ell \to r$ *where for* $\mathcal{U} = \mathcal{EU}^{\{s\}}_{\mathcal{R},\mathcal{Q}}(t|_p)$ *the rewrite relation* $\xrightarrow{\mathcal{Q}}_{\mathcal{U}}$ *is confluent and where there are only trivial critical pairs between* $\ell \to r$ *and* $\mathcal{U}$*.*[22]
- $NF(\mathcal{Q}) \subseteq NF(\mathcal{R})$

Proc is sound, and it is complete if every proper subterm of $t|_p$ *satisfies the* $\mathcal{Q}-\mathcal{R}$ *normal form condition w.r.t.* $\{s\}$*.*

Note that using Lemma 5.7 the application conditions of the rewriting processor can easily be checked. Moreover, after the following examples we will also see a possibility to estimate the $\mathcal{Q} - \mathcal{R}$ normal form condition, hence we can detect if an application of the rewriting processor is complete.

First, we show the benefit of the rewriting processor in the running example of this chapter.

Example 5.11. We recapitulate the situation at the end of Example 5.4.

$$\begin{aligned} \mathsf{PLUS}(\mathsf{s}(x), y) &\to \mathsf{PLUS1}(\mathsf{s}(x), \mathsf{s}(y)) && (80)\\ \mathsf{PLUS1}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{PLUS2}(\mathsf{p}(\mathsf{s}(x)), \mathsf{s}(y)) && (90)\\ \mathsf{PLUS2}(x, y) &\to \mathsf{PLUS}(x, y) && (82) \end{aligned}$$

If we do not use negative polynomial orders we still have to solve the DP problem $\mathcal{D}_{10} = (\{(80), (90), (82)\}, \mathcal{Q}, \{(61)\}, \mathbf{m})$. Now, we can use the rewriting processor to replace pair (90) by the following new pair.

$$\mathsf{PLUS1}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{PLUS2}(x, \mathsf{s}(y)) \qquad (92)$$

Now every reduction pair processor of Section 4 can delete (92) with the embedding order and the remaining problem is solved by the processors based on the dependency graph.

For the other remaining DP problem $\mathcal{D}_{11}$ it turns out that neither the (forward-) instantiation processors nor the rewriting processors are applicable on this DP problem. To solve this problem we will need the narrowing processor of the next section.

The following examples show why possible extensions of the rewriting processor would destroy soundness or completeness, respectively. To be more precise, we show that each of the three application conditions is needed for the soundness of the rewriting processor, and the criterion to detect whether an application of the rewriting processor is complete cannot be weakened, too. Moreover, in Example 5.12 we show that the graph structure of DP problems not only increases efficiency but also power.

---

[22] Note that the critical pairs of $\ell \to r$ and $\mathcal{U}$ are only those when overlapping subterms of ℓ with left-hand sides of $\mathcal{U}$, but it is not required to consider overlaps of subterms of left-hand sides of $\mathcal{U}$ with ℓ.

Example 5.12. Consider the TRS $\mathcal{R} = \{\mathsf{g}(\mathsf{h}(x), y) \to y\}$ and the following pairs.

$$\begin{aligned} \mathsf{F}(x, x) &\to \mathsf{F}(\mathsf{a}, \mathsf{g}(\mathsf{h}(x), x)) && (93) \\ \mathsf{F}(\mathsf{a}, \mathsf{g}(x, y)) &\to \mathsf{F}(\mathsf{a}, y) && (94) \end{aligned}$$

For $\mathcal{Q} = \mathit{lhs}(\mathcal{R}) \cup \{\mathsf{h}(\mathsf{a})\}$ the dependency graph for (93) and (94) has the edges $E = \{(93) \to (94), (94) \to (93), (94) \to (94)\}$. That there is no edge from (93) to itself can be argued as follows. If there were an edge then x would have to be instantiated by a. But then the term $\mathsf{g}(\mathsf{h}(\mathsf{a}), \mathsf{a})$ could not be rewritten to a as $\mathsf{h}(\mathsf{a})$ is not in $\mathcal{Q}$-normal form. So, let $\mathcal{P} = (\{(93), (94)\}, E)$

No previous processor is able to simplify the DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ even if one allows arbitrary reduction pairs. The reason is that ignoring $\mathcal{Q}$ allows infinite chains which contain both pairs infinitely often:

$$\begin{aligned} & \mathsf{F}(\mathsf{a}, \mathsf{a}) \\ \to_{(93)} \; & \mathsf{F}(\mathsf{a}, \mathsf{g}(\mathsf{h}(\mathsf{a}), \mathsf{a})) \\ \to_{(94)} \; & \mathsf{F}(\mathsf{a}, \mathsf{a}) \\ \to_{(93)} \; & \cdots \end{aligned}$$

However using the rewriting processor we can replace (93) by the new pair (95).

$$\mathsf{F}(x, x) \to \mathsf{F}(\mathsf{a}, x) \qquad (95)$$

In this way we obtain the new DP problem $((\{(95), (94)\}, E'), \mathcal{Q}, \mathcal{R}, \mathbf{m})$ where $E' = \{(95) \to (94), (94) \to (95), (94) \to (94)\}$. Now every reduction pair processor can delete (94) with the embedding order and the argument filter π with $\pi(\mathsf{F}) = 2$ and $\pi(\mathsf{g}) = [2]$. The resulting DP problem $((\{(95)\}, \varnothing), \mathcal{Q}, \mathcal{R}, \mathbf{m})$ is now solved by the processor of Theorem 3.4 as there are no edges left.

But if DP problems did not have a graph structure then we would obtain the new DP problem $(\{(95), (94)\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$. This DP problem is clearly not finite as $(95), (95), \ldots$ is an infinite chain. Hence, our rewriting processor which works on DP problems with a graph-component is more powerful than previous rewriting processors [GA01, GTS05a, GTSF06] which work on sets of pairs.

The example also illustrates why the $\mathcal{Q}$–$\mathcal{R}$ normal form condition for all direct subterms of $t|_p$ is required to obtain completeness. If we changed $\mathcal{P}$ to $\mathcal{P}' = (\{(93), (94)\}, E \cup \{(93) \to (93)\})$ then there would not be an infinite $(\mathcal{P}', \mathcal{Q}, \mathcal{R})$-chain. The reason is that the DP problem $(\mathcal{P}', \mathcal{Q}, \mathcal{R}, \mathbf{m})$ could be transformed to $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ by the dependency graph processor and then be solved as before. However, the rewriting processor would replace $\mathcal{P}'$ by $\mathcal{P}'' = (\{(95), (94)\}, E'')$ where E'' would contain the edge from (95) to itself and we would obtain the minimal infinite $(\mathcal{P}'', \mathcal{Q}, \mathcal{R})$-chain. Note however that the redex $\mathsf{g}(\mathsf{h}(x), x)$ of the reduction has the direct subterm $\mathsf{h}(x)$ which does not satisfy the $\mathcal{Q} - \mathcal{R}$ normal form condition: if we instantiate x by a then we obtain $\mathsf{h}(\mathsf{a})$, which is in normal form w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ but not w.r.t. $\mathcal{Q}$. So the condition for completeness in Theorem 5.10 is violated.

Example 5.13. Since we only have to look at the critical pairs on root level for confluence of the $\mathcal{Q}$-restricted rewrite relation by Lemma 5.7, a natural question is whether the rewrite processor in Theorem 5.10 would already be sound if we only considered the critical pairs *at the root level* between $\ell \to r$ and the usable rules. This is refuted by the following counterexample. Let $\mathcal{R} = \{\mathsf{f}(\mathsf{c}) \to \mathsf{d}, \mathsf{f}(\mathsf{h}(x)) \to \mathsf{a}, \mathsf{h}(\mathsf{b}) \to \mathsf{c}, \mathsf{g}(\mathsf{d}, x) \to \mathsf{g}(\mathsf{f}(\mathsf{h}(x)), x)\}$ and $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$. Then $\mathcal{R}$ is not innermost terminating (i.e., not $\mathcal{Q}$-terminating):

$$\mathsf{g}(\mathsf{d}, \mathsf{b}) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \mathsf{g}(\mathsf{f}(\mathsf{h}(\mathsf{b})), \mathsf{b}) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \mathsf{g}(\mathsf{f}(\mathsf{c}), \mathsf{b}) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \mathsf{g}(\mathsf{d}, \mathsf{b}) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \ldots$$

The dependency graph has only one SCC $\{\mathsf{G}(\mathsf{d}, x) \to \mathsf{G}(\mathsf{f}(\mathsf{h}(x)), x)\}$. Since $\mathcal{R}$ has no critical pairs on root level, we know by Lemma 5.7 that $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ is confluent. As $\mathsf{G}(\mathsf{f}(\mathsf{h}(x)), x) \to_{\mathcal{R}} \mathsf{G}(\mathsf{a}, x)$, rewriting would transform this pair into the new pair $\mathsf{G}(\mathsf{d}, x) \to \mathsf{G}(\mathsf{a}, x)$. Now the dependency graph processor would delete this pair, since it is obviously not connected to itself in the dependency graph. Thus, we could falsely "prove" innermost termination.

The problem is that although the dependency pair was rewritten by a $\mathcal{Q}$-restricted step, it is no longer $\mathcal{Q}$-restricted if one instantiates x with b. So to guarantee that any reduction from $\mathsf{G}(\mathsf{f}(\mathsf{h}(x)), x)\sigma$ to an instantiated left-hand side of a dependency pair is also possible from $\mathsf{G}(\mathsf{a}, x)\sigma$, one needs to consider all critical pairs between $\ell \to r$ and the usable rules, and not just the critical pairs at the root level.

Example 5.14. In previous formulations of the rewriting processor instead of confluence and the condition that there are only trivial critical pairs between $\ell \to r$ and the usable rules, one required that the usable rules (as defined in the corresponding papers) of $t|_p$ are non-overlapping. Clearly, the requirement of non-overlapping usable rules guarantees confluence but it does not suffice to ensure the latter condition. The problem is that $\ell \to r$ is not necessarily usable w.r.t. Definition 3.24.[23] And indeed, these rewriting processors can become unsound if one uses arbitrary estimations of the semantic usable rules.

Let $\mathcal{P} = \{\mathsf{F}(\mathsf{g}(\mathsf{b})) \to \mathsf{F}(\mathsf{g}(\mathsf{h}(\mathsf{a})))\}$, let $\mathcal{R} = \{\mathsf{g}(\mathsf{h}(x)) \to \mathsf{c}, \mathsf{h}(\mathsf{a}) \to \mathsf{b}\}$, and let $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$. Then we have an infinite minimal chain, as $\mathsf{F}(\mathsf{g}(\mathsf{b})) \to_{\mathcal{P}} \mathsf{F}(\mathsf{g}(\mathsf{h}(\mathsf{a}))) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \mathsf{F}(\mathsf{g}(\mathsf{b})) \to_{\mathcal{P}} \ldots$. However, the first rule of $\mathcal{R}$ is not usable, i.e., not contained in $\mathcal{U}^{\{\mathsf{F}(\mathsf{g}(\mathsf{b}))\}}_{\mathcal{R},\mathcal{Q}}(\mathsf{F}(\mathsf{g}(\mathsf{h}(\mathsf{a}))))$. Thus, the second rule of $\mathcal{R}$ is the only usable rule which is not overlapping with itself. Using the rewrite processor with the first rule would result in a new DP problem with a new component $\mathcal{P}' = \{\mathsf{F}(\mathsf{g}(\mathsf{b})) \to \mathsf{F}(\mathsf{c})\}$ instead of $\mathcal{P}$ and this new DP problem is clearly finite.

One can argue that in this example the real problem is that the rewrite transformation from $\mathcal{P}$ to $\mathcal{P}'$ does not respect the evaluation strategy. And indeed, if one requires $t \xrightarrow{\mathcal{Q}}_{\mathcal{R},p} t'$ instead of $t \to_{\mathcal{R},p} t'$ then non-overlappingness of the usable rules suffices. However, this alternative formulation of the processor is more restrictive than the current version: whenever $t \xrightarrow{\mathcal{Q}}_{\mathcal{R},p} t'$ then $\ell \to r$ is already usable,[24] and non-overlappingness is a stronger requirement than confluence and absence of non-trivial critical pairs between $\ell \to r$ and the usable rules due to Lemma 5.7.

Example 5.15. This example shows why we defined a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ to be "infinite" if it is not finite *or if $\mathcal{R}$ is not $\mathcal{Q}$-terminating*, cf. Definition 2.13. The reason is that if "infinite" were defined as "not finite", then the rewriting processor would be incomplete, i.e., it could transform DP problems that are not infinite into problems with infinite chains, even if $\mathit{NF}(\mathcal{Q}) = \mathit{NF}(\mathcal{R})$. Let $\mathcal{P} = \{\mathsf{F}(x, x) \to \mathsf{F}(\mathsf{a}, \mathsf{g}(\mathsf{h}(x), x))\}$, let $\mathcal{R} = \{\mathsf{g}(x, y) \to y, \mathsf{h}(\mathsf{a}) \to \mathsf{h}(\mathsf{a})\}$, and let $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$. Obviously, $\mathcal{R}$ is not $\mathcal{Q}$-terminating. But there is no infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain as $\mathsf{F}(\mathsf{a}, \mathsf{g}(\mathsf{h}(x_1), x_1))\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} \mathsf{F}(x_2, x_2)\sigma$ implies $\sigma(x_2) = \mathsf{a}$.

---

[23] Note that $\ell \to r$ is usable for all previous definitions of usable rules. Hence our processor is not weaker than previous formulations of the rewriting processor.

[24] To be more precise, $\ell \to r$ is usable only if $s \in \mathit{NF}(\mathcal{Q})$. But if $s \notin \mathit{NF}(\mathcal{Q})$ then one can delete the pair by the dependency graph processors.

Thus $\mathsf{F}(\mathsf{a}, \mathsf{g}(\mathsf{h}(x_2), x_2))\sigma = \mathsf{F}(\mathsf{a}, \mathsf{g}(\mathsf{h}(\mathsf{a}), \mathsf{a}))$ can only be reduced by $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ to itself, but it does not unify with $\mathsf{F}(x_3, x_3)$.

However, the rewriting processor replaces the right-hand side $\mathsf{F}(\mathsf{a}, \mathsf{g}(\mathsf{h}(x), x))$ of the pair by $\mathsf{F}(\mathsf{a}, x)$. This results in $\mathcal{P}' = \{\mathsf{F}(x, x) \to \mathsf{F}(\mathsf{a}, x)\}$. Now there is clearly an infinite (minimal) $(\mathcal{P}', \mathcal{Q}, \mathcal{R})$-chain.

What is missing up to now is a criterion to estimate the $\mathcal{Q} - \mathcal{R}$ normal condition. We currently only know that the condition is always satisfied if $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ is exactly the innermost rewrite relation, i.e., if $\mathit{NF}(\mathcal{Q}) = \mathit{NF}(\mathcal{R})$. As argued above even when starting with the innermost relation we will encounter DP problems where $\mathit{NF}(\mathcal{Q}) \subset \mathit{NF}(\mathcal{R})$. For these problems we present the following estimation which has a quite similar structure to the improved estimation of usable rules in Definition 3.26. The first two cases are closures under subterms and closures under usable rules. The third case is the one where we detect that the $\mathcal{Q} - \mathcal{R}$ normal form condition is violated. If there is a redex $q\mu$ with $q \in \mathcal{Q}$ but where $q\mu$ is in normal form w.r.t. $\mathcal{R}$ then $q\mu$ is also in normal form w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ but not in normal form w.r.t. $\mathcal{Q}$. Note that this cannot happen if $\mathit{NF}(\mathcal{Q}) = \mathit{NF}(\mathcal{R})$ and hence, even with our estimation we can detect that the $\mathcal{Q} - \mathcal{R}$ normal form condition is always satisfied if we are in the innermost case.

Definition 5.16 (Estimation of the $\mathcal{Q} - \mathcal{R}$ Normal Form Condition)**.** *We define the* estimated $\mathcal{Q}-\mathcal{R}$ normal form condition *enfc as follows. The value of* $\mathit{enfc}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(f(t_1, \dots, t_n))$ *is false if one of the following conditions is valid.*

- $\mathit{enfc}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_i) = \mathit{false}$ *for some* $1 \le i \le n$
- *there is a rule* $f(\ell_1, \dots, \ell_n) = \ell \to r \in \mathcal{R}$ *and for* $\mathcal{S}' = \{\ell_1, \dots, \ell_n\}$ *the terms* ℓ *and* $f(\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1), \dots, \mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n))$ *are unifiable by some mgu* μ *such that* $\mathcal{S}\mu \cup \mathcal{S}'\mu \subseteq \mathit{NF}(\mathcal{Q})$ *and* $\mathit{enfc}^{\mathcal{S}'}_{\mathcal{R},\mathcal{Q}}(r) = \mathit{false}$
- *there is a term* $q \in \mathcal{Q}$*, the terms* q *and* $f(\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1), \dots, \mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n))$ *are unifiable with mgu* μ*, and both* $\mathcal{S}\mu \subseteq \mathit{NF}(\mathcal{Q})$ *and* $q\mu \in \mathit{NF}(\mathcal{R})$

If $\mathit{enfc}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$ *is not forced to evaluate to false by one of the previous conditions then* $\mathit{enfc}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) = \mathit{true}$*.*

The following theorem states that Definition 5.16 really estimates the $\mathcal{Q} - \mathcal{R}$ normal condition. Whenever the estimation says that a term satisfies the $\mathcal{Q} - \mathcal{R}$ normal condition then this really is the case.

Theorem 5.17 (Soundness of the Estimated $\mathcal{Q} - \mathcal{R}$ Normal Form Condition)**.** *Let* $\mathcal{R}$ *be a TRS and* $\mathcal{Q}$ *be a set of terms with* $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$*. For all terms* t *and sets of terms* $\mathcal{S}$ *with* $\mathcal{V}(t) \subseteq \mathcal{V}(\mathcal{S})$ *the* $\mathcal{Q} - \mathcal{R}$ *normal form condition* $\mathit{nfc}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$ *is satisfied if* $\mathit{enfc}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) = \mathit{true}$*.*

We illustrate the estimated $\mathcal{Q}-\mathcal{R}$ normal form condition using the TRS of Example 5.9.

Example 5.18. The TRS $\mathcal{R}$ in Example 5.9 consists of the following three rules and $\mathcal{Q} = \mathit{lhs}(\mathcal{R}) \cup \{\mathsf{h}(\mathsf{a})\}$.

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

We already detected that $\mathit{nfc}^{\{x\}}_{\mathcal{R},\mathcal{Q}}(\mathsf{f}(x))$ is not satisfied. This will also be the result of our estimation. To compute $\mathit{enfc}^{\{x\}}_{\mathcal{R},\mathcal{Q}}(\mathsf{f}(x))$ we first check the third case which would directly lead to the result *false*. However, since $\mathsf{f}(x) \notin \mathit{NF}(\mathcal{R})$ the third case cannot apply. But we also have to consider $\mathit{enfc}^{\{x\}}_{\mathcal{R},\mathcal{Q}}(\mathsf{g}(x))$ due to the second case. Again, the third case cannot be applied since $\mathsf{g}(x) \notin \mathit{NF}(\mathcal{R})$. However, we can once again apply the second case with the second and the third rule, such that we also have to consider $\mathit{enfc}^{\{\mathsf{h}(x)\}}_{\mathcal{R},\mathcal{Q}}(\mathsf{h}(x))$ and $\mathit{enfc}^{\{x\}}_{\mathcal{R},\mathcal{Q}}(\mathsf{h}(x))$, respectively. Now latter normal form condition must be evaluated to *false* since for $\mu = \{x/\mathsf{a}\}$ and $q = \mathsf{h}(\mathsf{a})$ we have $\mathsf{h}(x)\mu = q\mu$, $\mathcal{S}\mu = \{\mathsf{a}\} \subseteq \mathit{NF}(\mathcal{Q})$, and $q\mu = \mathsf{h}(\mathsf{a}) \in \mathit{NF}(\mathcal{R})$. Hence, we see that our initial normal form condition is not satisfied.

However, if one removes the third rule from $\mathcal{R}$ then $\mathit{enfc}^{\{x\}}_{\mathcal{R},\mathcal{Q}}(\mathsf{f}(x)) = \mathit{true}$. The reason is that we have to consider exactly the same conditions as before except for $\mathit{enfc}^{\{x\}}_{\mathcal{R},\mathcal{Q}}(\mathsf{h}(x))$, the condition which forced us to evaluate to *false*. Moreover, the only remaining check with the third case on condition $\mathit{enfc}^{\{\mathsf{h}(x)\}}_{\mathcal{R},\mathcal{Q}}(\mathsf{h}(x))$ will also not enforce the result *false* since for $\mu = \{x/\mathsf{a}\}$ we will instantiate $\mathcal{S} = \{\mathsf{h}(x)\}$ to $\{\mathsf{h}(\mathsf{a})\}$ which is not a subset of $\mathit{NF}(\mathcal{Q})$.

5.4. Narrowing

The basic idea of the *narrowing* processor is as follows. If for a pair $s \to t$ and all its succeeding pairs $u \to v$ the terms t and u are not unifiable, then there must be at least one reduction step from $t\sigma$ to $u\sigma$. Then one can already perform this first reduction step in $t\sigma$ and create corresponding new pairs, one for each possible reduction. As parts of the first redex may be in σ we must not only look at all reductions of t, but at all narrowings of t. If $\mathit{NF}(\mathcal{Q}) \nsubseteq \mathit{NF}(\mathcal{R})$ then it can also occur that the first reduction step is completely inside σ. To this end, a linearity restriction is integrated in the narrowing processor.

More formally, a term t *narrows* to a term t' at a position p via the substitution μ (denoted by $t \overset{\mathcal{Q}}{\rightsquigarrow}_{\mathcal{R}} t'$), if p is a non-variable position of t, μ is the most general unifier of $t|_p$ and ℓ for some rewrite rule $\ell \to r$ of $\mathcal{R}$, all proper subterms of $\ell\mu$ are in $\mathcal{Q}$-normal form and $t' = t\mu[r\mu]_p$. Narrowing is extended to pairs as follows: if $t \overset{\mathcal{Q}}{\rightsquigarrow}_{\mathcal{R}} t'$ via the substitution μ and $s\mu$ is a $\mathcal{Q}$-normal form, then the pair $s \to t$ narrows to the pair $s\mu \to t'$.

Before we present the narrowing processor, we discuss the idea of the integration of a position into the narrowing, so called *positional narrowing*. A main disadvantage of the previous narrowing processors of [AG00, GTS05a, GTSF06] is that they sometimes produce lots of new pairs. Remember the basic idea of the narrowing processor: whenever t and u are not unifiable then there must be at least one reduction step. But one can also look at subterms: whenever $t|_p$ and $u|_p$ are not unifiable then there also has to be a reduction. If moreover $t\sigma$ cannot be reduced strictly above p – to detect this one can use an estimated *Cap*-function – then the subterm $t|_p\sigma$ has to be reduced at least one step. Alternatively, if the subterm $t|_p$ is not in $\mathcal{Q}$-normal form then $t|_p\sigma$ has to be reduced, too. Thus, only the narrowings below p have to be considered which can be far less then all narrowings. This clearly increases efficiency, but when continuing our running example in Example 5.22 we will also demonstrate that less new pairs are sometimes crucial for a successful termination proof.

Theorem 5.19 (Positional Narrowing Processors). *Let Proc be a processor which transforms a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ with $\mathcal{P} = (N \uplus \{s \to t\}, E)$ into $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$ or into $\{(\mathcal{P}[s \to t/N'\}, \mathcal{Q}, \mathcal{R}, f)\}$. In the latter case, all of the following conditions must be satisfied.*

- *$t|_p$ is not in $\mathcal{Q}$-normal form or both $p \in \mathit{Pos}(\mathit{ECap}^{\{s\}}_{\mathcal{R},\mathcal{Q}}(t))$ and for all $(s \to t, u \to v) \in E$ the following two criteria are satisfied*
  - *$p \in \mathit{Pos}(u)$ and*
  - *if $t|_p$ and $u|_p$ are unifiable with an mgu μ, then $s\mu$ or $u\mu$ are not in $\mathcal{Q}$-normal form*
- *$N' = \{s' \to t' \mid s \to t$ narrows to $s' \to t'$ at a position below $p\}$*
- *$\mathcal{Q} = \varnothing$ or $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$*
- *if $\mathit{NF}(\mathcal{Q}) \nsubseteq \mathit{NF}(\mathcal{R})$, then t is linear*

Then the processor Proc is sound. And it is complete if $\mathcal{Q} = \varnothing$ or if the following two criteria are satisfied for every pair $s' \to t'$ in N' that has been built from narrowing $s \to t$ at position p' with rule $\ell \to r$ and unifier μ.

- *for $\mathcal{U} = \mathcal{EU}^{\{s'\}}_{\mathcal{R},\mathcal{Q}}(t\mu|_{p'})$ the relation $\xrightarrow{\mathcal{Q}}_{\mathcal{U}}$ is confluent and there are only trivial critical pairs between $\ell \to r$ and $\mathcal{U}$*
- *every proper subterm of $t\mu|_{p'}$ satisfies the $\mathcal{Q} - \mathcal{R}$ normal form condition w.r.t. $\{s'\}$*

Note that in the innermost case the $\mathcal{Q} - \mathcal{R}$ normal form condition is always satisfied. Hence, the completeness of the narrowing processor encompasses the completeness result of [GA01, Theorem 17] for innermost termination.

Compared to the rewriting processor, the narrowing processor has two advantages and one disadvantage. The disadvantage is that one has to build all narrowings whereas in the rewriting processor only one new pair is created. This was allowed in the rewriting processor due to the additional requirement of unique normal forms (which is ensured by confluent usable rules). A corresponding requirement for the narrowing processor is only needed to ensure completeness in the innermost case, which is the first advantage. The second advantage of the narrowing processor is that it can already be applied if only parts of the redex are available in a pair, whereas in the rewriting processor we need a complete redex. Both advantages will be needed to proceed in our running example. To summarize, the rewriting processor produces less pairs, but the narrowing processor is strictly more often applicable: whenever the rewriting processor is applicable then for some pair $s \to t$ the term t contains a redex at position p. But then one can apply the narrowing processor, too, since in that case $t|_p$ cannot be in $\mathcal{Q}$-normal form and the other requirements are trivially satisfied.

In the following examples we investigate possible improvements of the narrowing processor which turn out to be unsound. Then in the last example of this chapter, finally termination of our running example is proven. Thereby the power of our new narrowing processor is demonstrated and it is explained why previous versions of the narrowing processor [AG00, GTS05a, GTSF06] fail on this TRS.

Example 5.20. The narrowing processor has the restriction that either $\mathcal{Q} = \varnothing$ or $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$. This example illustrates that the processor is unsound without this restriction. Let $\mathcal{P} = \{\mathsf{F}(\mathsf{g}(x), \mathsf{a}) \to \mathsf{F}(\mathsf{g}(\mathsf{b}), \mathsf{h}(x))\}$, let $\mathcal{R} = \{\mathsf{b} \to \mathsf{a}, \mathsf{h}(\mathsf{a}) \to \mathsf{a}\}$, and let $\mathcal{Q} = \{\mathsf{g}(\mathsf{a})\}$. We obtain an infinite chain for $x = \mathsf{b}$ as

$$\mathsf{F}(\mathsf{g}(\mathsf{b}), \mathsf{a}) \to_{\mathcal{P}} \mathsf{F}(\mathsf{g}(\mathsf{b}), \mathsf{h}(\mathsf{b})) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \mathsf{F}(\mathsf{g}(\mathsf{b}), \mathsf{h}(\mathsf{a})) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \mathsf{F}(\mathsf{g}(\mathsf{b}), \mathsf{a}) \to_{\mathcal{P}} \dots$$

However, when applying the narrowing processor at the root position we obtain the new pairs $\mathsf{F}(\mathsf{g}(x),\mathsf{a}) \to \mathsf{F}(\mathsf{g}(\mathsf{a}),\mathsf{h}(x))$ and $\mathsf{F}(\mathsf{g}(\mathsf{a}),\mathsf{a}) \to \mathsf{F}(\mathsf{g}(\mathsf{b}),\mathsf{a})$. Then the second pair cannot occur in any chain as its left-hand side is not in $\mathcal{Q}$-normal form. And from the first pair one cannot build chains with more than one element: the right-hand side contains the subterm $\mathsf{g}(\mathsf{a})$, which cannot be reduced by $\mathcal{R}$ and which is not in $\mathcal{Q}$-normal form. Thus, the right-hand side can never be reduced to a $\mathcal{Q}$-normal form.

This shows that even in the case where all rules and pairs are linear and where we do not make use of the new feature to choose a position p we cannot allow arbitrary sets $\mathcal{Q}$.

Example 5.21. A severe restriction in the narrowing processor is the linearity of t in the case $NF(\mathcal{Q}) \not\subseteq NF(\mathcal{R})$. One obvious question is whether it is possible to just require linearity of the subterm $t|_p$. Unfortunately, the following example demonstrates that this relaxation of the requirements is unsound. Let $\mathcal{P} = \{\mathsf{F}(\mathsf{a},\mathsf{b},x) \to \mathsf{F}(\mathsf{g}(x),x,x) = t\}$, let $\mathcal{R} = \{\mathsf{b} \to \mathsf{a}, \mathsf{g}(\mathsf{a}) \to \mathsf{a}\}$, and let $\mathcal{Q} = \varnothing$. We obtain an infinite chain for $x = \mathsf{b}$ as

$$\mathsf{F}(\mathsf{a},\mathsf{b},\mathsf{b}) \to_{\mathcal{P}} \mathsf{F}(\mathsf{g}(\mathsf{b}),\mathsf{b},\mathsf{b}) \overset{\mathcal{Q}}{\to}_{\mathcal{R}} \mathsf{F}(\mathsf{g}(\mathsf{a}),\mathsf{b},\mathsf{b}) \overset{\mathcal{Q}}{\to}_{\mathcal{R}} \mathsf{F}(\mathsf{a},\mathsf{b},\mathsf{b}) \to_{\mathcal{P}} \ldots$$

When choosing $p = 1$ then all requirements but the linearity of t are satisfied. However, the subterm $t|_1 = \mathsf{g}(x)$ is linear. By narrowing the only pair in $\mathcal{P}$ we only obtain one new pair $\mathsf{F}(\mathsf{a},\mathsf{b},\mathsf{a}) \to \mathsf{F}(\mathsf{a},\mathsf{a},\mathsf{a})$ and the resulting DP problem is clearly finite. Hence, linearity must be required for the whole right-hand side t.

Example 5.22. We recapitulate some pairs and rules

$$\begin{array}{rcll}
\mathsf{QUOT}(x,y,z) & \to & \mathsf{IF}(\mathsf{ge}(\mathsf{0},y),\mathsf{ge}(x,y),\mathsf{minus}(x,y),y,\mathsf{plus}(z,\mathsf{s}(\mathsf{0}))) & (83)\\
\mathsf{IF}(\mathsf{ff},\mathsf{tt},x,y,z) & \to & \mathsf{IF2}(\mathsf{tt},x,y,z) & (91)\\
\mathsf{IF2}(\mathsf{tt},x,y,z) & \to & \mathsf{QUOT}(x,y,z) & (85)\\
\mathsf{p}(\mathsf{s}(x)) & \to & x & (61)\\
\mathsf{p}(\mathsf{0}) & \to & \mathsf{s}(\mathsf{0}) & (62)\\
\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{minus}(x,y) & (67)\\
\mathsf{minus}(\mathsf{0},x) & \to & \mathsf{0} & (68)\\
\mathsf{minus}(x,\mathsf{0}) & \to & x & (69)\\
\mathsf{minus}(x,x) & \to & \mathsf{0} & (70)\\
\mathsf{ge}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{ge}(x,y) & (71)\\
\mathsf{ge}(\mathsf{0},\mathsf{s}(x)) & \to & \mathsf{ff} & (72)\\
\mathsf{ge}(x,\mathsf{0}) & \to & \mathsf{tt} & (73)
\end{array}$$

and continue to prove termination of Example 5.11, where we still have to solve the DP problem $\mathcal{D}_{11} = (\{(83),(91),(85)\},\mathcal{Q},\{(61)-(73)\},\mathbf{m})$. Note that without the conditions $y > 0$ and $x \geq y$ the DP problem would not be finite. The reason is that then $x - y$ is not necessarily smaller than x: if $y = 0$ then $x - y = x - 0 = x$ and if $x < y$ is allowed then for $x = 0$ we obtain $x - y = 0 - y = 0 = x$. This is the reason why negative polynomial orders fail on this DP problem. They must prove $x > \mathsf{minus}(x,y)$ for all natural numbers x and y, and they cannot integrate the conditions on x and y.

We transform the pair (83) by the narrowing processor at position 1. Hence, we first use the fact that $y > 0$ must hold, and obtain the following new pairs.

$$\begin{array}{rcll}
\mathsf{QUOT}(x,\mathsf{s}(y),z) & \to & \mathsf{IF}(\mathsf{ff},\mathsf{ge}(x,\mathsf{s}(y)),\mathsf{minus}(x,\mathsf{s}(y)),\mathsf{s}(y),\mathsf{plus}(z,\mathsf{s}(\mathsf{0}))) & (96)\\
\mathsf{QUOT}(x,\mathsf{0},z) & \to & \mathsf{IF}(\mathsf{tt},\mathsf{ge}(x,\mathsf{0}),\mathsf{minus}(x,\mathsf{0}),\mathsf{0},\mathsf{plus}(z,\mathsf{s}(\mathsf{0}))) & (97)
\end{array}$$

By the processors based on the dependency graph the second new pair can be deleted and the DP problem $(\{(96),(91),(85)\}, \mathcal{Q}, \{(61)-(73)\}, \mathbf{m})$ remains to be solved. Now we have integrated the knowledge that the second argument of QUOT is larger than 0. Note that here we needed the new improvement of *positional* narrowing. With the narrowing processor of [AG00, GTS05a, GTSF06] we would have to build all narrowings of (83) which results in the following additional pairs.

$$\begin{aligned}
\mathsf{QUOT}(\mathsf{s}(x),\mathsf{s}(y),z) &\rightarrow \mathsf{IF}(\mathsf{ge}(\mathsf{0},\mathsf{s}(y)),\mathsf{ge}(x,y),\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)),\mathsf{s}(y),\mathsf{plus}(z,\mathsf{s}(\mathsf{0})))\\
\mathsf{QUOT}(\mathsf{0},\mathsf{s}(y),z) &\rightarrow \mathsf{IF}(\mathsf{ge}(\mathsf{0},\mathsf{s}(y)),\mathsf{ff},\mathsf{minus}(\mathsf{0},\mathsf{s}(y)),\mathsf{s}(y),\mathsf{plus}(z,\mathsf{s}(\mathsf{0})))\\
\mathsf{QUOT}(x,\mathsf{0},z) &\rightarrow \mathsf{IF}(\mathsf{ge}(\mathsf{0},\mathsf{0}),\mathsf{tt},\mathsf{minus}(x,\mathsf{0}),\mathsf{0},\mathsf{plus}(z,\mathsf{s}(\mathsf{0})))\\
\mathsf{QUOT}(\mathsf{s}(x),\mathsf{s}(y),z) &\rightarrow \mathsf{IF}(\mathsf{ge}(\mathsf{0},y),\mathsf{ge}(\mathsf{s}(x),\mathsf{s}(y)),\mathsf{minus}(x,y),\mathsf{s}(y),\mathsf{plus}(z,\mathsf{s}(\mathsf{0})))\\
\mathsf{QUOT}(\mathsf{0},y,z) &\rightarrow \mathsf{IF}(\mathsf{ge}(\mathsf{0},y),\mathsf{ge}(\mathsf{0},y),\mathsf{0},y,\mathsf{plus}(z,\mathsf{s}(\mathsf{0})))\\
\mathsf{QUOT}(x,\mathsf{0},z) &\rightarrow \mathsf{IF}(\mathsf{ge}(\mathsf{0},\mathsf{0}),\mathsf{ge}(x,\mathsf{0}),x,\mathsf{0},\mathsf{plus}(z,\mathsf{s}(\mathsf{0})))\\
\mathsf{QUOT}(x,x,z) &\rightarrow \mathsf{IF}(\mathsf{ge}(\mathsf{0},x),\mathsf{ge}(x,x),\mathsf{0},x,\mathsf{plus}(z,\mathsf{s}(\mathsf{0})))\\
\mathsf{QUOT}(x,y,\mathsf{s}(z)) &\rightarrow \mathsf{IF}(\mathsf{ge}(\mathsf{0},y),\mathsf{ge}(x,y),\mathsf{minus}(x,y),y,\mathsf{plus1}(\mathsf{s}(z),\mathsf{s}(\mathsf{0})))
\end{aligned}$$

Thus, we see that positional narrowing produces far less new pairs than non-positional narrowing. That this is sometimes crucial can be demonstrated with the last pair. It does not reveal any information about x and y which is required to finally solve the DP problem. And this information cannot be gained by a repeated application of the (non-positional) narrowing processor. The reason is that essentially, by narrowing we evaluate the accumulator more and more, which is not connected to the variables x and y that are used in the test.

$$\begin{aligned}
& \mathsf{QUOT}(x,y,\mathsf{s}(z)) \rightarrow \mathsf{IF}(\mathsf{ge}(\mathsf{0},y),\mathsf{ge}(x,y),\mathsf{minus}(x,y),y,\mathsf{plus1}(\mathsf{s}(z),\mathsf{s}(\mathsf{0})))\\
\overset{\mathcal{Q}}{\rightsquigarrow}_{\mathcal{R}}\ & \mathsf{QUOT}(x,y,\mathsf{s}(z)) \rightarrow \mathsf{IF}(\mathsf{ge}(\mathsf{0},y),\mathsf{ge}(x,y),\mathsf{minus}(x,y),y,\mathsf{plus2}(\mathsf{p}(\mathsf{s}(z)),\mathsf{s}(\mathsf{s}(\mathsf{0}))))\\
\overset{\mathcal{Q}}{\rightsquigarrow}_{\mathcal{R}}\ & \mathsf{QUOT}(x,y,\mathsf{s}(z)) \rightarrow \mathsf{IF}(\mathsf{ge}(\mathsf{0},y),\mathsf{ge}(x,y),\mathsf{minus}(x,y),y,\mathsf{plus2}(z,\mathsf{s}(\mathsf{s}(\mathsf{0}))))\\
\overset{\mathcal{Q}}{\rightsquigarrow}_{\mathcal{R}}\ & \mathsf{QUOT}(x,y,\mathsf{s}(z)) \rightarrow \mathsf{IF}(\mathsf{ge}(\mathsf{0},y),\mathsf{ge}(x,y),\mathsf{minus}(x,y),y,\mathsf{plus}(z,\mathsf{s}(\mathsf{s}(\mathsf{0}))))\\
\overset{\mathcal{Q}}{\rightsquigarrow}_{\mathcal{R}}^{*} & \mathsf{QUOT}(x,y,\mathsf{s}(\mathsf{s}(z))) \rightarrow \mathsf{IF}(\mathsf{ge}(\mathsf{0},y),\mathsf{ge}(x,y),\mathsf{minus}(x,y),y,\mathsf{plus}(z,\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0})))))\\
\overset{\mathcal{Q}}{\rightsquigarrow}_{\mathcal{R}}^{*} & \ldots
\end{aligned}$$

We continue to prove termination with positional narrowing. As stated above, we must also use the condition $x \geq \mathsf{s}(y)$ to prohibit situations where x may be instantiated by $\mathsf{0}$. Therefore, we narrow the new pair (96), but this time at position 2. Note that without the application of the forward instantiation processor in Example 5.6 the narrowing processor would not be applicable, as the subterm $\mathsf{ge}(x,\mathsf{s}(y))$ of (96) unifies with the variable argument b of the left-hand side of (84) and the mgu instantiates both left-hand sides of (96) and (84) to $\mathcal{Q}$-normal forms. But by the forward instantiation processor the variable b was instantiated by tt, which does not unify with $\mathsf{ge}(x,\mathsf{s}(y))$ any more.

We obtain the following new pairs.

$$\begin{aligned}
\mathsf{QUOT}(\mathsf{s}(x),\mathsf{s}(y),z) &\rightarrow \mathsf{IF}(\mathsf{ff},\mathsf{ge}(x,y),\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)),\mathsf{s}(y),\mathsf{plus}(z,\mathsf{s}(\mathsf{0}))) && (98)\\
\mathsf{QUOT}(\mathsf{0},\mathsf{s}(y),z) &\rightarrow \mathsf{IF}(\mathsf{ff},\mathsf{ff},\mathsf{minus}(\mathsf{0},\mathsf{s}(y)),\mathsf{s}(y),\mathsf{plus}(z,\mathsf{s}(\mathsf{0}))) && (99)
\end{aligned}$$

As before by the dependency graph processors one can delete the second new pair and simplify the resulting DP problem to $(\{(98),(91),(85)\}, \mathcal{Q}, \{(61)-(73)\}, \mathbf{m})$. This remaining

DP problem can now be solved by the reduction pair processor of Theorem 4.27[25] using the negative polynomial order $\mathcal{P}ol$ with $\mathcal{P}ol(\mathsf{QUOT}(x,y,z)) = \mathcal{P}ol(\mathsf{IF}(b_1,b_2,x,y,z)) = \mathcal{P}ol(\mathsf{IF2}(b,x,y,z)) = x$, $\mathcal{P}ol(\mathsf{0}) = 0$, $\mathcal{P}ol(\mathsf{s}(x)) = x+1$, and $\mathcal{P}ol(\mathsf{minus}(x,y)) = max(0,x-y)$. For this polynomial order – which corresponds to an argument filter π with $\pi(\mathsf{IF}) = 3$ and $\pi(\mathsf{QUOT}) = 1$ – only the four minus-rules are usable. Note however that the automation of negative polynomial orders in [HM07] is not able to handle this DP problem due to the constraint of rule (68). But even the embedding order would satisfy all constraints if one would rewrite the subterm $\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y))$ to $\mathsf{minus}(x,y)$ in pair (98). Unfortunately, the rewriting processor of Theorem 5.10 is not applicable if one uses Lemma 5.7 as confluence criterion: the usable rules (67) and (70) have the non-trivial critical pair $(\mathsf{0},\mathsf{minus}(x,x))$.

This is in contrast to the narrowing processor, which can narrow (98) at position 3 since the corresponding subterm $\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y))$ is not in $\mathcal{Q}$-normal form. We obtain the following two new pairs.

$$\mathsf{QUOT}(\mathsf{s}(x),\mathsf{s}(y),z) \to \mathsf{IF}(\mathsf{ff},\mathsf{ge}(x,y),\mathsf{minus}(x,y),\mathsf{s}(y),\mathsf{plus}(z,\mathsf{s}(\mathsf{0}))) \tag{100}$$
$$\mathsf{QUOT}(\mathsf{s}(x),\mathsf{s}(x),z) \to \mathsf{IF}(\mathsf{ff},\mathsf{ge}(x,x),\mathsf{0},\mathsf{s}(x),\mathsf{plus}(z,\mathsf{s}(\mathsf{0}))) \tag{101}$$

This time one cannot delete any new pair by the dependency graph processors. Hence, the resulting DP problem is $(\{(100),(101),(91),(85)\},\mathcal{Q},\{(61)-(73)\},\mathbf{m})$. But one can delete (100) and (101) by the reduction pair processor of Theorem 4.27. To this end we use the polynomial order with $\mathcal{P}ol(\mathsf{minus}(x,y)) = \mathcal{P}ol(\mathsf{QUOT}(x,y,z)) = \mathcal{P}ol(\mathsf{IF}(b_1,b_2,x,y,z)) = \mathcal{P}ol(\mathsf{IF2}(b,x,y,z)) = x$, $\mathcal{P}ol(\mathsf{0}) = 0$, and $\mathcal{P}ol(\mathsf{s}(x)) = x+1$. The remaining DP problem can then be solved by the processors based on the dependency graph. Thus, to prove termination of the TRS in Example 5.1 (which is not $\mathcal{C}_\varepsilon$-terminating) we only needed the transformations of this chapter and some simple linear polynomial orders for the reduction pair processors of the previous chapter.

Summary of Chapter 5

In this chapter we have seen four processors which can transform the pairs of DP problems. These are often crucial for a successful termination proof, especially if tests and selectors are used in a DP problem instead of pattern matching.

Note that there is a problem when automating these processors. Although each transformation can easily be automated, it is unclear how often one should apply these processors, as each one can be applied infinitely many times. To solve this problem, we have published a successful heuristic in [GTSF06, Definition 33].

One of our contributions in this chapter is the development of the new forward instantiation processor which is published in [GTSF06]. But even for the narrowing, rewriting, and instantiation transformations – which have been presented in [AG00, GA01] – we have provided substantial improvements. All of them are now applicable at any time during a termination proof, we have generalized them to $\mathcal{Q}$-restricted rewriting, one can use them with arbitrary estimations of usable rules and of *Cap*, the requirements to apply the processors have been relaxed, and for narrowing we get less new pairs due to our integration of a position. Note that some of these contributions have already been published by us in [GTS05a, GTSF06].

---

[25] The other reduction pair processors of Section 4 are not applicable. Theorem 4.32 is not applicable as the given polynomial order is not $\mathcal{C}_\varepsilon$-compatible and Theorem 4.2 cannot handle the constraints of the rules (61) and (62).

Related work can also be found in [Zan05a]. There it is allowed to rewrite right-hand sides on the level of TRSs in a similar way to our rewriting processor. Both techniques require that there is no critical pair between the rule $\ell \to r$ that is used for rewriting and the other rules. However, there are some important differences. Our requirement of confluence of the usable rules is replaced by confluence of the TRS that consists only of the rule $\ell \to r$. And even more important, the rewriting technique in [Zan05a] can be applied for full rewriting. But in [Zan05a] there are the additional requirements of left-linearity and non-erasingness, i.e., $\mathcal{V}(\ell) = \mathcal{V}(r)$. Whereas left-linearity is often satisfied, non-erasingness is a real restriction, e.g., one cannot rewrite with selectors of the form $\mathsf{head}(\mathsf{cons}(x, xs)) \to x$. We strongly conjecture that one can allow erasing rules when lifting the technique of [Zan05a] to the DP framework, making it an interesting future work.

6. Processors for Applicative Rewriting

In this chapter we will present a method for termination analysis of untyped higher-order functions which do not use λ-abstraction.[26] Due to the absence of λ, such functions can be represented in curried form as *applicative* first-order TRSs (cf. e.g., [KKSV96]). A signature $\mathcal{F}$ is *applicative* if it only contains constants and a binary symbol $'$ for function application. Moreover, any TRS or DP problem over $\mathcal{F}$ is called *applicative*. So instead of the higher-order rule

$$\mathsf{map}(\alpha, \mathsf{cons}(x, xs)) \quad \to \quad \mathsf{cons}(\alpha(x), \mathsf{map}(\alpha, xs))$$

we use the first-order rule

$$'(\,'(\mathsf{map}, \alpha), \,'(\,'(\mathsf{cons}, x), xs)) \quad \to \quad '(\,'(\mathsf{cons}, \,'(\alpha, x)), \,'(\,'(\mathsf{map}, \alpha), xs))$$

where there is no variable-application $\alpha(x)$ any more. To ease readability, we use $'$ as an infix-symbol and we let $'$ associate to the left. Then this rule can be written as

$$\mathsf{map}'\alpha'(\mathsf{cons}'x'xs) \quad \to \quad \mathsf{cons}'(\alpha'x)'(\mathsf{map}'\alpha'xs)$$

This is very similar to the usual notation of higher-order functions where application is just denoted by juxtaposition, i.e., here one would write

$$\mathsf{map}\ \alpha\ (\mathsf{cons}\ x\ xs) \quad \to \quad \mathsf{cons}\ (\alpha\ x)\ (\mathsf{map}\ \alpha\ xs)$$

Example 6.1. As running example of this chapter we take the following TRS which computes the well-known map-function that takes a function and a list as input and applies the function on each element of the list. Moreover, we also add first-order functions for subtraction and division in applicative form. Note that a direct termination proof with simplification orders is impossible.

$$\begin{aligned}
\mathsf{map}'\alpha'\mathsf{nil} &\to \mathsf{nil} && (102)\\
\mathsf{map}'\alpha'(\mathsf{cons}'x'xs) &\to \mathsf{cons}'(\alpha'x)'(\mathsf{map}'\alpha'xs) && (103)\\
\mathsf{minus}'x'\mathsf{0} &\to x && (104)\\
\mathsf{minus}'(\mathsf{s}'x)'(\mathsf{s}'y) &\to \mathsf{minus}'x'y && (105)\\
\mathsf{div}'\mathsf{0}'(\mathsf{s}'y) &\to \mathsf{0} && (106)\\
\mathsf{div}'(\mathsf{s}'x)'(\mathsf{s}'y) &\to \mathsf{s}'(\mathsf{div}'(\mathsf{minus}'x'y)'(\mathsf{s}'y)) && (107)
\end{aligned}$$

---

[26] If λ does not occur in left-hand sides then one can always eliminate λ by introducing corresponding named functions. For example, instead of using the rule

$$\mathsf{f}\ x\ xs \to \mathsf{map}\ (\lambda y.y + x * y)\ xs$$

one can add an additional function g and then use the following two rules.

$$\begin{aligned}
\mathsf{f}\ x\ xs &\to \mathsf{map}\ (\mathsf{g}\ x)\ xs\\
\mathsf{g}\ x\ y &\to y + x * y
\end{aligned}$$

In this TRS the only defined function symbol is $'$ while all remaining function symbols are constructors. Therefore, we obtain the following set of dependency pairs. Here, $\sharp$ is the tuple-symbol of $'$ and we use $\sharp$ as an infix-symbol, too.

$$
\begin{align}
\mathsf{map}'\alpha \,\sharp\, (\mathsf{cons}'x'xs) &\to \mathsf{cons}'(\alpha'x) \,\sharp\, (\mathsf{map}'\alpha'xs) \tag{108}\\
\mathsf{map}'\alpha \,\sharp\, (\mathsf{cons}'x'xs) &\to \mathsf{cons} \,\sharp\, (\alpha'x) \tag{109}\\
\mathsf{map}'\alpha \,\sharp\, (\mathsf{cons}'x'xs) &\to \alpha \,\sharp\, x \tag{110}\\
\mathsf{map}'\alpha \,\sharp\, (\mathsf{cons}'x'xs) &\to \mathsf{map}'\alpha \,\sharp\, xs \tag{111}\\
\mathsf{minus}'(\mathsf{s}'x) \,\sharp\, (\mathsf{s}'y) &\to \mathsf{minus}'x \,\sharp\, y \tag{112}\\
\mathsf{minus}'(\mathsf{s}'x) \,\sharp\, (\mathsf{s}'y) &\to \mathsf{minus} \,\sharp\, x \tag{113}\\
\mathsf{div}'(\mathsf{s}'x) \,\sharp\, (\mathsf{s}'y) &\to \mathsf{s} \,\sharp\, (\mathsf{div}'(\mathsf{minus}'x'y)'(\mathsf{s}'y)) \tag{114}\\
\mathsf{div}'(\mathsf{s}'x) \,\sharp\, (\mathsf{s}'y) &\to \mathsf{div}'(\mathsf{minus}'x'y) \,\sharp\, (\mathsf{s}'y) \tag{115}\\
\mathsf{div}'(\mathsf{s}'x) \,\sharp\, (\mathsf{s}'y) &\to \mathsf{div} \,\sharp\, (\mathsf{minus}'x'y) \tag{116}\\
\mathsf{div}'(\mathsf{s}'x) \,\sharp\, (\mathsf{s}'y) &\to \mathsf{minus}'x \,\sharp\, y \tag{117}\\
\mathsf{div}'(\mathsf{s}'x) \,\sharp\, (\mathsf{s}'y) &\to \mathsf{minus} \,\sharp\, x \tag{118}
\end{align}
$$

Since $\mathcal{R}$ falls in a class of TRSs where innermost termination is equivalent to termination we choose $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$. As usual we first transform the initial DP problem $(\mathit{DP}(\mathcal{R}), \mathcal{Q}, \mathcal{R}, \mathbf{m})$ by the dependency graph processors of Theorems 3.3 and 3.4. We use the graph estimation of Definition 3.9 together with the estimated *Cap*-function *ICap*. This results in three new DP problems corresponding to the three different functions map, minus, and div: $(\{(110), (111)\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$, $(\{(112)\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$, and $(\{(115)\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$. This corresponds to the real dependency graph.

Note that here we really need the estimated *Cap*-function *ICap* which is based on unification. With previous dependency graph estimations which are based on the *Cap*-function of [AG00] that just looks at the root symbols we obtain a different result. In that case the estimation cannot delete any edge between two dependency pairs of $\{(108), (110), (111), (112), (115), (117)\}$. Thus, instead of a clear separation one would have to prove termination of all functions together. Note that this is not possible using all processors of Chapters 3 and 4 if one uses quasi-simplification orders. (Of course, we exclude the processor to estimate the dependency graph with *ICap*.) Moreover, even the processors of this chapters will not be successful.

The problem is that if an estimated *Cap*-function *ECap* only looks at the root symbols then it must replace every term $t_1{}'t_2$ by a fresh variable. But as nearly every term is built by $'$ this obviously is a coarse estimation. Consider the right-hand side of (108). Then $\mathit{ECap}^{\{\mathsf{map}'\alpha \,\sharp\, (\mathsf{cons}'x'xs)\}}_{\mathcal{R},\mathcal{Q}}(\mathsf{cons}'(\alpha'x) \,\sharp\, (\mathsf{map}'\alpha'xs)) = y \,\sharp\, z$ since the subterms $\mathsf{cons}'(\alpha'x)$ and $\mathsf{map}'\alpha'xs$ have the defined symbol $'$ as root. And as the term $y \,\sharp\, z$ unifies with the left-hand sides of every dependency pair, there are connections from (108) to every other dependency pair if one uses *ECap*.

In contrast, $\mathit{ICap}^{\{\mathsf{map}'\alpha \,\sharp\, (\mathsf{cons}'x'xs)\}}_{\mathcal{R},\mathcal{Q}}(\mathsf{cons}'(\alpha'x) \,\sharp\, (\mathsf{map}'\alpha'xs)) = \mathsf{cons}'y \,\sharp\, z$. The reason is that both the terms $\alpha'x$ and $\mathsf{map}'\alpha'xs$ unify with left-hand sides of $\mathcal{R}$ and therefore are replaced by fresh variables. However, the term $\mathsf{cons}'y$ does not unify with any left-hand side of $\mathcal{R}$ and thus, is not replaced by a fresh variable. And since $\mathsf{cons}'y \,\sharp\, z$ does not unify with any left-hand side of $\mathit{DP}(\mathcal{R})$, the dependency pair (108) has no outgoing edge to another dependency pair.

As we are in the innermost case we can often simplify the remaining DP problems by the usable rules processor of Theorem 3.25. Here, we obtain a similar result as for the dependency graph estimation. For each of the three DP problems all rules of $\mathcal{R}$ are usable if one uses the estimation of [AG00]. Again the reason is that in [AG00] only the root symbols were analyzed. In that case if a right-hand side contains the application symbol $'$ then all $'$-rules are usable. As every rule of $\mathcal{R}$ has as root the application symbol these are all rules.

But if we use the improved usable rules estimation $\mathcal{IU}$ of Definition 3.26 with *ICap* then we obtain the real usable rules for all three DP problems: $\mathcal{IU}((\{(110),(111)\},\mathcal{Q},\mathcal{R})) = \mathcal{IU}((\{(112)\},\mathcal{Q},\mathcal{R})) = \varnothing$ and $\mathcal{IU}((\{(115)\},\mathcal{Q},\mathcal{R})) = \{(104),(105)\}$.

Two of the resulting DP problems can be solved by any reduction pair processor of Section 4. For $\mathcal{D}_{12} = (\{(110),(111)\},\mathcal{Q},\varnothing,\mathbf{m})$ and $(\{(112)\},\mathcal{Q},\varnothing,\mathbf{m})$ it suffices to use the embedding order and no argument filter to delete all pairs (110), (111), and (112). However, the remaining DP problem $\mathcal{D}_{13} = (\{(115)\},\mathcal{Q},\{(104),(105)\},\mathbf{m})$ cannot be solved by any reduction pair processor when using a quasi-simplification order. Since it does not contain a map-rule any more, one would like to change it back to conventional functional form. Then it could be replaced by the DP problem $\mathcal{D}_{14} = (\mathcal{P}',\mathcal{Q}',\mathcal{R}',\mathbf{m})$ with $\mathcal{P}' = \{\mathsf{DIV}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{DIV}(\mathsf{minus}(x,y),\mathsf{s}(y))\}$, the set $\mathcal{Q}' = \{\mathsf{map}(\alpha,\mathsf{nil}), \mathsf{map}(\alpha,\mathsf{cons}(x,xs)), \mathsf{minus}(x,\mathsf{0}), \mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)), \mathsf{div}(\mathsf{0},\mathsf{s}(y)), \mathsf{div}(\mathsf{s}(x),\mathsf{s}(y))\}$, and the TRS $\mathcal{R}' = \{\mathsf{minus}(x,\mathsf{0}) \to x, \mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(x,y)\}$. This DP problem is then easy to solve: for example, the constraints of any reduction pair processor of Section 4 are satisfied by the polynomial order which maps $\mathsf{s}(x)$ to $x+1$, $\mathsf{minus}(x,y)$ to x, and every other symbol to the sum of its arguments. Thus, termination could immediately be proved automatically.

This chapter is organized as follows. In Section 6.1 it is shown how to transform DP problems like $\mathcal{D}_{13}$ into $\mathcal{D}_{14}$. Thereby, we extend the work of [KKSV96] in two ways. Whereas in [KKSV96] it is shown that one can do this transformation on the level of TRSs, we perform it on DP problems. And moreover, in [KKSV96] only full rewriting is considered whereas we consider $\mathcal{Q}$-restricted rewriting. Our results also extends the work of [GTS05b], since there, only full- and innermost rewriting are considered, and moreover, minimality is not investigated. Here, we will provide a new example to prove that even for full termination minimality cannot be preserved. To overcome this problem, in Section 6.2 we combine the transformation from applicative to functional form with the reduction pair processors which are based on needed rules. In that way, minimality can be preserved. Finally, needed rules w.r.t. argument filters are integrated in Section 6.3.

6.1. From Applicative to Functional Form

Some applicative DP problems can be transformed (back) to ordinary functional form. In particular, this holds for problems resulting from first-order functions (encoded by currying). This transformation is advantageous: e.g., the reduction pair processors in Chapter 4 are significantly more powerful for DP problems in functional form, since standard reduction orders focus on the root symbol when comparing terms.

Now we characterize those applicative TRSs which correspond to first-order functions and can be translated into functional form. In these TRSs, for any function symbol f there is a number n (called its *applicative arity* or just *a-arity*) such that f only occurs in terms of the form $f\,'t_1\,'\ldots\,'t_n$. So there are no applications with too few or too many arguments. Moreover, there are no terms $x\,'t$ where the first argument of $'$ is a variable. Definition 6.2 extends this idea from TRSs to DP problems.

As we have seen in Example 6.1, in DP problems there is not just one application symbol, but usually there are two application symbols $'$ and $^\sharp$, and the remaining symbols are constants. We generalize from two application symbols to a set of function symbols.

For the remainder of this chapter we fix the applicative signature to $\mathcal{F} \cup \mathcal{F}'$ where $\mathcal{F}$ only contains constants and $\mathcal{F}'$ consists of binary function symbols. Elements of $\mathcal{F}$ are usually denoted with $f, g, \dots$ and elements of $\mathcal{F}'$ are ${}^{?_1}, {}^{?_2}, \dots$ and are always used in infix-notation. We also fix a function $\textit{a-ar} : \mathcal{F} \to \mathbb{N}$ which specifies the applicative arities for the symbols in $\mathcal{F}$. Additionally we introduce the functional signature $\mathcal{F}_{\mathit{func}} = \{f_{?_1 \dots ?_n} \mid f \in \mathcal{F}, \textit{a-ar}(f) = n\}$ where each $f_{?_1 \dots ?_n}$ has arity n. To increase readability in examples we often identify $f_{\prime \dots \prime}$ with f and $f_{\prime \dots \prime \sharp}$ with F.

Definition 6.2 (Proper)**.** *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ be an applicative DP problem. A term t is* proper *iff $t \in \mathcal{V}$ or $t = f^{?_1} t_1^{?_2} \dots {}^{?_n} t_n$ where in the latter case, $\textit{a-ar}(f) = n$ and all t_i are proper. Moreover, a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ is* proper *iff all terms in $\mathcal{P}$, $\mathcal{Q}$, and $\mathcal{R}$ are proper.*

The remaining DP problem $\mathcal{D}_{13}$ of Example 6.1 for div is proper. Here, minus, div, cons, and map have a-arity 2, s has a-arity 1, and 0 and nil have a-arity 0. But the problem $\mathcal{D}_{12}$ for map is not proper as (110) contains the subterm $\alpha^\sharp x$ with $\alpha \in \mathcal{V}$. Note that $\mathcal{D}_{13}$ would not be proper if $\mathcal{Q}$ were a set of rules, since then $\mathcal{Q}$ would still contain the recursive, non-proper map-rule. Hence, here we benefit from $\mathcal{Q}$ being a set of terms.

The following transformation translates proper terms from applicative to functional form. To this end, $f^{?_1} t_1^{?_2} \dots {}^{?_n} t_n$ is replaced by $f_{?_1 \dots ?_n}(\dots)$, where $\textit{a-ar}(f) = n$ and $f_{?_1 \dots ?_n}$ is a function symbol of the functional signature $\mathcal{F}_{\mathit{func}}$. In this way, $\mathcal{D}_{13}$ is transformed into $\mathcal{D}_{14}$ in Example 6.1.

Definition 6.3 ($\mathcal{A}$-Transformation)**.** *$\mathcal{A}$ maps every proper term of $\mathcal{T}(\mathcal{F} \cup \mathcal{F}', \mathcal{V})$ to a term of $\mathcal{T}(\mathcal{F}_{\mathit{func}}, \mathcal{V})$:*

- *$\mathcal{A}(x) = x$ for all $x \in \mathcal{V}$*
- *$\mathcal{A}(f^{?_1} t_1^{?_2} \dots {}^{?_n} t_n) = f_{?_1 \dots ?_n}(\mathcal{A}(t_1), \dots, \mathcal{A}(t_n))$ for all $f \in \mathcal{F}$*

$\mathcal{A}$ is lifted to sets of terms, rules, TRSs, pair-graphs, and substitutions by applying $\mathcal{A}$ component-wise.

In the following, we say that a substitution σ is *proper* if $\sigma(x)$ is proper for all $x \in \mathcal{V}$. Moreover, let $\mathcal{T}_{\mathit{proper}}$ be the set of proper terms from $\mathcal{T}(\mathcal{F} \cup \mathcal{F}', \mathcal{V})$ and let $\mathcal{T}_{\mathit{func}} = \mathcal{T}(\mathcal{F}_{\mathit{func}}, \mathcal{V})$.

Lemma 6.4 (Properties of $\mathcal{A}$)**.** *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ be a proper DP problem and let $\mathcal{A}^{-1}$ be the inverse mapping to $\mathcal{A}$. For all t of $\mathcal{T}_{\mathit{proper}}$, all u, v of $\mathcal{T}_{\mathit{func}}$, all proper substitutions $\sigma : \mathcal{V} \to \mathcal{T}_{\mathit{proper}}$, and all substitutions $\delta : \mathcal{V} \to \mathcal{T}_{\mathit{func}}$, the following properties are valid.*

(i) $\mathcal{A}(t\sigma) = \mathcal{A}(t)\mathcal{A}(\sigma)$ *and* $\mathcal{A}^{-1}(u\delta) = \mathcal{A}^{-1}(u)\mathcal{A}^{-1}(\delta)$

(ii) If $t \to_{\mathcal{R}} s$ *then* $s \in \mathcal{T}_{\mathit{proper}}$

(iii) $t \in \mathit{NF}(\mathcal{Q})$ *iff* $\mathcal{A}(t) \in \mathit{NF}(\mathcal{A}(\mathcal{Q}))$

(iv) $t \overset{\mathcal{Q}}{\to}{}^{m}_{\mathcal{R}} s$ *implies* $\mathcal{A}(t) \overset{\mathcal{A}(\mathcal{Q})}{\to}{}^{m}_{\mathcal{A}(\mathcal{R})} \mathcal{A}(s)$, *and* $u \overset{\mathcal{A}(\mathcal{Q})}{\to}{}^{m}_{\mathcal{A}(\mathcal{R})} v$ *implies* $\mathcal{A}^{-1}(u) \overset{\mathcal{Q}}{\to}{}^{m}_{\mathcal{R}} \mathcal{A}^{-1}(v)$

Our aim is to prove soundness of the processor which replaces proper DP problems $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ by $(\mathcal{A}(\mathcal{P}), \mathcal{A}(\mathcal{Q}), \mathcal{A}(\mathcal{R}), f)$. To this end, one can show that every $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$-chain results in an $(\mathcal{A}(\mathcal{P}), \mathcal{A}(\mathcal{Q}), \mathcal{A}(\mathcal{R}), f)$-chain, i.e., that $t_i\sigma \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} s_{i+1}\sigma$ implies that there is a reduction $\mathcal{A}(t_i)\sigma' \overset{\mathcal{A}(\mathcal{Q})}{\to}{}^{*}_{\mathcal{A}(\mathcal{R})} \mathcal{A}(s_{i+1})\sigma'$ for some substitution σ'. The problem is that although all terms in $\mathcal{P}$, $\mathcal{Q}$, and $\mathcal{R}$ are proper, the substitution σ may introduce non-proper terms. For that reason we cannot directly use Lemma 6.4.

However, we now show that if $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$ then every $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain which uses a substitution σ can also be obtained by using a substitution with *proper* terms. To this end, we introduce another transformation $\mathcal{Y}$ from arbitrary to proper terms which simply replaces non-proper subterms t by a fresh variable $\bot_t$, and then we replace σ by $\mathcal{Y}(\sigma)$.

Unfortunately $t \overset{\mathcal{Q}}{\to}_{\mathcal{R}} s$ does not imply $\mathcal{Y}(t) \overset{\mathcal{Q}}{\to}_{\mathcal{R}} \mathcal{Y}(s)$ in general. The problem is that improper terms can be rewritten to proper ones. Consider $t = \mathsf{minus}'\mathsf{s}'\mathsf{0}'x$. While $t \overset{\mathcal{Q}}{\to}_{\mathcal{R}} \mathsf{s}'x$ using the rule (104), the term $\mathcal{Y}(t) = \bot_{\mathsf{minus}'\mathsf{s}'\mathsf{0}'x}$ is a normal form and cannot be reduced to $\mathcal{Y}(s) = \mathsf{s}'x$. Hence, if the substitution can contain $\mathcal{R}$-redexes, i.e., if $\mathit{NF}(\mathcal{Q}) \not\subseteq \mathit{NF}(\mathcal{R})$ then $\mathcal{Y}$ cannot be successfully used. However, in the other case $t \overset{\mathcal{Q}}{\to}_{\mathcal{R}} s$ indeed implies $\mathcal{Y}(t) \overset{\mathcal{Q}}{\to}_{\mathcal{R}} \mathcal{Y}(s)$, provided that t has the form $q\sigma$ for a proper term q and a $\mathcal{Q}$-normal substitution σ.

Later in Definition 6.10, we will also see another transformation $\mathcal{Z}$ which can be used for arbitrary sets $\mathcal{Q}$ but which has other drawbacks compared to $\mathcal{Y}$.

Definition 6.5 ($\mathcal{Y}$-Transformation). *$\mathcal{Y}$ is the transformation from $\mathcal{T}(\mathcal{F} \cup \mathcal{F}', \mathcal{V})$ to $\mathcal{T}(\mathcal{F} \cup \mathcal{F}', \mathcal{V} \uplus \mathcal{V}')$, where $\mathcal{V}' = \{\bot_t \mid t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}', \mathcal{V})\}$ are fresh variables.*

- $\mathcal{Y}(x) = x$, *for all* $x \in \mathcal{V}$
- $\mathcal{Y}(f^{?_1}t_1{}^{?_2}\dots{}^{?_n}t_n) = f^{?_1}\mathcal{Y}(t_1)^{?_2}\dots{}^{?_n}\mathcal{Y}(t_n)$, *for all* $f \in \mathcal{F}$ *with* $\mathit{a\text{-}ar}(f) = n$
- $\mathcal{Y}(t) = \bot_t$, *for all other* $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}', \mathcal{V})$

Moreover, for any substitution σ, $\mathcal{Y}(\sigma)$ is the substitution with $\mathcal{Y}(\sigma)(x) = \mathcal{Y}(\sigma(x))$.

Lemma 6.6 (Properties of $\mathcal{Y}$). *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ be a proper DP problem, let t and s be from $\mathcal{T}(\mathcal{F} \cup \mathcal{F}', \mathcal{V})$, let $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F} \cup \mathcal{F}', \mathcal{V})$, and let $\mu : \mathcal{V} \to \mathcal{T}(\mathcal{F} \cup \mathcal{F}', \mathcal{V} \uplus \mathcal{V}')$.*

(i) $\mathcal{Y}(t)$ is proper

(ii) If t is proper then $\mathcal{Y}(t\sigma) = t\mathcal{Y}(\sigma)$.

(iii) If t is in $\mathcal{Q}$-normal form then $\mathcal{Y}(t)$ is in $\mathcal{Q}$-normal form.

(iv) If σ is $\mathcal{Q}$-normal, t is proper, and $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$ then $t\sigma \overset{\mathcal{Q}}{\to}{}^{m}_{\mathcal{R}} s$ implies $\mathcal{Y}(t\sigma) \overset{\mathcal{Q}}{\to}{}^{m}_{\mathcal{R}} \mathcal{Y}(s)$.

(v) If σ is $\mathcal{Q}$-normal and $\mathcal{Y}(t\sigma)$ is in $\mathcal{Q}$-normal form then $t\sigma$ is in $\mathcal{Q}$-normal form.

(vi) If σ is $\mathcal{Q}$-normal and t is proper then termination of $t\sigma$ w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ implies termination of $\mathcal{Y}(t\sigma)$ w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$.[27]

---

[27] Note that $\mathcal{Y}$ does not preserve termination w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ for arbitrary terms t. Consider the TRS $\mathcal{R} = \{\mathsf{f}'(\mathsf{g}'x'y) \to \mathsf{f}'(\mathsf{g}'x'y), \mathsf{g}'x'x \to \mathsf{a}, \mathsf{b} \to \mathsf{a}, \mathsf{c} \to \mathsf{a}\}$ and $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$. Then $t = \mathsf{f}'(\mathsf{g}'(x'\mathsf{b})'(x'\mathsf{c}))$ is terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$: the only possible reduction is $t \overset{\mathcal{Q}}{\to}{}^{2}_{\mathcal{R}} \mathsf{f}'(\mathsf{g}'(x'\mathsf{a})'(x'\mathsf{a})) \overset{\mathcal{Q}}{\to}_{\mathcal{R}} \mathsf{f}'\mathsf{a}$ which is a normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$. On the other hand $\mathcal{Y}(t) = \mathsf{f}'(\mathsf{g}'\bot_{x'\mathsf{a}}{}'\bot_{x'\mathsf{b}}) \overset{\mathcal{Q}}{\to}_{\mathcal{R}} \mathcal{Y}(t) \overset{\mathcal{Q}}{\to}_{\mathcal{R}} \dots$.

$$\begin{array}{ccccccc}
t_1\sigma & \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} & s_2\sigma & , & t_2\sigma & \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} & s_3\sigma \;, \ldots \\
\mathcal{Y}(t_1\sigma) & \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} & \mathcal{Y}(s_2\sigma) & , & \mathcal{Y}(t_2\sigma) & \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} & \mathcal{Y}(s_3\sigma) \;, \ldots \\
t_1\mathcal{Y}(\sigma) & \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} & s_2\mathcal{Y}(\sigma) & , & t_2\mathcal{Y}(\sigma) & \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} & s_3\mathcal{Y}(\sigma) \;, \ldots \\
\mathcal{A}(t_1\mathcal{Y}(\sigma)) & \overset{\mathcal{A}(\mathcal{Q})}{\longrightarrow}{}^{*}_{\mathcal{A}(\mathcal{R})} & \mathcal{A}(s_2\mathcal{Y}(\sigma)) & , & \mathcal{A}(t_2\mathcal{Y}(\sigma)) & \overset{\mathcal{A}(\mathcal{Q})}{\longrightarrow}{}^{*}_{\mathcal{A}(\mathcal{R})} & \mathcal{A}(s_3\mathcal{Y}(\sigma)), \ldots \\
\mathcal{A}(t_1)\mathcal{A}(\mathcal{Y}(\sigma)) & \overset{\mathcal{A}(\mathcal{Q})}{\longrightarrow}{}^{*}_{\mathcal{A}(\mathcal{R})} & \mathcal{A}(s_2)\mathcal{A}(\mathcal{Y}(\sigma)), & & \ldots & & \ldots
\end{array}$$

Figure 6.7.: Transformation of chains

Combining Lemmas 6.4 and 6.6 allows us to transform each $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain into an $(\mathcal{A}(\mathcal{P}), \mathcal{A}(\mathcal{Q}), \mathcal{A}(\mathcal{R}))$-chain as depicted in Figure 6.7: The first row is the original chain where σ may contain improper terms. Then applying $\mathcal{Y}$ yields the second row, where still all reductions are possible due to Lemma 6.6 (iv). Afterwards we can extract the terms s_i and t_i by Lemma 6.6 (ii) which results in the third row. Hence, we have now obtained a chain where the substitution contains only proper terms due to Lemma 6.6 (i). Therefore, it is now possible to apply the $\mathcal{A}$-transformation where by Lemma 6.4 (iv) the reductions can by done by $\overset{\mathcal{A}(\mathcal{Q})}{\longrightarrow}{}^{*}_{\mathcal{A}(\mathcal{R})}$. By Lemma 6.4 (i) we finally get our desired chain in the fifth row: By using the remaining properties of Lemmas 6.4 and 6.6, one can show the two desired properties that minimality is preserved and that each term $\mathcal{A}(s_i)\mathcal{A}(\mathcal{Y}(\sigma))$ is in normal form w.r.t. $\mathcal{A}(\mathcal{Q})$.

Hence, we can formulate the desired processor which transforms proper applicative DP problems into functional form whenever $NF(\mathcal{Q}) \subseteq NF(\mathcal{R})$.

Theorem 6.8 ($\mathcal{A}$-Transformation Processor)**.** *The following processor Proc is sound and complete. For any DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, let Proc return*

- $\{(\mathcal{A}(\mathcal{P}), \mathcal{A}(\mathcal{Q}), \mathcal{A}(\mathcal{R}), f)\}$ *if $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ is proper and $NF(\mathcal{Q}) \subseteq NF(\mathcal{R})$*
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$ *otherwise.*

We demonstrate the use of the new theorem in the running example of this chapter.

Example 6.9. We continue in the termination proof of Example 6.1 with its remaining DP problem $\mathcal{D}_{13} = (\{(115)\}, \mathcal{Q}, \{(104), (105)\}, \mathbf{m})$.

$$\begin{array}{rcll}
\mathsf{div}'(\mathsf{s}'x)^{\sharp}(\mathsf{s}'y) & \to & \mathsf{div}'(\mathsf{minus}'x'y)^{\sharp}(\mathsf{s}'y) & (115) \\
\mathsf{minus}'x'\mathsf{0} & \to & x & (104) \\
\mathsf{minus}'(\mathsf{s}'x)'(\mathsf{s}'y) & \to & \mathsf{minus}'x'y & (105)
\end{array}$$

The result of the $\mathcal{A}$-transformation processor of Theorem 6.8 is the DP problem $\mathcal{D}_{14} = (\{(119)\}, \mathcal{A}(\mathcal{Q}), \{(120), (121)\}, \mathbf{m})$.

$$\begin{array}{rcll}
\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{DIV}(\mathsf{minus}(x, y), \mathsf{s}(y)) & (119) \\
\mathsf{minus}(x, \mathsf{0}) & \to & x & (120) \\
\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}(x, y) & (121)
\end{array}$$

Now, $\mathcal{D}_{14}$ can be solved using a reduction pair processor with the embedding order and a suitable argument filter.

The problem of Theorem 6.8 is its restricted applicability. We would also like to have a corresponding theorem without the restriction $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$. As argued before, the transformation $\mathcal{Y}$ cannot be used in the general case.

However, we now present a transformation $\mathcal{Z}$ such that instead of σ one can use the proper substitution $\mathcal{Z}(\sigma)$ to build a chain. Here, $\mathcal{Z}$ transforms arbitrary terms t and s into proper ones such that $t \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} s$ implies $\mathcal{Z}(t) \to^{*}_{\mathcal{R}} \mathcal{Z}(s)$. $\mathcal{Z}$ replaces terms where a variable is on the first argument of a binary symbol ${}^{?_i}$ or where a function symbol f has too few arguments by a fresh variable $\bot$. If f is applied to more arguments than its a-arity n, the first n arguments are modified by applying them to the arguments on positions $n+1, n+2, \ldots$. Afterwards, the arguments on the positions $n+1, n+2, \ldots$ are deleted.

As an example, regard the non-proper term $t = \mathsf{minus}\,'\,\mathsf{s}\,'\,\mathsf{0}\,'x$ where the symbol minus with a-arity 2 is applied to 3 arguments. $\mathcal{Z}$ removes the argument x and modifies the arguments s and $\mathsf{0}$ by applying them to x. So t is replaced by $\mathsf{minus}\,'(\mathsf{s}\,'x)\,'(\mathsf{0}\,'x)$. Now $\mathcal{Z}$ is called recursively on the subterms and therefore, the argument x of the symbol $\mathsf{0}$ with a-arity 0 is removed. Hence, $\mathcal{Z}(t) = \mathsf{minus}\,'(\mathsf{s}\,'x)\,'\mathsf{0}$. Note that for the original non-proper term t, we have $t \to_{\mathcal{R}} \mathsf{s}\,'x$ by the collapsing minus-rule (104). Similarly, we now also have $\mathcal{Z}(t) \to_{\mathcal{R}} \mathcal{Z}(\mathsf{s}\,'x) = \mathsf{s}\,'x$.

Definition 6.10 ($\mathcal{Z}$-Transformation). *$\mathcal{Z}$ is the following transformation from terms of $\mathcal{T}(\mathcal{F} \cup \mathcal{F}', \mathcal{V})$ to $\mathcal{T}(\mathcal{F} \cup \mathcal{F}', \mathcal{V} \uplus \{\bot\})$, where $\bot$ is a fresh variable. Here, $x \in \mathcal{V}$ and $f \in \mathcal{F}$ with a-ar$(f) = n$.*

- $\mathcal{Z}(x) = x$
- $\mathcal{Z}(f\,{}^{?_1}t_1\,{}^{?_2}\ldots{}^{?_k}t_k) =$
  $f\,{}^{?_1}\mathcal{Z}(t_1\,{}^{?_{n+1}}t_{n+1}\,{}^{?_{n+2}}\ldots{}^{?_k}t_k)\,{}^{?_2}\ldots{}^{?_n}\mathcal{Z}(t_n\,{}^{?_{n+1}}t_{n+1}\,{}^{?_{n+2}}\ldots{}^{?_k}t_k)$, *if* $k \geq n$
- $\mathcal{Z}(t) = \bot$, *for all other* $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}', \mathcal{V})$

Moreover, for any substitution σ, $\mathcal{Z}(\sigma)$ is the substitution with $\mathcal{Z}(\sigma)(x) = \mathcal{Z}(\sigma(x))$.

That $\mathcal{Y}$ cannot be used in the general case was already illustrated in the paragraph before Definition 6.5. However, we cannot use $\mathcal{Z}$ to get the results of Theorem 6.8 either, as $\mathcal{Z}$ preserves neither the strategy nor the termination behavior as $\mathcal{Y}$ does. Consider for example $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$ together with the TRS $\mathcal{R}$ with the rules $\mathsf{f}\,'(\mathsf{g}\,'x\,'y\,'z) \to z$ and $\mathsf{g}\,'x\,'x\,'y \to \mathsf{0}$. We obtain $t \overset{\mathcal{Q}}{\to}_{\mathcal{R}} \mathsf{0}$ for the non-proper term $t = \mathsf{f}\,'(\mathsf{g}\,'(\mathsf{0}\,'x)\,'(\mathsf{0}\,'y)\,'\mathsf{0})$, whereas $\mathcal{Z}(t) = \mathsf{f}\,'(\mathsf{g}\,'\mathsf{0}\,'\mathsf{0}\,'\mathsf{0})$ only reduces to $\mathsf{f}\,'\mathsf{0}$. So the problem is that $\mathcal{Z}$ can make different subterms equal by eliminating "superfluous" arguments. A similar example can be given to show that termination is not preserved. So we really need both transformations $\mathcal{Y}$ and $\mathcal{Z}$.

The following lemma states the desired properties of $\mathcal{Z}$ formally.

Lemma 6.11 (Properties of $\mathcal{Z}$). *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ be a proper DP problem, let t and s be from $\mathcal{T}(\mathcal{F} \cup \mathcal{F}', \mathcal{V})$, and let $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F} \cup \mathcal{F}', \mathcal{V})$.*

(i) $\mathcal{Z}(t)$ *is proper*

(ii) *If t is proper then $\mathcal{Z}(t\sigma\,{}^{?_{n+1}}t_{n+1}\,{}^{?_{n+2}}\ldots{}^{?_k}t_k) = t\mathcal{Z}(\overline{\sigma})$. Here, $\overline{\sigma}$ is substitution defined by $\overline{\sigma}(x) = \sigma(x)\,{}^{?_{n+1}}t_{n+1}\,{}^{?_{n+2}}\ldots{}^{?_k}t_k$.*

(iii) $t \to^{*}_{\mathcal{R}} s$ *implies* $\mathcal{Z}(t) \to^{*}_{\mathcal{R}} \mathcal{Z}(s)$

Now we can formulate a second processor to transform applicative DP problems using the $\mathcal{A}$-transformation where the idea is to transform chains in the same way as before. One just has to exchange $\mathcal{Y}$ by $\mathcal{Z}$ in Figure 6.7, and additionally one has to drop $\mathcal{Q}$ from the second row onwards.

Theorem 6.12 ($\mathcal{A}$-Transformation Processor)**.** *The following processor Proc is sound. For a DP problem* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, *Proc returns*

- $\{(\mathcal{A}(\mathcal{P}), \varnothing, \mathcal{A}(\mathcal{R}), \mathbf{a})\}$, *if* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ *is applicative and* $\mathcal{P} \cup \mathcal{R}$ *is proper*
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$, *otherwise.*

If $\mathcal{Q} = \varnothing$ *then Proc is even complete.*

With the new processors of Theorems 6.8 and 6.12 and our new improved estimation of dependency graphs (Definition 3.9 using Definition 3.11 as estimated *Cap*-function), it does not matter any more for an (innermost) termination proof whether first-order functions are represented in applicative or in ordinary functional form. The reason is that if they are represented by applicative rules, then all dependency pairs with non-proper right-hand sides are not in SCCs of the estimated dependency graph. Hence, after applying the dependency graph processors, all remaining DP problems are proper and can be transformed into functional form by Theorems 6.8 or 6.12.

Moreover, in this way one can also prove that $\mathcal{R}$ is (innermost) terminating iff $\mathcal{A}(\mathcal{R})$ is (innermost) terminating, provided that $\mathcal{R}$ is proper. For termination this was already shown in [KKSV96], but it is a new result for innermost termination.

Nevertheless, there are some problems with Theorem 6.12, since the processor preserves neither the strategy nor minimality. That this cannot be achieved without requiring $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$ as in Theorem 6.8, is illustrated in the following examples.

Example 6.13. We first show that the strategy cannot be preserved if we are not in the innermost case. Let $\mathcal{P}$ consist of the pair

$$\mathsf{f}\,'(\mathsf{g}'x)'(\mathsf{h}'y)^{\sharp}\,z \;\rightarrow\; \mathsf{f}\,'z'z^{\sharp}\,z \tag{122}$$

let $\mathcal{R}$ consist of the two rules

$$\mathsf{c}'x'y \;\rightarrow x \tag{123}$$

$$\mathsf{c}'x'y \;\rightarrow y \tag{124}$$

and let $\mathcal{Q} = \{\mathsf{c}'(\mathsf{g}'x)'y, \mathsf{c}'x'(\mathsf{g}'y)\}$. Then one can build the following infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain.

$$\begin{aligned}
& \mathsf{f}\,'(\mathsf{c}'\mathsf{g}'\mathsf{h}'x)'(\mathsf{c}'\mathsf{g}'\mathsf{h}'x)^{\sharp}\,(\mathsf{c}'\mathsf{g}'\mathsf{h}'x)\\
\xrightarrow{\mathcal{Q}}_{\mathcal{R}}\; & \mathsf{f}\,'(\mathsf{g}'x)'(\mathsf{c}'\mathsf{g}'\mathsf{h}'x)^{\sharp}\,(\mathsf{c}'\mathsf{g}'\mathsf{h}'x)\\
\xrightarrow{\mathcal{Q}}_{\mathcal{R}}\; & \mathsf{f}\,'(\mathsf{g}'x)'(\mathsf{h}'x)^{\sharp}\,(\mathsf{c}'\mathsf{g}'\mathsf{h}'x)\\
\xrightarrow{\mathcal{Q}}_{\mathcal{P},\varepsilon}\; & \mathsf{f}\,'(\mathsf{c}'\mathsf{g}'\mathsf{h}'x)'(\mathsf{c}'\mathsf{g}'\mathsf{h}'x)^{\sharp}\,(\mathsf{c}'\mathsf{g}'\mathsf{h}'x)\\
\xrightarrow{\mathcal{Q}}_{\mathcal{R}}\; & \cdots
\end{aligned}$$

Note that to build this chain we used the non-proper subterm $\mathsf{c}'\mathsf{g}'\mathsf{h}'x$ which can be reduced to the two proper terms $\mathsf{g}'x$ and $\mathsf{h}'x$.

However, for the $\mathcal{A}$-transformed problem with the following pairs and rules

$$\mathsf{F}(\mathsf{g}(x),\mathsf{h}(y),z) \to \mathsf{F}(z,z,z) \quad (125)$$

$$\mathsf{c}(x,y) \to x \quad (126)$$

$$\mathsf{c}(x,y) \to y \quad (127)$$

and the set $\mathcal{A}(\mathcal{Q}) = \{\mathsf{c}(\mathsf{g}(x),y), \mathsf{c}(x,\mathsf{g}(y))\}$ there is no infinite chain any more. This can be seen as follows: obviously the variable z of pair (125) must be instantiated by σ such that $z\sigma$ is reducible to both $\mathsf{g}(x)\sigma$ and $\mathsf{h}(x)\sigma$. But then $z\sigma$ must contain a subterm $\mathsf{c}(\mathsf{g}(t_1),t_2)$ or $\mathsf{c}(t_3,\mathsf{g}(t_4))$. In both cases $z\sigma$ is not in $\mathcal{A}(\mathcal{Q})$-normal form. Hence, the instantiated left-hand side of (125) is not in $\mathcal{A}(\mathcal{Q})$-normal form which proves that there is no infinite chain.

In Example 6.13 we have seen that the strategy cannot be preserved in the processor of Theorem 6.12. The next example demonstrates that this applies for minimality as well, even if $\mathcal{Q}$ is the empty set.

Example 6.14. We extend the TRS $\mathcal{R}$ of Example 6.13 by the following rules.

$$\mathsf{c}\,'(\mathsf{g}\,'x)\,'y \to \mathsf{c}\,'(\mathsf{g}\,'x)\,'y \quad (128)$$

$$\mathsf{c}\,'x\,'(\mathsf{g}\,'y) \to \mathsf{c}\,'x\,'(\mathsf{g}\,'y) \quad (129)$$

Now consider the DP problem $(\{(122)\}, \varnothing, \{(126),(127),(128),(129)\}, f)$. We obtain exactly the same infinite chain as in Example 6.13 which is again minimal. However, with the same argument as before any infinite chain in the $\mathcal{A}$-transformed DP problem must instantiate z by a term that contains $\mathsf{c}(\mathsf{g}(t_1),t_2)$ or $\mathsf{c}(t_3,\mathsf{g}(t_4))$ as a subterm. But then the resulting chain cannot be minimal due to the $\mathcal{A}$-transformed rules (128) and (129).

Notice that the two $\mathcal{A}$-transformation processors of Theorems 6.8 and 6.12 do not subsume each other, even in the case that $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$. The reason is that Theorem 6.12 does not require that $\mathcal{Q}$ is proper. Nevertheless, losing minimality is a severe problem as many of the powerful processors of Chapter 4 require minimality. However, the situation is quite similar to the one where one applies the needed rules processor of Theorem 4.12: using Theorem 6.12 directly destroys strategy and minimality. A better way would be to combine it with the reduction pair processor in a similar way to that of Theorem 4.18. Then one preserves minimality but only has to search for orders that orient the constraints arising of the $\mathcal{A}$-transformed rules and pairs. This approach and the adaptation of other processors of Sections 4.2 and 4.3 are investigated in the following section.

6.2. Needed Rules for Applicative DP Problems

The main problem of the processors of the previous section is that they do not work well if we are not in the innermost case. First, as the usable rules are often the whole TRS one cannot easily delete the improper rules and thus, the $\mathcal{A}$-transformation cannot be applied as all rules must be proper. And second, even if $\mathcal{A}$ is applicable then we lose minimality which is required for most of the powerful processors of Chapter 4.

To solve the first problem one can apply the needed rules processor of Theorem 4.12 to delete improper unneeded rules, but with the disadvantage that minimality is lost.[28] Alternatively, one can try to use the needed rules processor of Theorem 4.20, but this processor often is not applicable as one has to find reduction pairs over the applicative signature. And even after its successful application one still would lose minimality by the $\mathcal{A}$-transformation of Theorem 6.12.

To this end, we will adapt the transformation $\mathcal{I}$ to a similar transformation $\mathcal{I}'$ which combines $\mathcal{I}$ with $\mathcal{A}$. For this transformation it is only required that $\mathcal{P}$ and the needed rules are proper, and the unneeded rules may contain arbitrary improper terms. Then we can formulate new processors which can delete all strictly decreasing pairs, strictly decreasing rules, and unneeded rules, if the $\mathcal{A}$-transformed constraints can be solved. Moreover, we present a new processor which can apply the $\mathcal{A}$-transformation and loses neither minimality nor the strategy.

The main idea of the transformation $\mathcal{I}'$ is similar to the idea of $\mathcal{I}$. If we are not sure that only needed rules can be applied on a term t then we use a fresh binary symbol c to store all possible results of reducing t into a set which is encoded by c. Then using $\mathcal{C}_\varepsilon$ we can access each of this reducts of t. However, the main difference to $\mathcal{I}$ is that we are only interested in reductions leading to proper terms and that then we directly store the $\mathcal{A}$-transformed terms in the set.

In the details of $\mathcal{I}'$ we see two more differences to $\mathcal{I}$. First there is an additional forth case. This case is needed to handle improper terms. And second, one has to deal with possible reductions of a term $f^{?_1}t_1{}^{?_2}\dots{}^{?_n}t_n$ where f has a-arity n and where the redex is $f^{?_1}t_1{}^{?_2}\dots{}^{?_i}t_i$ for $i < n$.

Definition 6.15 ($\mathcal{I}'$). *Let $\mathcal{N} = \mathcal{N}(\mathcal{P},\mathcal{Q},\mathcal{R})$ be the set of needed rules of the given DP problem, and let $u_1 \to v_1, u_2 \to v_2, \dots$ be an infinite minimal $(\mathcal{P},\mathcal{Q},\mathcal{R})$-chain. Let ECap be the estimated Cap-function that is used to define the needed rules. Let $\mathcal{P} \cup \mathcal{N}$ be proper. W.l.o.g. we assume that σ is the substitution used for instantiating every $u_i \to v_i$. Moreover, whenever in the reduction of $v_i\sigma$ a rule $\ell_j \to r_j \in \mathcal{R}$ is applied, then by renaming the variables in the rule we again assume that the rule is instantiated by σ in that rewrite step. Let $\mathcal{S}_{all}$ contain all u_i and all direct subterms of each ℓ_j. Let c be the new constant and let $\bot$ be the new variable which are introduced by Comp. We define the mapping $\mathcal{I}'$ from terms that terminate w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ to functional terms of $\mathcal{T}(\mathcal{F}_{func} \uplus \{\mathsf{c}\}, \mathcal{V} \uplus \{\bot\})$ as follows.*

- *$\mathcal{I}'(x) = x$ for every variable x*
- *$\mathcal{I}'(f^{?_1}t_1{}^{?_2}\dots{}^{?_n}t_n) = f_{?_1\dots ?_n}(\mathcal{I}'(t_1),\dots,\mathcal{I}'(t_n))$, if a-ar$(f) = n$ and if there is no rule $\ell \to r \in \mathcal{R} \setminus \mathcal{N}$ and no $0 \le i \le n$ such that $Cap^{\mathcal{S}_{all}\sigma}_{\mathcal{R},\mathcal{Q}}(f^{?_1}t_1{}^{?_2}\dots{}^{?_{i-1}}t_{i-1})^{?_i}\,Cap^{\mathcal{S}_{all}\sigma}_{\mathcal{R},\mathcal{Q}}(t_i)$ unifies with ℓ by some mgu μ where $(\mathcal{S}_{all}\sigma \cup \{\ell_1,\dots,\ell_k\})\mu \subseteq NF(\mathcal{Q})$. Here, $\ell_1,\dots,\ell_k$ are the direct subterms of ℓ.*
- *$\mathcal{I}'(f^{?_1}t_1{}^{?_2}\dots{}^{?_n}t_n) = \mathsf{c}(f_{?_1\dots ?_n}(\mathcal{I}'(t_1),\dots,\mathcal{I}'(t_n)), Comp(Red'(t)))$, if the former case is not applicable, and if a-ar$(f) = n$.*
- *$\mathcal{I}'(t) = \mathsf{c}(\bot, Comp(Red'(t)))$, otherwise.*

---

[28] If we naively use the needed rules processor of Theorem 4.12 then afterwards the $\mathcal{A}$-transformation is still not applicable as the resulting DP problem is never proper. The reason is that $\mathcal{C}_\varepsilon$ consists of improper rules. However, one can easily adapt Theorem 4.12 where one replaces $\mathcal{C}_\varepsilon$ by $\mathcal{A}^{-1}(\mathcal{C}_\varepsilon)$. And if improper terms only occurred in the unneeded rules then one could indeed apply the $\mathcal{A}$-transformation afterwards. Nevertheless, minimality is lost which is not the case in the upcoming results.

As usual, $Red'(t) = \{\mathcal{I}'(s) \mid t \xrightarrow{\mathcal{Q}}_{\mathcal{R}} s\}$*, and* $\mathcal{I}'$ *is extended to substitutions.*

We obtain similar properties as in Lemma 4.11. The only main difference is that one has to demand that certain terms are proper and the resulting reductions are not w.r.t. $\to_{\mathcal{N}\cup\mathcal{C}_\varepsilon}$, but w.r.t. $\to_{\mathcal{A}(\mathcal{N})\cup\mathcal{C}_\varepsilon}$.

Lemma 6.16 (Properties of $\mathcal{I}'$)**.** *Let* $\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{N}, u_1 \to v_1, u_2, \to v_2, \ldots, \mathcal{S}_{all}, \sigma$ *be as in Definition 6.15. Let* t *be terminating w.r.t.* $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$.

(i) If $\mathcal{N}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{N}$, $\mathcal{S} \subseteq \mathcal{S}_{all}$*, and* t *is proper then* $\mathcal{I}'(t\sigma) = \mathcal{A}(t)\mathcal{I}'(\sigma)$.

(ii) If t *is proper then* $\mathcal{I}'(t\sigma) \to^*_{\mathcal{C}_\varepsilon} \mathcal{A}(t)\mathcal{I}'(\sigma)$.

(iii) If $t \xrightarrow{\mathcal{Q}}_{\mathcal{R}} s$*, and* $\mathcal{I}'(t)$ *is built by the third or forth case then* $\mathcal{I}'(t) \to^+_{\mathcal{C}_\varepsilon} \mathcal{I}'(s)$.

(iv) If $t \xrightarrow{\mathcal{Q}}_{\mathcal{R}} s$ *is a reduction at the root position and* $\mathcal{I}'(t)$ *is built by the second case then* $\mathcal{I}'(t) \to^*_{\mathcal{C}_\varepsilon}\to_{\mathcal{A}(\mathcal{N})} \mathcal{I}'(s)$.

(v) If $t \xrightarrow{\mathcal{Q}} s$ *then* $\mathcal{I}'(t) \to^+_{\mathcal{A}(\mathcal{N})\cup\mathcal{C}_\varepsilon} \mathcal{I}'(s)$.

(vi) $\mathcal{A}(u_1 \to v_1), \mathcal{A}(u_2 \to v_2), \ldots$ *is an* $(\mathcal{A}(\mathcal{P}), \varnothing, \mathcal{A}(\mathcal{N}) \cup \mathcal{C}_\varepsilon)$*-chain.*

Now we can formulate the processor based on $\mathcal{I}'$. It encapsulates many processors of Chapter 4 based on needed rules in one definition, and it combines them all with the $\mathcal{A}$-transformation. In all cases we only have to solve constraints over the functional signature and it is only required that the needed rules are proper. We will discuss the relations between these processors and the corresponding processors of Chapter 4 directly after the theorem.

Theorem 6.17 (Processors Based on Needed Rules for Applicative DP Problems)**.** *Let* $(\succsim, \succ)$ *be a reduction pair where* $\succsim$ *is* $\mathcal{C}_\varepsilon$*-compatible. Let ECap be the estimated Cap-function for the needed rules, let* $\mathcal{N} = \mathcal{N}(\mathcal{P}, \mathcal{Q}, \mathcal{R})$*. Then the following processor Proc is sound. For a DP problem* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$*, Proc can return* $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$ *or one of the following sets of DP problems.*

(A) $\{(\mathcal{A}(\mathcal{P}), \varnothing, \mathcal{A}(\mathcal{N}) \cup \mathcal{C}_\varepsilon, \mathbf{a})\}$,
if $\mathcal{P} \cup \mathcal{N}$ *is proper and if* $f = \mathbf{m}$

(B) $\{(\mathcal{P}, \mathcal{Q}, \mathcal{N}', f)\}$,
if $\mathcal{P} \cup \mathcal{N}$ *is proper,* $f = \mathbf{m}$, $\mathcal{A}(\mathcal{P} \cup \mathcal{N}) \subseteq \succsim$, $\mathcal{N}' = \mathcal{N} \setminus \{\ell \to r \in \mathcal{N} \mid \mathcal{A}(\ell) \succ \mathcal{A}(r)\}$,
and if $\succ$ *is monotonic and* $\mathcal{C}_\varepsilon$*-compatible*

(C) $\{(\mathcal{A}(\mathcal{P}), \mathcal{A}(\mathcal{Q}), \mathcal{A}(\mathcal{N}'), f)\}$,
if $\mathcal{P}\cup\mathcal{N}\cup\mathcal{Q}$ *is proper,* $f = \mathbf{m}$, $\mathcal{A}(\mathcal{P}\cup\mathcal{N}) \subseteq \succsim$, $\mathcal{N}' = \mathcal{N}\setminus\{\ell \to r \in \mathcal{N} \mid \mathcal{A}(\ell) \succ \mathcal{A}(r)\}$,
and if $\succ$ *is monotonic and* $\mathcal{C}_\varepsilon$*-compatible*

(D) $\{(\mathcal{P} \setminus \{s \to t \in \mathcal{P} \mid \mathcal{A}(s) \succ \mathcal{A}(t)\}, \mathcal{Q}, \mathcal{R}, f)\}$,
if $\mathcal{P} \cup \mathcal{N}$ *is proper,* $f = \mathbf{m}$, $\mathcal{A}(\mathcal{P} \cup \mathcal{N}) \subseteq \succsim$*, and if* $\mathcal{A}(\mathcal{P}) \cap \succ \neq \varnothing$

The processor is complete in all cases but (A).

Alternative (A) clearly corresponds to the combination of applying first the needed rules processor of Theorem 4.12 followed by an application of the $\mathcal{A}$-transformation processor of Theorem 6.12, cf. Footnote 28. This processor loses minimality and is incomplete, but on the other hand it has very few requirements: it is not required to find a suitable reduction pair directly. Hence, this processor should be used if one wants to apply other processors that benefit from different root symbols afterwards. Examples include processors that are based on semantic labeling, cf. Chapter 7.

However, if one can find a reduction pair satisfying the constraints then one should not use alternative (A). If one uses monotonic reduction pairs that are $\mathcal{C}_\varepsilon$-compatible then with alternative (B) we get a similar processor to the needed rules processor of Theorem 4.20. However, here we also integrated the effect of the rule removal processor of Theorem 4.22. The reason is that in Chapter 4 both theorems for the needed rules processor and the rule removal processor can be applied in sequence without problems. This is not the case in the applicative case. Even after one has removed all unneeded rules ($\mathcal{R} \setminus \mathcal{N}$), one still remains with an applicative problem. Hence, one again wants to look at the $\mathcal{A}$-transformed constraints to remove rules, and this is what is done in alternative (B).

Note that in alternatives (A) and (B) there is no requirement that $\mathcal{Q}$ is proper. This is different in alternative (C). If additionally $\mathcal{Q}$ is proper then one can return the $\mathcal{A}$-transformed DP problem of the result of alternative (B).[29] The requirement that $\mathcal{Q}$ must be proper is essential, because otherwise it would be impossible to apply $\mathcal{A}$ on $\mathcal{Q}$.

Finally, alternatives (B) and (C) require that the strict order is monotonic which forbids the use of argument filters. This is not the case in alternative (D) which directly corresponds to the reduction pair processor of Theorem 4.18.

We illustrate the use of Theorem 6.17 in the following variant of our running example.

Example 6.18. Let $\mathcal{R}$ be the TRS of Example 6.1 where we added rules to get a random number and where we modified the recursive minus-rule.

$$\begin{aligned}
\mathsf{map}\,{}'\alpha{}'\mathsf{nil} &\to \mathsf{nil} && (102)\\
\mathsf{map}\,{}'\alpha{}'(\mathsf{cons}\,{}'x{}'xs) &\to \mathsf{cons}\,{}'(\alpha{}'x){}'(\mathsf{map}\,{}'\alpha{}'xs) && (103)\\
\mathsf{random} &\to \mathsf{0} && (130)\\
\mathsf{random} &\to \mathsf{s}{}'\mathsf{0} && (131)\\
\mathsf{p}{}'(\mathsf{s}{}'x) &\to x && (132)\\
\mathsf{minus}{}'x{}'\mathsf{0} &\to x && (104)\\
\mathsf{minus}{}'x{}'(\mathsf{s}{}'y) &\to \mathsf{p}{}'(\mathsf{minus}{}'x{}'(\mathsf{p}{}'(\mathsf{s}{}'y))) && (133)\\
\mathsf{div}{}'\mathsf{0}{}'(\mathsf{s}{}'y) &\to \mathsf{0} && (106)\\
\mathsf{div}{}'(\mathsf{s}{}'x){}'(\mathsf{s}{}'y) &\to \mathsf{s}{}'(\mathsf{div}{}'(\mathsf{minus}{}'x{}'y){}'(\mathsf{s}{}'y)) && (107)
\end{aligned}$$

Note that this TRS does not belong to a class where innermost termination implies termination due to the random-rules. Hence, using the processors of Chapters 3 and 4 in combination with standard orders, we remain with the two DP problems $(\{(134)\}, \varnothing, \mathcal{R}, \mathbf{m})$

---

[29] For alternative (C) it is not really required to integrate the effect of the rule removal processor as this processor would be applicable afterwards. However, then (B) would have less requirements than (C) but it would be able to delete more rules which looks counterintuitive. Therefore, also in (C) it is allowed to delete rules which are strictly decreasing.

and $(\{(115)\}, \varnothing, \mathcal{R}, \mathbf{m})$ where both DP problems still contain the whole TRS $\mathcal{R}$.

$$\begin{array}{rcll}
\mathsf{minus}'x\,{}^{\sharp}(\mathsf{s}'y) & \rightarrow & \mathsf{minus}'x\,{}^{\sharp}(\mathsf{p}'(\mathsf{s}'y)) & (134)\\
\mathsf{div}'(\mathsf{s}'x)\,{}^{\sharp}(\mathsf{s}'y) & \rightarrow & \mathsf{div}'(\mathsf{minus}'x'y)\,{}^{\sharp}(\mathsf{s}'y) & (115)
\end{array}$$

Since $\mathcal{R}$ is not proper we cannot apply the $\mathcal{A}$-transformation with the processors of the previous section. However, the only needed rule of the further DP problem is (132). Hence, by using alternative (C) of Theorem 6.17 one can simplify this DP problem to $(\{\mathsf{MINUS}(x, \mathsf{s}(y)) \rightarrow \mathsf{MINUS}(x, \mathsf{p}(\mathsf{s}(y)))\}, \varnothing, \varnothing, \mathbf{m})$ by a polynomial order which maps $\mathsf{s}(x)$ to $x + 1$ and every other symbol to the sum of its arguments. This problem is then easy to solve by the processors based on the dependency graph.

For the other remaining DP problem we can neither apply alternative (C) nor alternative (B) if we use a quasi-simplification order due to the monotonicity requirements. But the other two alternatives succeed. The set of needed rules is $\{(132), (104), (133)\}$ and for example, we can choose the embedding order and the argument filter which only replaces minus by its first argument to solve the constraints of alternative (D).

6.3. Argument Filters for Applicative DP Problems

What is missing in the previous section is a processor that corresponds to the reduction pair processor based on argument filters and needed rules of Theorem 4.32 where one only has to build constraints for the needed rules w.r.t. a given argument filter. It may be the case that the needed rules still contain improper terms, but if one regards the argument filter then all these improper rules are not needed any more, or the improper parts of the rules are dropped by the filter. This is illustrated further in the upcoming example.

Of course, one wants to obtain needed rules w.r.t. an argument filter over the functional signature with n-ary symbols and not with an argument filter over the applicative signature where we can specify a filter only for the binary application symbols.

Example 6.19. We consider the following TRS implementing the sieve of Eratosthenes.

$$\begin{array}{rcll}
\mathsf{divides}'(\mathsf{s}'x)'\mathsf{0} & \rightarrow & \mathsf{true} & (135)\\
\mathsf{divides}'(\mathsf{s}'x)'(\mathsf{s}'y) & \rightarrow & \mathsf{div}'(\mathsf{s}, x)'y'x & (136)\\
\mathsf{div}'x'y'\mathsf{0} & \rightarrow & \mathsf{divides}'x'y & (137)\\
\mathsf{div}'x'\mathsf{0}'(\mathsf{s}'z) & \rightarrow & \mathsf{false} & (138)\\
\mathsf{div}'x'(\mathsf{s}'y)'(\mathsf{s}'z) & \rightarrow & \mathsf{div}'x'y'z & (139)\\
\mathsf{generate}'x'\mathsf{0} & \rightarrow & \mathsf{nil} & (140)\\
\mathsf{generate}'x'(\mathsf{s}'y) & \rightarrow & \mathsf{cons}'x'(\mathsf{generate}'(\mathsf{s}'x)'y) & (141)\\
\mathsf{filter}'\alpha'\mathsf{nil} & \rightarrow & \mathsf{nil} & (142)\\
\mathsf{filter}'\alpha'(\mathsf{cons}'x'xs) & \rightarrow & \mathsf{if}'(\alpha'x)'x'(\mathsf{filter}'\alpha'xs) & (143)\\
\mathsf{if}'\mathsf{true}'x'xs & \rightarrow & xs & (144)\\
\mathsf{if}'\mathsf{false}'x'xs & \rightarrow & \mathsf{cons}'x'xs & (145)\\
\mathsf{if}'\mathsf{true}'x'xs & \rightarrow & xs & (146)\\
\mathsf{sieve}'\mathsf{nil} & \rightarrow & \mathsf{nil} & (147)\\
\mathsf{sieve}'(\mathsf{cons}'x'xs) & \rightarrow & \mathsf{cons}'x'(\mathsf{sieve}'(\mathsf{filter}'(\mathsf{divides}'x)'xs)) & (148)
\end{array}$$

Here, $\mathsf{divides}$ computes whether the second argument is divisible by its first argument. The term $\mathsf{generate}\,'x\,'y$ evaluates to the list $[x, x+1, \ldots, x+y-1]$. The higher-order function filter takes a function α and a list as input and drops all elements x from the list where $\alpha(x)$ is true. Finally, $\mathsf{sieve}\,'[2, 3, \ldots, x]$ computes the list of prime numbers between 2 and x by filtering recursively all numbers that are divisible by a detected prime number. Hence, the term $\mathsf{sieve}\,'(\mathsf{generate}\,'2\,'(x-1))$ computes all primes up to x.

Here, we see a new challenge for proving termination. In the recursive call of the function sieve we have a call to the higher-order function filter and moreover, we see a partial application of the $\mathsf{divides}$-function. Hence, the resulting DP problem will not be proper. However, if we partially apply the $\mathcal{A}$-transformation and directly use an argument filter π over the functional signature which drops the first argument of filter and if, then the higher-order parts are deleted and we get no problems in the $\mathcal{A}$-transformation. This idea leads to the new notion of *π-proper* which essentially is the requirement that in this combined process of argument filtering and $\mathcal{A}$-transforming – which is called *$\mathcal{A}_\pi$-transformation* – one will never encounter subterms that cannot be $\mathcal{A}_\pi$-transformed.

Moreover, if we already know that certain parts are deleted then we can also exploit this fact to get less *needed rules w.r.t. π*. For example, then the $\mathsf{divides}$-rules are not needed any more. These considerations result in Definition 6.21 about needed rules w.r.t. π and in the corresponding processor in Theorem 6.22, which is strictly more powerful than the processor of Theorem 6.17 (D).

Definition 6.20 (π-Proper and $\mathcal{A}_\pi$)**.** *Let $\mathcal{F} \cup \mathcal{F}'$ be an applicative signature, let $a\text{-}ar : \mathcal{F} \to \mathbb{N}$ determine the arities of the applicative signature, let π be an argument filter over the functional signature $\mathcal{F}_{func}$. Then a term $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}', \mathcal{V})$ is* π-proper *iff*

- *t is a variable or*
- *$t = f\,{}^{?_1}t_1\,{}^{?_2}\ldots{}^{?_n}t_n$, $a\text{-}ar(f) = n$, and for every $i \in RegPos_\pi(f_{?_1\ldots?_n})$ the term t_i is π-proper.*

*We define the $\mathcal{A}_\pi$-*transformation *from π-proper terms to filtered functional terms as*

- *$\mathcal{A}_\pi(x) = x$ for every variable x,*
- *$\mathcal{A}_\pi(f\,{}^{?_1}t_1\,{}^{?_2}\ldots{}^{?_n}t_n) = \mathcal{A}_\pi(t_i)$, if $\pi(f_{?_1\ldots?_n}) = i$, and*
- *$\mathcal{A}_\pi(f\,{}^{?_1}t_1\,{}^{?_2}\ldots{}^{?_n}t_n) = f_{?_1\ldots?_n}(\mathcal{A}_\pi(t_{i_1}), \ldots, \mathcal{A}_\pi(t_{i_k}))$, if $\pi(f_{?_1\ldots?_n}) = [i_1, \ldots, i_k]$.*

We extend π-proper to rules, substitutions, and sets of rules where we require that all occurring terms are π-proper. We also extend $\mathcal{A}_\pi$ to π-proper rules, substitutions, and sets of rules where we apply $\mathcal{A}_\pi$ component-wise.

To define a version of needed rules w.r.t. an argument filter for applicative DP problems we essentially first apply the $\mathcal{A}$-transformation and then look at the needed rules w.r.t. an argument filter as in Definition 4.31. However, we also have to take into account that a term $f\,{}^{?_1}t_1\,{}^{?_2}\ldots{}^{?_n}t_n$ with $a\text{-}ar(f) = n$ can have a redex $f\,{}^{?_1}t_1\,{}^{?_2}\ldots{}^{?_i}t_i$ for every $i \leq n$. (Note that this problem was already addressed in the definition of $\mathcal{I}'$.) All this is captured formally in the upcoming definition.

Definition 6.21 (Applicative Needed Rules w.r.t. an Argument Filter)**.** *Let $\mathcal{Q}$ and $\mathcal{S}$ be sets of terms, let $\mathcal{P}$ and $\mathcal{R}$ be TRSs, let t be a term, where all terms in $\mathcal{Q} \cup \mathcal{S} \cup \mathcal{R} \cup \{t\}$ are over the applicative signature $\mathcal{F} \cup \mathcal{F}'$. Let a-ar : $\mathcal{F} \to \mathbb{N}$ determine the arities of the applicative signature, let π be an argument filter over the functional signature $\mathcal{F}_{func}$, and let ECap be an estimated Cap-function. The* applicative needed rules *of a term t w.r.t. π are defined as the smallest set ${\mathcal{N}'}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{R}$ such that*

(i) If $t = f^{?_1}t_1{}^{?_2} \dots {}^{?_n}t_n$, a-ar$(f) = n$, $\ell \to r \in \mathcal{R}$, and if there is an i with $0 \leq i \leq n$ such that $ECap^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(f^{?_1}t_1{}^{?_2} \dots {}^{?_{i-1}}t_{i-1})^{?_i} ECap^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_i)$ unifies with ℓ by some mgu μ where $(\mathcal{S} \cup \{\ell_1, \dots, \ell_k\})\mu \subseteq NF(\mathcal{Q})$ then $\ell \to r \in {\mathcal{N}'}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$. Here, $\ell_1, \dots, \ell_k$ are the direct subterms of ℓ.

(ii) If $t = f^{?_1}t_1{}^{?_2} \dots {}^{?_n}t_n$ and $i \in RegPos_\pi(f_{?_1 \dots ?_n})$ then ${\mathcal{N}'}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t_i) \subseteq {\mathcal{N}'}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$.

(iii) If $\ell \to r \in {\mathcal{N}'}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$ then ${\mathcal{N}'}^{\{\ell_1,\dots,\ell_k\},\pi}_{\mathcal{R},\mathcal{Q}}(r) \subseteq {\mathcal{N}'}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$. Again, $\ell_1, \dots, \ell_k$ are the direct subterms of ℓ.

As usual, $\mathcal{N}'(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \pi) = \bigcup_{s \to t \in \mathcal{P}} {\mathcal{N}'}^{\{s\},\pi}_{\mathcal{R},\mathcal{Q}}(t)$.

Now we can present the main result of this section, an improved version of the processor (D) in Theorem 6.17. Instead of a proper DP problem here we only require a π-proper DP problem and we get less needed rules and hence, less constraints which have to be satisfied.

Theorem 6.22 (Processors Based on Reduction Pairs and Needed Rules w.r.t. an Argument Filter for Applicative DP Problems)**.** *Let $(\succsim, \succ)$ be a reduction pair where $\succsim$ is $\mathcal{C}_\varepsilon$-compatible. Let a-ar determine the arities and let π be an argument filter over the functional signature. Let ECap be an estimated Cap-function that parameterizes the needed rules $\mathcal{N}'$ in Definition 6.21. Then the following processor Proc is sound and complete. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, Proc returns*

- *$\{(\mathcal{P} \setminus \{s \to t \in \mathcal{P} \mid \mathcal{A}_\pi(s) \succ \mathcal{A}_\pi(t)\}, \mathcal{Q}, \mathcal{R}, f)\}$,
  if $f = \mathbf{m}$, $\mathcal{N} = \mathcal{N}'(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \pi)$, $\mathcal{P} \cup \mathcal{N}$ is π-proper, and $\mathcal{A}_\pi(\mathcal{P} \cup \mathcal{N}) \subseteq \succsim$.*
- *$\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$, otherwise*

To prove Theorem 6.22 we essentially have to integrate the argument filter π into $\mathcal{I}'$ which results in the following definition.

Definition 6.23 ($\mathcal{I}'_\pi$)**.** *Let a-ar determines the arities, let $u_1 \to v_1, u_2 \to v_2, \dots$ be an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain, let π be an argument filter over the functional signature, and let $\mathcal{N} = \mathcal{N}'(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \pi)$ be set of needed rules of the given DP problem. Let ECap be the estimated Cap-function that is used to define the needed rules and let $\mathcal{P} \cup \mathcal{N}$ be π-proper. W.l.o.g. we assume that σ is the substitution used for instantiating every $u_i \to v_i$. Moreover, whenever in the reduction of $v_i\sigma$ a rule $\ell_j \to r_j \in \mathcal{R}$ is applied, then by renaming the variables in the rule we again assume that the rule is instantiated by σ in that rewrite step. Let $\mathcal{S}_{all}$ contain all u_i and all direct subterms of each ℓ_j. Let c be the new constant and let $\bot$ be the new variable which are introduced by Comp. We define the mapping $\mathcal{I}'_\pi$ from terms that terminate w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ to filtered functional terms of $\mathcal{T}_{func} = \mathcal{T}(\mathcal{F}_{func} \uplus \{\mathsf{c}\}, \mathcal{V} \uplus \{\bot\})$ as follows.*

- $\mathcal{I}'_\pi(x) = x$ *for every variable* x
- $\mathcal{I}'_\pi(f\,{}^{?_1}t_1{}^{?_2}\dots{}^{?_n}t_n) = \overline{f_{?_1\dots?_n}(\mathcal{I}'_\pi(t_1),\dots,\mathcal{I}'_\pi(t_n))}$, *if* $a\text{-}ar(f) = n$, *and if there is no rule* $\ell \to r \in \mathcal{R} \setminus \mathcal{N}$ *and no* $0 \le i \le n$ *such that* $Cap^{\mathcal{S}_{all}\sigma}_{\mathcal{R},\mathcal{Q}}(f\,{}^{?_1}t_1{}^{?_2}\dots{}^{?_{i-1}}t_{i-1})^{?_i}$ $Cap^{\mathcal{S}_{all}\sigma}_{\mathcal{R},\mathcal{Q}}(t_i)$ *unifies with* ℓ *by some mgu* μ *where* $(\mathcal{S}_{all}\sigma \cup \{\ell_1,\dots,\ell_k\})\mu \subseteq NF(\mathcal{Q})$. *Here,* $\ell_1,\dots,\ell_k$ *are the direct subterms of* ℓ.
- $\mathcal{I}'_\pi(f\,{}^{?_1}t_1{}^{?_2}\dots{}^{?_n}t_n) = \mathsf{c}(\overline{f_{?_1\dots?_n}(\mathcal{I}'_\pi(t_1),\dots,\mathcal{I}'_\pi(t_n))}, Comp(Red'_\pi(t)))$, *if* $a\text{-}ar(f) = n$ *and if the former case is not applicable.*
- $\mathcal{I}'_\pi(t) = Comp(Red'_\pi(t))$, *otherwise.*

Here, $\overline{t}$ *is the term where* π *is applied on* t *only at the root level, i.e., if* $\pi(f_{?_1\dots?_n}) = i$ *then* $\overline{f_{?_1\dots?_n}(s_1,\dots,s_n)} = s_i$, *and if* $\pi(f_{?_1\dots?_n}) = [i_1,\dots,i_k]$ *then* $\overline{f_{?_1\dots?_n}(s_1,\dots,s_n)} = f_{?_1\dots?_n}(s_{i_1},\dots,s_{i_k})$. *As usual,* $Red'_\pi(t) = \{\mathcal{I}'_\pi(s) \mid t \xrightarrow{\mathcal{Q}}_{\mathcal{R}} s\}$ *and* $\mathcal{I}'_\pi$ *is extended to substitutions.*

We obtain similar properties as in Lemma 6.16. Essentially we only have to replace proper by π-proper, $\mathcal{I}'$ by $\mathcal{I}'_\pi$, $\mathcal{N}$ by $\mathcal{N}'$, $\mathcal{A}$ by $\mathcal{A}_\pi$, and in (v) we have to replace $\to^+$ by $\to^*$. The only new case is in the proof of (v) where we have to consider reductions on positions that are dropped by the argument filter π.

Lemma 6.24 (Properties of $\mathcal{I}'_\pi$). *Let* $\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{N}, u_1 \to v_1, u_2, \to v_2, \dots, \mathcal{S}_{all}, \sigma$ *be as in Definition 6.23. Let* t *be terminating w.r.t.* $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$.

(i) If ${\mathcal{N}'}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{N}$, $\mathcal{S} \subseteq \mathcal{S}_{all}$, *and* t *is* π*-proper then* $\mathcal{I}'_\pi(t\sigma) = \mathcal{A}_\pi(t)\mathcal{I}'_\pi(\sigma)$.

(ii) If t *is* π*-proper then* $\mathcal{I}'_\pi(t\sigma) \to^*_{\mathcal{C}_\varepsilon} \mathcal{A}_\pi(t)\mathcal{I}'_\pi(\sigma)$.

(iii) If $t \xrightarrow{\mathcal{Q}}_{\mathcal{R}} s$, *and* $\mathcal{I}'_\pi(t)$ *is built by the third or forth case then* $\mathcal{I}'_\pi(t) \to^+_{\mathcal{C}_\varepsilon} \mathcal{I}'_\pi(s)$.

(iv) If $t \xrightarrow{\mathcal{Q}}_{\mathcal{R}} s$ *is a reduction at the root position and* $\mathcal{I}'_\pi(t)$ *is built by the second case then* $\mathcal{I}'_\pi(t) \to^*_{\mathcal{C}_\varepsilon} \to_{\mathcal{A}_\pi(\mathcal{N})} \mathcal{I}'_\pi(s)$.

(v) If $t \xrightarrow{\mathcal{Q}} s$ *then* $\mathcal{I}'_\pi(t) \to^*_{\mathcal{A}_\pi(\mathcal{N}) \cup \mathcal{C}_\varepsilon} \mathcal{I}'_\pi(s)$.

(vi) $\mathcal{A}_\pi(u_1 \to v_1), \mathcal{A}_\pi(u_2 \to v_2), \dots$ *is a* $(\mathcal{A}_\pi(\mathcal{P}), \varnothing, \mathcal{A}_\pi(\mathcal{N}) \cup \mathcal{C}_\varepsilon)$*-chain.*

At the end of this chapter we finally show how to prove termination of the TRS of Example 6.19.

Example 6.25. Using previous techniques we can transform the initial DP problem of Example 6.19 into the DP problem $(\{(149) - (152)\}, \varnothing, \mathcal{R}, \mathbf{m})$ with the following pairs.[30]

$$\begin{aligned}
\mathsf{sieve}^\sharp(\mathsf{cons}'x'xs) &\to \mathsf{sieve}^\sharp(\mathsf{filter}'(\mathsf{divides}'x)'xs) && (149)\\
\mathsf{sieve}^\sharp(\mathsf{cons}'x'xs) &\to \mathsf{filter}'(\mathsf{divides}'x)^\sharp xs && (150)\\
\mathsf{filter}'\alpha^\sharp(\mathsf{cons}'x'xs) &\to \alpha^\sharp x && (151)\\
\mathsf{filter}'\alpha^\sharp(\mathsf{cons}'x'xs) &\to \mathsf{filter}'\alpha^\sharp xs && (152)
\end{aligned}$$

Note that we still cannot apply Theorem 6.22 as (151) is never π-proper regardless of π. To this end we first apply the argument filter processor of Theorem 4.38 which replaces each $\sharp$-term by its second argument. In this way one obtains the new DP problem $(\{(153)–(156)\}, \varnothing, \mathcal{R}, \mathbf{m})$ with the following pairs.

---

[30] It is also possible to switch to the innermost case with $\mathcal{Q} = lhs(\mathcal{R})$ but this does not help in the termination proof.

$$\begin{aligned}
\mathsf{cons}\,'x\,'xs &\rightarrow \mathsf{filter}\,'(\mathsf{divides}\,'x)\,'xs && (153)\\
\mathsf{cons}\,'x\,'xs &\rightarrow xs && (154)\\
\mathsf{cons}\,'x\,'xs &\rightarrow x && (155)\\
\mathsf{cons}\,'x\,'xs &\rightarrow xs && (156)
\end{aligned}$$

Here, the improper pair (151) is transformed into the proper pair (155). Note, that even for this reduced DP problem all rules are needed. Thus, the needed rules are not proper as rule (143) contains the application of α on x. Hence, the processors of Theorem 6.17 are still not applicable.

In contrast we can now apply Theorem 6.22 using the applicative arities $a\text{-}ar(\mathsf{if}) = 3$, $a\text{-}ar(\mathsf{filter}) = a\text{-}ar(\mathsf{cons}) = 2$, $a\text{-}ar(\mathsf{divides}) = 1$, and $a\text{-}ar(\mathsf{nil}) = 0$ together with the argument filter π which only removes the first argument of $\mathsf{if}_{'''}$. Then only rules (142) – (145) are needed w.r.t. π. Note that there is no conflict with the applicative arity of $\mathsf{divides}$ since the rules (135) – (137) are not needed w.r.t. π. We obtain the following constraints by applying $\mathcal{A}_\pi$.

$$\begin{aligned}
\mathsf{cons}_{''}(x, xs) &\underset{(\sim)}{\succ} \mathsf{filter}_{''}(\mathsf{divides}_{'}(x), xs)\\
\mathsf{cons}_{''}(x, xs) &\underset{(\sim)}{\succ} xs\\
\mathsf{cons}_{''}(x, xs) &\underset{(\sim)}{\succ} x\\
\mathsf{cons}_{''}(x, xs) &\underset{(\sim)}{\succ} xs\\
\mathsf{filter}_{''}(\alpha, \mathsf{nil}) &\succsim \mathsf{nil}\\
\mathsf{filter}_{''}(\alpha, \mathsf{cons}_{''}(x, xs)) &\succsim \mathsf{if}_{'''}(x, \mathsf{filter}_{''}(\alpha, xs))\\
\mathsf{if}_{'''}(x, xs) &\succsim \mathsf{cons}_{''}(x, xs)\\
\mathsf{if}_{'''}(x, xs) &\succsim xs
\end{aligned}$$

These constraints are satisfied by the following polynomial order:

$$\begin{aligned}
\mathcal{P}ol(\mathsf{nil}) &= 0\\
\mathcal{P}ol(\mathsf{divides}_{'}(x)) &= 0\\
\mathcal{P}ol(\mathsf{cons}_{''}(x, y)) &= 1 + x + y\\
\mathcal{P}ol(\mathsf{filter}_{''}(x, y)) &= y\\
\mathcal{P}ol(\mathsf{if}_{'''}(x, y)) &= 1 + x + y
\end{aligned}$$

For this order, the first four constraints are strictly decreasing and can be removed by Theorem 6.22. As no pairs remain we have proven termination of $\mathcal{R}$.

Note that it is also possible to combine the argument filter processor of Theorem 4.38 with Theorem 6.22 to overcome the incompleteness of the argument filter processor. Given a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ one first applies the argument filter processor with some filter π. On the resulting DP problem $(\pi(\mathcal{P}), \mathcal{Q}, \mathcal{R}, \mathbf{m})$ one applies the processor of Theorem 6.22 to obtain the DP problem $(\mathcal{P}', \mathcal{Q}, \mathcal{R}, \mathbf{m})$ with $\mathcal{P}' \subseteq \pi(\mathcal{P})$. Then one can return the DP problem $(\{s \rightarrow t \in \mathcal{P} \mid \pi(s \rightarrow t) \in \mathcal{P}'\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ as the result of the combined approach. Since this resulting DP problem is a subproblem of $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$, the combined approach is complete due to Lemma 2.17.

Summary of Chapter 6

In this chapter we have investigated various ways to handle applicative TRSs and DP problems. While the estimations of the dependency graph and the usable rules work quite well with our improved estimations of Chapter 3, there still were problems when applying the processors based on well-founded orders of Chapter 4. To this end, we use the $\mathcal{A}$-transformation to transform applicative DP problems to functional form, if they are proper.

Since in the termination case there is the problem of losing minimality, we combined the reduction pair processors based on needed rules with the $\mathcal{A}$-transformation in such a way that minimality is preserved. Finally, we also integrated argument filters where there are even less needed rules, and where the rules only have to be π-proper, i.e., only those parts must be proper which are not dropped by π. Since these processors always simplify a DP problem, one should apply them as long as possible.

Our work extends the result of [KKSV96] that one can apply the $\mathcal{A}$-transformation for full rewriting if the whole TRS is proper. An extension to the DP framework and to the innermost case has already been published by us in [GTS05b]. However, in contrast to [GTS05b] here we consider $\mathcal{Q}$-restricted rewriting, we give all the details about the combination of the $\mathcal{A}$-transformation with the reduction pair processors, and we integrated argument filters.

Sometimes, the resulting DP problems are not even π-proper, or at least not π-proper for a suitable choice of π. In that case one might try the alternative transformation of [HM06b]. There, every applicative TRS can be translated into a TRS in functional form whenever all left-hand sides of the applicative TRS do not contain "variable applications". This criterion is strictly more liberal than *proper* but it is incomparable with π-*proper*. The main idea in [HM06b] is to uncurry the terms as far as possible and add a finite set of uncurrying rules to the resulting TRS. For example, the applicative TRS

$$\begin{array}{rcl}
\mathsf{id}\,'x & \to & x \\
\mathsf{plus}\,'\mathsf{0} & \to & \mathsf{id} \\
\mathsf{plus}\,'(\mathsf{s}\,'x)\,'y & \to & \mathsf{s}\,'(\mathsf{plus}\,'x\,'y)
\end{array}$$

is not proper and also not π-proper since there are different arities for plus. But the transformation of [HM06b] is applicable and it produces the following TRS.

$$\begin{array}{rcl}
\mathsf{id}_1(x) & \to & x \\
\mathsf{plus}_1(\mathsf{0}) & \to & \mathsf{id}_0 \\
\mathsf{plus}_2(\mathsf{s}_1(x), y) & \to & \mathsf{s}_1(\mathsf{plus}_2(x, y)) \\
\mathsf{plus}_1(x)\,'y & \to & \mathsf{plus}_2(x, y) \\
\mathsf{plus}_0\,'x & \to & \mathsf{plus}_1(x) \\
\mathsf{id}_0\,'x & \to & \mathsf{id}_1(x) \\
\mathsf{s}_0\,'x & \to & \mathsf{s}_1(x)
\end{array}$$

Comparing [HM06b] with our results there are some differences. The clear benefit of [HM06b] is that there are no arity-requirements and no requirements on the right-hand sides. On the other hand, if our transformation is applicable then we produce a smaller resulting system than [HM06b]. However, the main benefit of our results is the complete integration in the DP framework including the handling of $\mathcal{Q}$-restricted rewriting. (Therefore, we usually only have to transform a small subset of the original system.) Hence, it remains as an interesting question how to integrate [HM06b] to the DP framework.

Looking at other related work we see that most approaches for higher-order functions in term rewriting use higher-order TRSs. While there exist powerful termination criteria for higher-order TRSs (e.g., [Bla04, Pol96]), the main automated termination techniques for such TRSs are simplification orders (e.g., [JR07]) which fail on functions like div in Example 6.1.

Exceptions are the monotonic higher-order semantic path order [BR01] and the existing variants of dependency pairs for higher-order TRSs. However, these variants require considerable restrictions (e.g., on the TRSs [SK05] or on the orders that may be used [AY05, Kus01, SWS01]). So in contrast to our results, they are less powerful than the original dependency pair technique when applied to first-order functions.

Termination techniques for higher-order TRSs often handle a richer language than our results. But these approaches are usually difficult to automate (there are only few implementations of these techniques available). In contrast, it is very easy to integrate our results into existing termination provers for ordinary first-order TRSs using dependency pairs (and first-order reduction orders).

Other approaches [AY03, AY04, LB98, Toy04] represent higher-order functions by first-order TRSs, similar to us. However, they mostly use monomorphic types (this restriction is also imposed in some approaches for higher-order TRSs [BR01]). In other words, there the types are only built from basic types and type constructors like → or ×, but there are no type variables, i.e., no polymorphic types. Then terms like "$\mathsf{map}\,'\,\mathsf{minus}\,'\,xs$" and "$\mathsf{map}\,'\,(\mathsf{minus}\,'\,x)\,'\,xs$" cannot both be well typed, but one needs different map-symbols for arguments of different types. In contrast, our approach uses untyped term rewriting. Hence, it can be applied for termination analysis of polymorphic or untyped functional languages. Moreover, [LB98] and [Toy04] only consider extensions of the lexicographic path order, whereas we can also handle non-simply terminating TRSs like Example 6.1.

7. Processors Based on Semantic Labeling

Semantic labeling is a technique developed by Zantema [Zan95] to simplify termination proving. The main advantage is that by labeling one can rename different occurrences of the same function symbol f apart into $f_{l_1}, \ldots, f_{l_n}$ by attaching different labels l_1 to l_n. Then for example one can filter each f_{l_i} differently when solving constraints. However, to label a TRS one first has to find semantics for the TRS. For every symbol one needs an interpretation such that all rules are a model (or a quasi-model) for the given interpretation.

Of course, we want to lift semantic labeling from full rewriting to $\mathcal{Q}$-restricted rewriting and from TRSs to DP problems. Concerning the strategy there already has been work by us about semantic labeling for innermost rewriting in [TM07]. However, that work does not allow quasi-models and there are no completeness criteria given. In contrast we will show novel methods for $\mathcal{Q}$-restricted rewriting which are always complete for models, and even quasi-models are allowed and under certain conditions also complete.

For the switch from TRSs to DP problems the main problem is the minimality flag. Here, again we will show ways to preserve minimality in the model case, and we will give some sufficient conditions in the quasi-model case. An important new result is that for full rewriting minimality can always be carried over.

The chapter is organized as follows. In Section 7.1 we recapitulate the ingredients of semantic labeling in more detail, and we investigate how to extend semantic labeling for DP problems in the model case. The same extension is then performed in Section 7.2 for the quasi-model case. Finally, in Section 7.3 we show a way where semantic labeling always is complete and always preserves minimality, even for quasi-models and arbitrary strategies. The idea is to label a DP problem, simplify it, and then remove the labels afterwards.

7.1. Semantic Labeling with Models

Definition 7.1 (Models and Labelings). *Let $\mathcal{F}$ be a signature. An $\mathcal{F}$-*algebra *$\mathcal{M}$ consists of a set M, called the* carrier*, together with a set of* interpretations *$f_{\mathcal{M}} : M^n \to M$, one for each $f \in \mathcal{F}$.*

Given the set of variables $\mathcal{V}$ each variable assignment *$\beta : \mathcal{V} \to M$ induces the term* evaluation *$[\beta] : \mathcal{T}(\mathcal{F}, \mathcal{V}) \to M$ as follows.*

- $[\beta](x) = \beta(x)$
- $[\beta](f(t_1, \ldots, t_n)) = f_{\mathcal{M}}([\beta](t_1), \ldots, [\beta](t_n))$

An $\mathcal{F}$-algebra $\mathcal{M}$ is a model *of a set of rules $\mathcal{R}$ iff for every rule $\ell \to r \in \mathcal{R}$ and every variable assignment β the equation $[\beta](\ell) = [\beta](r)$ is satisfied.*

For each $f \in \mathcal{F}$ with arity n let L_f be a non-empty set of labels*, and let $\lambda_f : M^n \to L_f$ be a* labeling map*. Then the* labeled signature *is $\overline{\mathcal{F}} = \{f_l \mid f \in \mathcal{F}, l \in L_f\}$, and the* labeling function *$Lab : \mathcal{T}(\mathcal{F}, \mathcal{V}) \times (\mathcal{V} \to M) \to \mathcal{T}(\overline{\mathcal{F}}, \mathcal{V})$ is defined as follows.*

- $Lab(x, \beta) = x$
- $Lab(f(t_1, \dots, t_n), \beta) = f_{\lambda_f([\beta](t_1),\dots,[\beta](t_n))}(Lab(t_1, \beta), \dots, Lab(t_n, \beta))$

Labeling is extended to rewrite rules, sets of terms, and pair-graphs as follows. For a TRS $\mathcal{R}$ over the signature $\mathcal{F}$ the labeled TRS is defined as $\overline{\mathcal{R}} = \{Lab(\ell, \beta) \to Lab(r, \beta) \mid \ell \to r \in \mathcal{R}, \beta : \mathcal{V} \to M\}$, for a set of terms $\mathcal{Q}$ we define the labeled version as $\overline{\mathcal{Q}} = \{Lab(q, \beta) \mid q \in \mathcal{Q}, \beta : \mathcal{V} \to M\}$, and for a pair-graph $\mathcal{P} = (N, E)$ we define the labeled version $\overline{\mathcal{P}} = (\overline{N}, \overline{E})$ where $(Lab(s, \beta) \to Lab(t, \beta), Lab(u, \beta') \to Lab(v, \beta')) \in \overline{E}$ iff $(s \to t, u \to v) \in E$.

Removing the labels is done by the function $Unlab : \mathcal{T}(\overline{\mathcal{F}}, \mathcal{V}) \to \mathcal{T}(\mathcal{F}, \mathcal{V})$:

- $Unlab(x) = x$
- $Unlab(f_l(t_1, \dots, t_n)) = f(Unlab(t_1), \dots, Unlab(t_n))$

From now on we often assume that some $\mathcal{F}$-algebra and corresponding label functions are given. There is the following main result about semantic labeling due to Zantema.

Lemma 7.2 (Labeling and Rewriting [Zan95, Lemma 3])**.** *Let $\mathcal{M}$ be a model of $\mathcal{R}$ and let $\beta : \mathcal{V} \to M$ be a variable assignment. Then*

$$t \to_{\mathcal{R}} t' \text{ implies } Lab(t, \beta) \to_{\overline{\mathcal{R}}} Lab(t', \beta)$$

and

$$t \to_{\overline{\mathcal{R}}} t' \text{ implies } Unlab(t) \to_{\mathcal{R}} Unlab(t')$$

By Lemma 7.2 one directly obtains the result that $\mathcal{R}$ and $\overline{\mathcal{R}}$ have the same termination behavior. The advantage of proving termination of the labeled TRS is that different occurrences of the same unlabeled symbol f may now be different labeled symbols f_{l_1} and f_{l_2}. This has many benefits, e.g., when having to solve term-constraints it is possible to use different interpretations or precedences for f_{l_1} and f_{l_2}, or in the estimation of the dependency graph one may obtain less edges. The latter benefit is illustrated in more detail in Example 7.6.

The question now is how the result of Lemma 7.2 can be extended to the DP framework. Here, we have to solve three problems. The first one is how to deal with the additional pairs in $\mathcal{P}$. In [Ohl01] it is required that additionally to the rules of $\mathcal{R}$ also the pairs of $\mathcal{P}$ have to satisfy the model condition.[31] We show that it is already sufficient if one finds a model for the rules of $\mathcal{R}$ and does not impose any condition on $\mathcal{P}$. However, this is only a minor improvement as for standard DP problems, where all pairs are rooted with head symbols, one can just map every head symbol to the same element of the carrier. Then the model condition for the pairs of $\mathcal{P}$ is trivially satisfied. And since the labeled pairs and rules do not depend on the interpretations of the head symbols, fixing the interpretations for the head symbol does not restrict the set of resulting labeled systems one can get by semantic labeling.

---

[31] To be more precise in the [Ohl01, Corollary 1.5] semantic labeling is performed after argument filtering and it is directly combined with the basic reduction pair processor of Theorem 4.2, i.e., for a given argument filter π one has to find a model or quasi-model of the rewrite system $\pi(\mathcal{P} \cup \mathcal{R})$ and then solve the labeled constraints. However, since each interpretation of the filtered system can be extended to one over the original system such that one obtains a model or quasi-model of the original system, we can easily simulate the approach of [Ohl01] by first applying semantic labeling on the original system and then use a reduction pair processor afterwards.

The second problem is whether the minimality flag can be preserved. We will show as a new result that in general this is not the case, but will present different novel restrictions under which minimality is not lost.

The third problem we have to face is to handle the evaluation strategy given in $\mathcal{Q}$. Here, we generalize a result of [TM07] which shows how to perform semantic labeling for innermost rewriting. As we will see later in Section 7.2 even in the innermost case our generalization is more often applicable than using the results of [TM07].

We start with handling the third problem, i.e., we first show in Lemma 7.4 that indeed we can simulate every $\mathcal{Q}$-restricted rewrite step of the original system in the labeled system. Therefore, we first need a correspondence of normal forms which is formulated in the following lemma.

Lemma 7.3 (Labeling and Normal Forms)**.** *Let $\mathcal{Q}$ be a set of terms and β be a variable assignment. Then $t \in NF(\mathcal{Q})$ iff $Lab(t, \beta) \in NF(\overline{\mathcal{Q}})$.*

Note that we only get a result about normal forms of unlabeled terms and their (correctly) labeled versions, but there is no statement about arbitrary terms over the labeled signature and their unlabeled versions, i.e., in general $t \in NF(\overline{\mathcal{Q}})$ does not imply $Unlab(t) \in NF(\mathcal{Q})$. Therefore, we can only generalize the first result of Lemma 7.2 to $\mathcal{Q}$-restricted rewriting.

Lemma 7.4 (Labeling and $\mathcal{Q}$-Restricted Rewriting)**.** *Let $\mathcal{M}$ be a model of $\mathcal{R}$ and let $\beta : \mathcal{V} \to M$ be a variable assignment. Then*

$$t \overset{\mathcal{Q}}{\to}_{\mathcal{R}} t' \textit{ implies } Lab(t, \beta) \overset{\overline{\mathcal{Q}}}{\to}_{\overline{\mathcal{R}}} Lab(t', \beta)$$

The proof is a straightforward extension of Zantema's proof of Lemma 7.2 by integrating Lemma 7.3. However, the other direction that every reduction with the labeled system can be simulated by the unlabeled system is not true due to the reduction strategy given by $\mathcal{Q}$. This is later shown in Example 7.7 and Example 7.11. Nevertheless, using Lemma 7.4 one can prove the following theorem about a first processor based on semantic labeling.

Theorem 7.5 (Processors Based on Semantic Labeling)**.** *Let $\mathcal{M}$ be an $\mathcal{F}$-algebra, let L_f be a set of labels for every $f \in \mathcal{F}$, and let λ_f be the corresponding labeling map. The following processor Proc is sound. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, Proc returns*

- $\{(\overline{\mathcal{P}}, \overline{\mathcal{Q}}, \overline{\mathcal{R}}, \mathbf{a})\}$*, if $\mathcal{M}$ is a model of $\mathcal{R}$.*
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$*, otherwise*

The power of the semantic labeling processor is demonstrated in the following example.

Example 7.6. Let $\mathcal{Q} = \varnothing$ and let the TRS $\mathcal{R}$ consist of the following rules.

$$\begin{array}{rcl}
\mathsf{f}(\mathsf{true}, x) & \to & \mathsf{f}(\mathsf{odd}(\mathsf{double}(x)), x) \\
\mathsf{double}(\mathsf{0}) & \to & \mathsf{0} \\
\mathsf{double}(\mathsf{s}(x)) & \to & \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) \\
\mathsf{odd}(\mathsf{0}) & \to & \mathsf{false} \\
\mathsf{odd}(\mathsf{s}(\mathsf{0})) & \to & \mathsf{true} \\
\mathsf{odd}(\mathsf{s}(\mathsf{s}(x))) & \to & \mathsf{odd}(x)
\end{array}$$

Using the previous techniques we can remove all pairs in $\mathit{DP}(\mathcal{R})$ except for

$$\mathsf{F}(\mathsf{true}, x) \;\to\; \mathsf{F}(\mathsf{odd}(\mathsf{double}(x)), x) \tag{157}$$

One cannot prove finiteness of the DP problem $(\{(157)\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ with the previous techniques using standard orders. However, it is quite clear that the DP problem is finite as $2x$ is always an even number. This semantic knowledge can now be integrated into the DP problem when using the semantic labeling processor. To integrate the knowledge we use a modulo-2-counter. To be more precise, we use the algebra $\mathcal{M}$ with carrier $M = \{0, 1\}$ and $\mathsf{s}_{\mathcal{M}}(x) = (1 + x)\ \mathit{mod}\ 2$, $\mathsf{odd}_{\mathcal{M}}(x) = x$, $\mathsf{true}_{\mathcal{M}} = 1$, and $f_{\mathcal{M}}(\dots) = 0$ for all remaining function symbols f. Then indeed, $\mathcal{M}$ is a model of $\mathcal{R}$.

For the labeling we choose $L_{\mathsf{F}} = M$ and $L_f = \{\text{\textvisiblespace}\}$ for all remaining symbols f. Note that if L_f only contains one element $\text{\textvisiblespace}$ then the labeling function λ_f must always return $\text{\textvisiblespace}$ and hence, one always gets the same labeled function symbol $f_{\text{\textvisiblespace}}$ for f. So, we can identify $f_{\text{\textvisiblespace}}$ and f. In this sense whenever we speak of an *unlabeled symbol* f we mean that $L_f = \{\text{\textvisiblespace}\}$ and we write f instead of $f_{\text{\textvisiblespace}}$.

Hence, for the labeling we only have to specify the labeling function for F and we define it as $\lambda_{\mathsf{F}}(m_1, m_2) = m_1$. Then we obtain $\overline{\mathcal{R}} = \mathcal{R}$, $\overline{\mathcal{Q}} = \mathcal{Q} = \varnothing$, and $\overline{\mathcal{P}}$ consists of the pair

$$\mathsf{F}_1(\mathsf{true}, x) \;\to\; \mathsf{F}_0(\mathsf{odd}(\mathsf{double}(x)), x) \tag{158}$$

Since the two occurrences of the symbol F are now labeled apart into F_0 and F_1, it is easy to prove finiteness of the resulting DP problem $(\{(158)\}, \overline{\mathcal{Q}}, \overline{\mathcal{R}}, \mathbf{a})$ as there are no edges in the estimation of the dependency graph any more.

Note that techniques to find suitable models automatically have been developed in [KM07, KZ06, Zan05b].

Although the semantic labeling processor of Theorem 7.5 turns out to be quite useful, it also has two severe drawbacks. First, it does not preserve minimality and second, it is incomplete. The following example shows that this cannot be avoided in general.

Example 7.7. Let $\mathcal{R}$ consist of the following rules and let $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$.

$$\begin{aligned}
\mathsf{f}(\mathsf{a}, \mathsf{i}(x)) &\;\to\; \mathsf{g}(x, x)\\
\mathsf{g}(\mathsf{h}(x), y) &\;\to\; \mathsf{f}(x, \mathsf{k}(\mathsf{j}(\mathsf{h}(\mathsf{a}), y)))\\
\mathsf{k}(\mathsf{j}(x, y)) &\;\to\; \mathsf{i}(y)\\
\mathsf{j}(x, x) &\;\to\; \mathsf{b}\\
\mathsf{e}(x) &\;\to\; \mathsf{f}(\mathsf{a}, \mathsf{i}(\mathsf{h}(x)))
\end{aligned}$$

One can prove that $\mathcal{R}$ is innermost terminating which is equivalent to $\mathcal{Q}$-termination. The intuitive reason is that for an infinite reduction the x in the first rule must be instantiated by $\mathsf{h}(\mathsf{a})$. But then we obtain in the right hand-side of the second rule the term $\mathsf{f}(\mathsf{a}, \mathsf{k}(\mathsf{j}(\mathsf{h}(\mathsf{a}), \mathsf{h}(\mathsf{a}))))$. In this term we must reduce the subterm $\mathsf{j}(\mathsf{h}(\mathsf{a}), \mathsf{h}(\mathsf{a}))$ to b and hence, in the resulting term we cannot reduce $\mathsf{f}(\mathsf{a}, \mathsf{k}(\mathsf{b}))$ to an instance of the left-hand side of the first rule to obtain an infinite sequence.

A formal proof can also be done within the DP framework using our previous processors. There is only one SCC in the estimated dependency graph of the initial DP problem and this SCC contains the following two dependency pairs.

$$\begin{aligned}
\mathsf{F}(\mathsf{a}, \mathsf{i}(x)) &\;\to\; \mathsf{G}(x, x) && (159)\\
\mathsf{G}(\mathsf{h}(x), y) &\;\to\; \mathsf{F}(x, \mathsf{k}(\mathsf{j}(\mathsf{h}(\mathsf{a}), y))) && (160)
\end{aligned}$$

Then using the instantiation processor of Theorem 5.3 we see that the arguments of the left-hand side of (160) are identical and replace (160) by

$$\mathsf{G}(\mathsf{h}(x),\mathsf{h}(x)) \;\to\; \mathsf{F}(x,\mathsf{k}(\mathsf{j}(\mathsf{h}(\mathsf{a}),\mathsf{h}(x)))) \tag{161}$$

Moreover, using the forward instantiation processor of Theorem 5.5 we detect that the x in (161) has to be instantiated by a and hence, we replace (161) by

$$\mathsf{G}(\mathsf{h}(\mathsf{a}),\mathsf{h}(\mathsf{a})) \;\to\; \mathsf{F}(\mathsf{a},\mathsf{k}(\mathsf{j}(\mathsf{h}(\mathsf{a}),\mathsf{h}(\mathsf{a})))) \tag{162}$$

We can now perform the reduction of $\mathsf{j}(\dots)$ by the rewriting processor of Theorem 5.10 and replace (162) by[32]

$$\mathsf{G}(\mathsf{h}(\mathsf{a}),\mathsf{h}(\mathsf{a})) \;\to\; \mathsf{F}(\mathsf{a},\mathsf{k}(\mathsf{b})) \tag{163}$$

Finally, for the DP problem $(\{(159),(163)\},\mathcal{Q},\mathcal{R},f)$ the estimated dependency graph does not contain SCCs any more and $\mathcal{Q}$-termination is proved.

However, we can transform the original finite DP problem into an infinite one if we apply semantic labeling. We use the $\mathcal{F}$-algebra $\mathcal{M}$ with carrier $M = \{0,1\}$ and $f_{\mathcal{M}}(m_1,\dots,m_n) = 0$ for all $f \in \mathcal{F}$. Then clearly $\mathcal{M}$ is a model of $\mathcal{R}$. For the labeling we choose $L_{\mathsf{h}} = M$, $\lambda_{\mathsf{h}}(m) = m$, and the remaining function symbols are unlabeled. Hence, the labeled TRS $\overline{\mathcal{R}}$ consists of the following rules and $\overline{\mathcal{Q}} = \mathit{lhs}(\overline{\mathcal{R}})$.

$$\begin{aligned}
\mathsf{f}(\mathsf{a},\mathsf{i}(x)) &\;\to\; \mathsf{g}(x,x)\\
\mathsf{g}(\mathsf{h}_0(x),y) &\;\to\; \mathsf{f}(x,\mathsf{k}(\mathsf{j}(\mathsf{h}_0(\mathsf{a}),y)))\\
\mathsf{g}(\mathsf{h}_1(x),y) &\;\to\; \mathsf{f}(x,\mathsf{k}(\mathsf{j}(\mathsf{h}_0(\mathsf{a}),y)))\\
\mathsf{k}(\mathsf{j}(x,y)) &\;\to\; \mathsf{i}(y)\\
\mathsf{j}(x,x) &\;\to\; \mathsf{b}\\
\mathsf{e}(x) &\;\to\; \mathsf{f}(\mathsf{a},\mathsf{i}(\mathsf{h}_0(x)))\\
\mathsf{e}(x) &\;\to\; \mathsf{f}(\mathsf{a},\mathsf{i}(\mathsf{h}_1(x)))
\end{aligned}$$

We now show that $\overline{\mathcal{R}}$ is not $\overline{\mathcal{Q}}$-terminating any more. The reason is that the equivalence in Lemma 7.3 is too weak to simulate every reduction in the labeled system by a corresponding reduction is the original system. The problem is that not every term of $\mathcal{T}(\overline{\mathcal{F}},\mathcal{V})$ is correctly labeled, i.e., is a member of $\overline{\mathcal{T}(\mathcal{F},\mathcal{V})}$. For the ill-labeled term $t = \mathsf{f}(\mathsf{a},\mathsf{i}(\mathsf{h}_1(\mathsf{a})))$ we obtain the cyclic reduction

$$\begin{aligned}
t &\xrightarrow{\overline{\mathcal{Q}}}_{\overline{\mathcal{R}}} \mathsf{g}(\mathsf{h}_1(\mathsf{a}),\mathsf{h}_1(\mathsf{a}))\\
&\xrightarrow{\overline{\mathcal{Q}}}_{\overline{\mathcal{R}}} \mathsf{f}(\mathsf{a},\mathsf{k}(\mathsf{j}(\mathsf{h}_0(\mathsf{a}),\mathsf{h}_1(\mathsf{a}))))\\
&\xrightarrow{\overline{\mathcal{Q}}}_{\overline{\mathcal{R}}} \mathsf{f}(\mathsf{a},\mathsf{i}(\mathsf{h}_1(\mathsf{a}))) = t
\end{aligned}$$

This reduction is only possible as we now have two labeled variants of $\mathsf{h}(\mathsf{a})$. The correct variant $\mathsf{h}_0(\mathsf{a})$ is in the right-hand sides of the g-rules and the ill-labeled variant $\mathsf{h}_1(\mathsf{a})$ is hidden in the substitution. As these terms are not equal, the blocking effect of the term $\mathsf{j}(x,x) \in \overline{\mathcal{Q}}$ is useless.

To conclude, if we apply the semantic labeling processor on the non-infinite DP problem $(\varnothing,\mathcal{Q},\mathcal{R},f)$ we obtain the infinite DP problem $(\varnothing,\overline{\mathcal{Q}},\overline{\mathcal{R}},\mathsf{a})$ which shows that the semantic labeling processor is incomplete in general.

---

[32] Note that the requirements for the rewriting processor are indeed satisfied as the only usable rule is $\mathsf{j}(x,x) \to \mathsf{b}$.

Note, that the problem is even worse. Even if we start from correctly labeled terms, the labeled TRS $\overline{\mathcal{R}}$ allows to produce ill-labeled terms. As an example consider the term $\mathsf{e}(\mathsf{a})$ which is correctly labeled. However, $\mathsf{e}(\mathsf{a})$ can be reduced to the ill-labeled and non-terminating term t by the last rule of $\overline{\mathcal{R}}$. Using this effect it is now easy to give a DP problem which proves that minimality cannot be preserved by the semantic labeling processor. If we choose $\mathcal{P} = \{\mathsf{D}(x) \to \mathsf{D}(\mathsf{e}(\mathsf{a}))\}$ then clearly there is an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain as $\mathcal{R}$ is $\mathcal{Q}$-terminating. But as argued above the right-hand side $\mathsf{D}(\mathsf{e}(\mathsf{a}))$ of the only pair in $\overline{\mathcal{P}}$ is not $\overline{\mathcal{Q}}$-terminating w.r.t. $\overline{\mathcal{R}}$.

We will now present two alternatives to strengthen the semantic labeling processor of Theorem 7.5. One problem that occurred in Example 7.7 was the ability of the labeled system to transform correctly labeled terms to ill-labeled ones. This forced us in Theorem 7.5 to drop minimality. Reconsider the last two rules of the labeled system $\overline{\mathcal{R}}$ in Example 7.7. For each of the possible variable assignments of x there is a corresponding rule resulting in the two different right-hand sides $\mathsf{f}(\mathsf{a}, \mathsf{i}(\mathsf{h}_0(x)))$ and $\mathsf{f}(\mathsf{a}, \mathsf{i}(\mathsf{h}_1(x)))$. However, in the corresponding left-hand sides $\mathsf{e}_{\lambda_\mathsf{e}(0)}(x) = \mathsf{e}_{\llcorner\!\lrcorner}(x)$ and $\mathsf{e}_{\lambda_\mathsf{e}(1)}(x) = \mathsf{e}_{\llcorner\!\lrcorner}(x)$ we cannot detect the different assignments for x any more. The labeling map λ_e ignores its argument and always produces the same label $\llcorner\!\lrcorner$. But if we would have labeled e in the same way as h then there would not be a problem any more. The corresponding rules

$$
\begin{array}{lcl}
\mathsf{e}_0(x) & \to & \mathsf{f}(\mathsf{a}, \mathsf{i}(\mathsf{h}_0(x))) \\
\mathsf{e}_1(x) & \to & \mathsf{f}(\mathsf{a}, \mathsf{i}(\mathsf{h}_1(x)))
\end{array}
$$

can reduce the correctly labeled term $\mathit{Lab}(\mathsf{e}(\mathsf{a}), \beta) = \mathsf{e}_0(\mathsf{a})$ only to the correctly labeled term $\mathsf{f}(\mathsf{a}, \mathsf{i}(\mathsf{h}_0(\mathsf{a})))$ which is $\overline{\mathcal{Q}}$-terminating w.r.t. $\overline{\mathcal{R}}$.

The idea that the labeling should somehow contain the full information of the evaluation of their arguments is extended to the following definition. Then in Lemma 7.9 it is shown that for these labelings one cannot reduce correctly labeled terms to ill-labeled terms. This will allow us to keep minimality in the semantic labeling processor.

Definition 7.8 (Full Labeling)**.** *Let $\mathcal{M}$ be an $\mathcal{F}$-algebra with carrier M. Then for* full labeling *the labels are fixed to $L_f = M^n$ and the labeling maps are fixed to $\lambda_f(m_1, \ldots, m_n) = (m_1, \ldots, m_n)$ for every f with arity n.*

Choosing the full labeling has some benefits. It always produces the most distinct labeled systems, i.e., if for any labeling the rules $\mathit{Lab}(\ell, \beta) \to \mathit{Lab}(r, \beta)$ and $\mathit{Lab}(\ell, \beta') \to \mathit{Lab}(r, \beta')$ are different then they are also different when using the full labeling. Even more, with full labeling one does not produce rules twice, i.e., for two different variable assignments β and β' which differ on the variables of a rule $\ell \to r$ the rules $\mathit{Lab}(\ell, \beta) \to \mathit{Lab}(r, \beta)$ and $\mathit{Lab}(\ell, \beta') \to \mathit{Lab}(r, \beta')$ are different. Hence, with full labeling one gets the most distinct labeled system.

One disadvantage of full labeling is that the labeled systems are rather large which may become a problem for the automation. However, if one does not use full labeling then one also has a problem to mechanize semantic labeling: In addition to the problem of finding suitable interpretations, one has to come up with suitable labeling functions.

But the most important benefit of full labeling is the fact, that one cannot produce ill-labeled terms if the starting term is correctly labeled. This is shown in the upcoming Lemma 7.9 (ii) and it will allow us to preserve termination of a term by labeling. Therefore, we are able to carry over minimality in Theorem 7.10, an improved version of Theorem 7.5.

Lemma 7.9 (Properties of Full Labeling)**.** *Let $\mathcal{M}$ be an $\mathcal{F}$-algebra. If terms are labeled by full labeling then the following statements are valid.*

(i) Let x be a variable and β and β' be two variable assignments that differ on x, i.e., $\beta(x) \neq \beta'(x)$. For every non-variable term t with $x \in \mathcal{V}(t)$ and for every two substitution σ and σ' the terms $Lab(t,\beta)\sigma$ and $Lab(t,\beta')\sigma'$ are different.

(ii) If $Lab(t,\beta) \to_{\overline{\mathcal{R}}} s$ then $s = Lab(t',\beta)$ for some t' with $t \to_{\mathcal{R}} t'$.

Theorem 7.10 (Processors Based on Full Labeling)**.** *Let $\mathcal{M}$ be an $\mathcal{F}$-algebra, and let terms be labeled by full labeling. The following processor Proc is sound. For a DP problem $(\mathcal{P},\mathcal{Q},\mathcal{R},f)$, Proc returns*

- *$\{(\overline{\mathcal{P}},\overline{\mathcal{Q}},\overline{\mathcal{R}},f)\}$, if $\mathcal{M}$ is a model of $\mathcal{R}$.*
- *$\{(\mathcal{P},\mathcal{Q},\mathcal{R},f)\}$, otherwise*

It remains an open problem whether the processor of Theorem 7.10 is complete. To prove such a result one would need to transform infinite reductions in the labeled DP problem to infinite reductions in the original DP problem. Here, the problem is that one can start reductions of $\overline{\mathcal{R}}$ with arbitrary (ill-labeled) terms and as demonstrated in Example 7.7 these may have infinite reductions although the unlabeled terms are terminating. However, all the counter-examples like Example 7.7 and the upcoming example Example 7.11 do not work any more if one uses full labeling.

Hence, up to now it is unclear whether full labeling is complete. Therefore, we still are looking for conditions under which semantic labeling is complete. If we reconsider Example 7.7 there was a problem with the non-linear term $\mathsf{j}(x,x) \in \mathcal{Q}$. In the original TRS we encountered the subterm $\mathsf{k}(\mathsf{j}(\mathsf{h}(\mathsf{a}),\mathsf{h}(\mathsf{a})))$ which cannot be reduced by the rule $\mathsf{k}(\mathsf{j}(x,y)) \to \mathsf{i}(y)$ as the subterm $\mathsf{j}(\mathsf{h}(\mathsf{a}),\mathsf{h}(\mathsf{a}))$ is not in $\mathcal{Q}$-normal form. However, in the labeled system we had two different arguments of j. The term $\mathsf{k}(\mathsf{j}(\mathsf{h}_0(\mathsf{a}),\mathsf{h}_1(\mathsf{a})))$ can be reduced to $\mathsf{i}(\mathsf{h}_1(\mathsf{a}))$ as the ill-labeled subterm $\mathsf{h}_1(\mathsf{a})$ is different from $\mathsf{h}_0(\mathsf{a})$ and hence, the term $\mathsf{j}(x,x) \in \overline{\mathcal{Q}}$ does not prohibit the reduction.

Therefore, a natural question is whether the semantic labeling processor is complete if $\mathcal{Q}$ is linear. Unfortunately, the following example shows that linearity is neither sufficient for completeness nor does it allow to preserve the minimality flag.

Example 7.11. Let $\mathcal{R}$ consist of the following rules and let $\mathcal{Q} = lhs(\mathcal{R})$.

$$\begin{array}{rcl}
\mathsf{f}(\mathsf{a},x) & \to & \mathsf{f}(\mathsf{g}(x),x) \\
\mathsf{g}(\mathsf{h}(x)) & \to & \mathsf{i}(x) \\
\mathsf{i}(\mathsf{a}) & \to & \mathsf{a} \\
\mathsf{h}(\mathsf{a}) & \to & \mathsf{a} \\
\mathsf{e}(x) & \to & \mathsf{f}(\mathsf{a},\mathsf{h}(x))
\end{array}$$

One can prove that $\mathcal{R}$ is $\mathcal{Q}$-terminating in a similar way as in Example 7.7. The intuitive reason is that the x in the first rule must be instantiated by $\mathsf{h}(\mathsf{a})$ which is not allowed as this term is contained in $\mathcal{Q}$.

A formal proof within the DP framework is also possible. After applying the processors based on the dependency graph one remains with only one dependency pair

$$\mathsf{F}(\mathsf{a},x) \to \mathsf{F}(\mathsf{g}(x),x)$$

This pair can be replaced by the narrowing processor of Theorem 5.19 by

$$\mathsf{F}(\mathsf{a}, \mathsf{h}(x)) \;\rightarrow\; \mathsf{F}(\mathsf{i}(x), \mathsf{h}(x))$$

At this point the estimated dependency graph detects that there are no SCCs for the remaining DP problem and $\mathcal{Q}$-termination is proved.

But again, if we use the same $\mathcal{F}$-algebra, labeled signature, and labeling map as in Example 7.7 we obtain the following labeled rules $\overline{\mathcal{R}}$ and $\overline{\mathcal{Q}} = lhs(\overline{\mathcal{R}})$.

$$\begin{array}{rcl}
\mathsf{f}(\mathsf{a}, x) & \rightarrow & \mathsf{f}(\mathsf{g}(x), x) \\
\mathsf{g}(\mathsf{h}_0(x)) & \rightarrow & \mathsf{i}(x) \\
\mathsf{g}(\mathsf{h}_1(x)) & \rightarrow & \mathsf{i}(x) \\
\mathsf{i}(\mathsf{a}) & \rightarrow & \mathsf{a} \\
\mathsf{h}_0(\mathsf{a}) & \rightarrow & \mathsf{a} \\
\mathsf{e}(x) & \rightarrow & \mathsf{f}(\mathsf{a}, \mathsf{h}_0(x)) \\
\mathsf{e}(x) & \rightarrow & \mathsf{f}(\mathsf{a}, \mathsf{h}_1(x))
\end{array}$$

Hence, we can start an infinite reduction starting in the correctly labeled term $\mathsf{e}(\mathsf{a})$ as follows.

$$\begin{array}{rl}
 & \mathsf{e}(\mathsf{a}) \\
\xrightarrow{\overline{\mathcal{Q}}}_{\overline{\mathcal{R}}} & \mathsf{f}(\mathsf{a}, \mathsf{h}_1(\mathsf{a})) \\
\xrightarrow{\overline{\mathcal{Q}}}_{\overline{\mathcal{R}}} & \mathsf{f}(\mathsf{g}(\mathsf{h}_1(\mathsf{a})), \mathsf{h}_1(\mathsf{a})) \\
\xrightarrow{\overline{\mathcal{Q}}}_{\overline{\mathcal{R}}} & \mathsf{f}(\mathsf{i}(\mathsf{a}), \mathsf{h}_1(\mathsf{a})) \\
\xrightarrow{\overline{\mathcal{Q}}}_{\overline{\mathcal{R}}} & \mathsf{f}(\mathsf{a}, \mathsf{h}_1(\mathsf{a})) \\
\xrightarrow{\overline{\mathcal{Q}}}_{\overline{\mathcal{R}}} & \ldots
\end{array}$$

Thus, as in Example 7.7 neither is semantic labeling complete nor does it preserve minimality, even if $\mathcal{Q}$ is linear.

The problem in Example 7.11 is that $\overline{\mathcal{Q}}$ only contains the correctly labeled term $\mathsf{h}_0(\mathsf{a})$ but not the incorrect version $\mathsf{h}_1(\mathsf{a})$. To solve this problem we benefit from the flexibility that $\mathcal{Q}$ is just a set of terms which is unrelated to $\mathcal{R}$. This allows us to put more terms into the labeled version of $\mathcal{Q}$ than the correctly labeled ones, but we do not have to add corresponding ill-labeled rules. This idea leads to the following definition.

Definition 7.12 ($\underline{\mathcal{Q}}$)**.** *Let $\mathcal{Q} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$ and let $\overline{\mathcal{F}}$ be the labeled signature. Then we define $\underline{\mathcal{Q}}$ as*

$$\underline{\mathcal{Q}} = \{q \in \mathcal{T}(\overline{\mathcal{F}}, \mathcal{V}) \mid \mathit{Unlab}(q) \in \mathcal{Q}\}$$

The benefit of $\underline{\mathcal{Q}}$ is that one obtains a result similar to Lemma 7.2 provided that $\mathcal{Q}$ is linear. Before we present the corresponding Lemma 7.14 we need the following result that connects normal forms of $\mathcal{Q}$ with normal forms of $\underline{\mathcal{Q}}$.

Lemma 7.13 (Labeling and Normal Forms)**.** *Let β be an arbitrary variable assignment. Then $t \in \mathit{NF}(\mathcal{Q})$ implies $\mathit{Lab}(t, \beta) \in \mathit{NF}(\underline{\mathcal{Q}})$ and if $\mathcal{Q}$ is linear then $t \in \mathit{NF}(\underline{\mathcal{Q}})$ implies $\mathit{Unlab}(t) \in \mathit{NF}(\mathcal{Q})$.*

Lemma 7.14 (Labeling and $\mathcal{Q}$-Restricted Rewriting)**.** *Let $\mathcal{R}$ be a TRS over the signature $\mathcal{F}$ and let $\mathcal{Q} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$. If $\mathcal{M}$ is a model of $\mathcal{R}$ then for every variable assignment $\beta : \mathcal{V} \rightarrow M$*

$$t \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t' \text{ implies } \mathit{Lab}(t, \beta) \xrightarrow{\underline{\mathcal{Q}}}_{\overline{\mathcal{R}}} \mathit{Lab}(t', \beta)$$

and if $\mathcal{Q}$ *is linear then*

$$t \xrightarrow{\underline{\mathcal{Q}}}_{\overline{\mathcal{R}}} t' \textit{ implies } \textit{Unlab}(t) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \textit{Unlab}(t')$$

As for Lemma 7.4, the proof of Lemma 7.14 is an easy extension of the proof of Lemma 7.2 by using Lemma 7.13.

Now we can again present an improved version of the semantic labeling processor of Theorem 7.5 which is complete and preserves minimality if $\mathcal{Q}$ is linear.

Theorem 7.15 (Processors Based on Semantic Labeling). *Let* $\mathcal{M}$ *be an* $\mathcal{F}$*-algebra, let* L_f *be a set of labels for every* $f \in \mathcal{F}$*, and let* λ_f *be the corresponding labeling map. The following processor Proc is sound. For a DP problem* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$*, Proc returns*

- $\{(\overline{\mathcal{P}}, \underline{\mathcal{Q}}, \overline{\mathcal{R}}, f)\}$*, if* $\mathcal{M}$ *is a model of* $\mathcal{R}$ *and if* $\mathcal{Q}$ *is linear. In this case Proc is complete.*
- $\{(\overline{\mathcal{P}}, \underline{\mathcal{Q}}, \overline{\mathcal{R}}, \mathbf{a})\}$*, if* $\mathcal{M}$ *is a model of* $\mathcal{R}$ *and if* $\mathcal{Q}$ *is not linear.*
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$*, otherwise*

An obvious question is whether replacing $\overline{\mathcal{Q}}$ by $\underline{\mathcal{Q}}$ is already sufficient to obtain completeness and to preserve minimality, i.e., whether the additional requirement of linearity of $\mathcal{Q}$ is really required. As Example 7.11 only demonstrates that linearity on its own is not sufficient, this question is not answered yet. But a look at Example 7.7 will reveal the answer. It turns out that the given infinite reduction of the correctly labeled term $\mathsf{e}(\mathsf{a})$ is still possible if one replaces $\overline{\mathcal{Q}}$ by $\underline{\mathcal{Q}}$.

Comparing the semantic labeling processors of Theorems 7.10 and 7.15 we see that each variant has its own benefits. In Theorem 7.10 there is no restriction to linear sets $\mathcal{Q}$ and hence, minimality can always be carried over. However, in Theorem 7.15 we have the flexibility to choose arbitrary labels and labeling maps and we are not forced to produce the rather large labeled DP problem that we obtain from full labeling, but this flexibility may also be a problem for the automation. Note that there is no difference in the completeness results of the two theorems. The reason is that of course one can reformulate Theorem 7.10 and replace $\overline{\mathcal{Q}}$ by $\underline{\mathcal{Q}}$. Then one also obtains completeness in case of a linear $\mathcal{Q}$ for the semantic labeling processor of Theorem 7.10. Thus, for the automation we suggest to use the processor of Theorem 7.10. However, the additional flexibility in Theorem 7.15 may be helpful when extending semantic labeling to *predictive labeling* [HM06a, TM07], a related method that is discussed in more detail in the summary.

7.2. Semantic Labeling with Quasi-Models

Sometimes the requirement that $\mathcal{M}$ is a model of $\mathcal{R}$ is hard to satisfy. Therefore, in [Zan95] there is another version of semantic labeling based on quasi-models. Given a partial order $\geq$ on M, instead of requiring the model-condition $[\beta](\ell) = [\beta](r)$ for all rules $\ell \to r$ and all variable assignments β one just demands $[\beta](\ell) \geq [\beta](r)$. However, the cost of this relaxation is that the interpretation and the labeling functions must be weakly monotonic and one has to add additional rules. The details are captured in the following definition.

Definition 7.16 (Quasi-Model [Zan95]). *Let $\mathcal{M}$ be an $\mathcal{F}$-algebra equipped with a partial order $\geq$ on M such that the interpretations are weakly monotone, i.e., for all $f \in \mathcal{F}$ and for all $a_1, \ldots, a_n, b_1, \ldots, b_n \in M$ with $a_i \geq b_i$ the inequality $f_{\mathcal{M}}(a_1, \ldots, a_n) \geq f_{\mathcal{M}}(b_1, \ldots, b_n)$ must be satisfied.*

Then $\mathcal{M}$ is a quasi-model *of a set of rules $\mathcal{R}$ iff for every rule $\ell \to r \in \mathcal{R}$ and every variable assignment β the inequality $[\beta](\ell) \geq [\beta](r)$ is satisfied.*

For each $f \in \mathcal{F}$ let L_f be a non-empty set of labels provided with a well-founded partial order $\geq$ on L_f. For each $f \in \mathcal{F}$ with arity n let $\lambda_f : M^n \to L_f$ be a labeling map which is weakly monotonic. Then the set of decreasing rules *$\mathcal{D}ecr$ consists of all rules*

$$f_l(x_1, \ldots, x_n) \to f_{l'}(x_1, \ldots, x_n)$$

for all $f \in \mathcal{F}$ and all $l, l' \in L_f$ with $l > l'$. Here, $>$ denotes the strict part of $\geq$.

From [Zan95] we know that if $\mathcal{M}$ is a quasi-model of $\mathcal{R}$ then termination of $\mathcal{R}$ is equivalent to termination of $\overline{\mathcal{R}} \cup \mathcal{D}ecr$. Of course, we again want to generalize this result to $\mathcal{Q}$-restricted rewriting and lift it from TRSs to DP problems as in Section 7.1. Note that in [TM07] it was shown that it is impossible to use quasi-models in combination with innermost rewriting. However, in the following we show how quasi-models can be used for $\mathcal{Q}$-restricted rewriting, even if $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$. This is possible only due to the flexibility of $\mathcal{Q}$-restricted rewriting where one has a separate set for the strategy. The problem for innermost rewriting are the decreasing rules. On the one hand they must be added to $\mathcal{R}$ to perform the necessary rewriting steps, but on the other hand they must not be added to $\mathcal{R}$ for the innermost test that all arguments of a redex are in normal form. However, this is not a problem for $\mathcal{Q}$-restricted rewriting where one can add the decreasing rules to $\mathcal{R}$ without adding them to $\mathcal{Q}$.

The main lemma provided by [Zan95] for quasi-models is the following.

Lemma 7.17 (Labeling and Rewriting [Zan95, Lemma 7]). *Let $\mathcal{M}$ be a quasi-model of $\mathcal{R}$ and let $\beta : \mathcal{V} \to M$ be a variable assignment. Then*

$$t \to_{\mathcal{R}} t' \text{ implies } \mathit{Lab}(t, \beta) \to_{\overline{\mathcal{R}}} \to^*_{\mathcal{D}ecr} \mathit{Lab}(t', \beta)$$

and

$$\begin{aligned} t \to_{\mathcal{D}ecr} t' &\text{ implies } \mathit{Unlab}(t) = \mathit{Unlab}(t') \qquad \text{and} \\ t \to_{\overline{\mathcal{R}}} t' &\text{ implies } \mathit{Unlab}(t) \to_{\mathcal{R}} \mathit{Unlab}(t') \end{aligned}$$

Note that not even the first part of Lemma 7.17 carries over to $\mathcal{Q}$-restricted rewriting. The problem is that $\mathcal{D}ecr$-rules must sometimes be applied although the arguments are not in $\mathcal{Q}$-normal form. Consider the TRS $\mathcal{R} = \{\mathsf{a} \to \mathsf{b}\}$, $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$, and the reduction $\mathsf{f}(\mathsf{a}, \mathsf{a}) \overset{\mathcal{Q}}{\to}_{\mathcal{R}} \mathsf{f}(\mathsf{a}, \mathsf{b})$. We choose the quasi-model with $M = \{0, 1\}, \mathsf{a}_{\mathcal{M}} = 1, \mathsf{b}_{\mathcal{M}} = 0, L_{\mathsf{f}} = M, \lambda_{\mathsf{f}}(m_1, m_2) = m_2$ together with the order $1 > 0$. Then $\mathcal{D}ecr$ consists of the rule $\mathsf{f}_1(x, y) \to \mathsf{f}_0(x, y)$ and $\mathit{Lab}(\mathsf{f}(\mathsf{a}, \mathsf{a}), \beta) = \mathsf{f}_1(\mathsf{a}, \mathsf{a}) \overset{\mathcal{Q}}{\to}_{\overline{\mathcal{R}}} \mathsf{f}_1(\mathsf{a}, \mathsf{b})$, but $\mathsf{f}_1(\mathsf{a}, \mathsf{b})$ cannot be reduced further to $\mathsf{f}_0(\mathsf{a}, \mathsf{b}) = \mathit{Lab}(\mathsf{f}(\mathsf{a}, \mathsf{b}), \beta)$ using $\overset{\mathcal{Q}}{\to}_{\mathcal{D}ecr}$ since a is not in normal form w.r.t. $\underline{\mathcal{Q}}$.

Therefore, in the first part of the following lemma we only consider reductions resulting in normal forms.

Lemma 7.18 (Labeling and $\mathcal{Q}$-Restricted Rewriting). *Let $\mathcal{M}$ be a quasi-model of $\mathcal{R}$ and let $\beta : \mathcal{V} \to M$ be a variable assignment. Then*

$$t \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} t' \text{ and } t' \in \mathit{NF}(\mathcal{Q}) \text{ implies } \mathit{Lab}(t, \beta) \overset{\underline{\mathcal{Q}}}{\to}{}^*_{\overline{\mathcal{R}} \cup \mathcal{D}ecr} \mathit{Lab}(t', \beta)$$

and if $\mathcal{Q}$ is linear then

$$\begin{array}{llll} t \xrightarrow{\mathcal{Q}}_{\mathcal{D}ecr} t' & \text{implies } Unlab(t) = & Unlab(t') & \text{and} \\ t \xrightarrow{\mathcal{Q}}_{\overline{\mathcal{R}}} t' & \text{implies } Unlab(t) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} & Unlab(t') & \end{array}$$

To adapt the semantic labeling processor of Theorem 7.15 to quasi-models is now straightforward.

Theorem 7.19 (Processors Based on Semantic Labeling with Quasi-Models)**.** *Let $\mathcal{M}$, L_f, λ_f, and $\geq$ as in Definition 7.16. The following processor Proc is sound. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, Proc returns*

- $\{(\overline{\mathcal{P}}, \underline{\mathcal{Q}}, \overline{\mathcal{R}} \cup \mathcal{D}ecr, f)\}$, *if $\mathcal{M}$ is a quasi-model of $\mathcal{R}$ and if $\mathcal{Q}$ is linear. In this case Proc is even complete.*
- $\{(\overline{\mathcal{P}}, \underline{\mathcal{Q}}, \overline{\mathcal{R}} \cup \mathcal{D}ecr, \mathbf{a})\}$, *if $\mathcal{M}$ is a quasi-model of $\mathcal{R}$ and if $\mathcal{Q}$ is not linear.*
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$, *otherwise*

Although the semantic labeling processor of Theorem 7.19 is complete and preserves minimality if $\mathcal{Q}$ is linear, it has some disadvantages. The reason is that due to the decreasing rules $\mathcal{D}ecr$ all previous head symbols are transformed to non-head symbols in the labeled DP problem. That this is a disadvantage can be seen in the following example.

Example 7.20. Let $\mathcal{R}$ consist of the following rules.

$$\begin{array}{lll} \mathsf{g}(x) & \to & x \\ \mathsf{g}(\mathsf{a}) & \to & \mathsf{c} \end{array}$$

For $\mathcal{P} = \{\mathsf{F}(\mathsf{a}) \to \mathsf{F}(\mathsf{g}(\mathsf{b}))\}$ and $\mathcal{Q} = \varnothing$ the $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-dependency graph is empty. However, this cannot be detected by the estimations presented in Section 3.1.

We use the algebra $\mathcal{M}$ with $M = \{0, 1, 2\}$ and $\mathsf{g}_{\mathcal{M}}(m) = \mathsf{F}_{\mathcal{M}}(m) = m$, $\mathsf{a}_{\mathcal{M}} = 2$, $\mathsf{b}_{\mathcal{M}} = 1$, and $\mathsf{c}_{\mathcal{M}} = 0$. Then $\mathcal{M}$ is a quasi-model of $\mathcal{R}$ if we use the standard order $\geq$ on M but it is not a model as

$$[\beta](\mathsf{g}(\mathsf{a})) = 2 > 0 = [\beta](\mathsf{c})$$

We choose $L_{\mathsf{F}} = M$ and $\lambda_{\mathsf{F}}(m) = m$, $L_{\mathsf{g}} = \{0, 1\}$ and $\lambda_{\mathsf{g}}(m) = min(1, m)$, and we do not label the remaining function symbols. Then one obtains the following labeled rules $\overline{\mathcal{R}}$.

$$\begin{array}{lll} \mathsf{g}_0(x) & \to & x \\ \mathsf{g}_1(x) & \to & x \\ \mathsf{g}_1(\mathsf{a}) & \to & \mathsf{c} \end{array}$$

The TRS $\mathcal{D}ecr$ contains the rules

$$\begin{array}{lll} \mathsf{F}_2(x) & \to & \mathsf{F}_1(x) \\ \mathsf{F}_2(x) & \to & \mathsf{F}_0(x) \\ \mathsf{F}_1(x) & \to & \mathsf{F}_0(x) \\ \mathsf{g}_1(x) & \to & \mathsf{g}_0(x) \end{array}$$

For $\overline{\mathcal{P}} = \{\mathsf{F}_2(\mathsf{a}) \to \mathsf{F}_1(\mathsf{g}_1(\mathsf{b}))\}$ it cannot be detected by the dependency graph estimations that there is no edge in the $(\overline{\mathcal{P}}, \underline{\mathcal{Q}}, \overline{\mathcal{R}} \cup \mathcal{D}ecr)$-dependency graph. The problem is that the rules in $\mathcal{D}ecr$ allow to rewrite a term $t = \mathsf{F}_1(\dots)$ at the root position and hence, applying an (estimated) *Cap*-function will replace t by a fresh variable. And as a fresh variable unifies with every left-hand side of pairs of $\overline{\mathcal{P}}$, no edges can be deleted. So even if the original pair would be $\mathsf{G}(\dots) \to \mathsf{F}(\mathsf{g}(\mathsf{b}))$ where G is a different head-symbol then after the labeling it would not be detectable that there is no connection in the dependency graph.

To handle the problem with the $\mathcal{D}ecr$-rules for head symbols a possibility is to split the $\mathcal{D}ecr$-rules into two parts. The part $\mathcal{D}ecr_{\mathcal{H}}$ contains the decreasing rules for the head symbols and $\mathcal{D}ecr_{\neg\mathcal{H}}$ consists of the remaining decreasing rules. In the previous example $\mathcal{D}ecr_{\mathcal{H}}$ consists of the first three rules and $\mathcal{D}ecr_{\neg\mathcal{H}}$ is the set with the single rule $\mathsf{g}_1(x) \to \mathsf{g}_0(x)$.

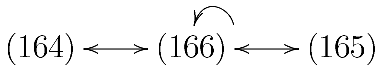
Now the idea is to add $\mathcal{D}ecr_{\mathcal{H}}$ to $\overline{\mathcal{P}}$ and $\mathcal{D}ecr_{\neg\mathcal{H}}$ to $\overline{\mathcal{R}}$. Then for the corresponding DP problem there are four pairs in $\overline{\mathcal{P}} \cup \mathcal{D}ecr_{\mathcal{H}}$ but every estimation can detect there are no SCCs in the dependency graph. The reason is that now F_0, F_1, and F_2 are head-symbols and are not replaced by fresh variables using an estimated *Cap*-function.

However, there also is a problem when adding $\mathcal{D}ecr_{\mathcal{H}}$ to $\overline{\mathcal{P}}$. Note that $\overline{\mathcal{P}}$ is not just a set of pairs but a graph (N, E). Clearly, one should add $\mathcal{D}ecr_{\mathcal{H}}$ to N but the question is which edges should be added to E. That this may be a problem for completeness is shown in the following example.

Example 7.21. Consider the DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$ where $\mathcal{R} = \{\mathsf{d} \to \mathsf{c}\}$, $\mathcal{Q} = \varnothing$ and where we have the following pairs.

$$\begin{aligned} \mathsf{F}(x, \mathsf{a}, \mathsf{c}) &\to \mathsf{F}(x, \mathsf{b}, \mathsf{d}) \qquad (164)\\ \mathsf{F}(x, \mathsf{b}, \mathsf{c}) &\to \mathsf{F}(x, \mathsf{a}, \mathsf{d}) \qquad (165)\\ \mathsf{F}(\mathsf{s}(x), y, \mathsf{c}) &\to \mathsf{F}(x, y, \mathsf{d}) \qquad (166) \end{aligned}$$

The structure of $\mathcal{P}$ is given in the following graph.



Note that there is no edge between (164) and (165) which prevents an infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain.[33]

We now use semantic labeling with the algebra $\mathcal{M}$ with carrier $M = \{0, 1\}$ and with interpretations $\mathsf{d}_{\mathcal{M}} = 1$ and $f_{\mathcal{M}}(\dots) = 0$ for all remaining function symbols f. Then $\mathcal{M}$ is a quasi-model of $\mathcal{R}$ using the usual order $\geq$ on M. We choose $L_{\mathsf{F}} = M$ and $\lambda_{\mathsf{F}}(m_1, m_2, m_3) = m_3$, and we do not label the remaining function symbols. In this way we obtain $\overline{\mathcal{R}} = \mathcal{R}$, $\mathcal{D}ecr_{\neg\mathcal{H}} = \underline{\mathcal{Q}} = \varnothing$, and the following labeled pairs.

$$\begin{aligned} \mathsf{F}_0(x, \mathsf{a}, \mathsf{c}) &\to \mathsf{F}_1(x, \mathsf{b}, \mathsf{d}) \qquad (167)\\ \mathsf{F}_0(x, \mathsf{b}, \mathsf{c}) &\to \mathsf{F}_1(x, \mathsf{a}, \mathsf{d}) \qquad (168)\\ \mathsf{F}_0(\mathsf{s}(x), y, \mathsf{c}) &\to \mathsf{F}_1(x, y, \mathsf{d}) \qquad (169) \end{aligned}$$

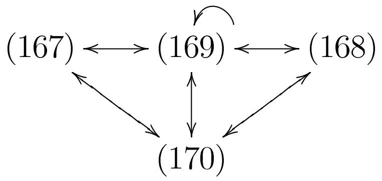
Moreover, there is one decreasing pair in $\mathcal{D}ecr_{\mathcal{H}} = \mathcal{D}ecr$.

$$\mathsf{F}_1(x, y, z) \to \mathsf{F}_0(x, y, z) \qquad (170)$$

---

[33] A problem like this may arise from applying the argument filter processor of Theorem 4.38.

Let us consider the graph structure of the union of $\overline{\mathcal{P}}$ and $\mathcal{D}ecr_{\mathcal{H}}$. The connections between (167), (168), and (169) are just copied from $\mathcal{P}$. To detect the required connections between (170) and the pairs of $\overline{\mathcal{P}}$ we consider the chains in the original DP problem. As each edge in $\mathcal{P}$ really corresponds to a chain, for correctness we must be able to simulate each of these chains in the labeled DP problem. For example, the chain $(164), (166), (165)$ can only be simulated by $(167), (170), (169), (170), (168)$. Hence, we must add incoming and outgoing edges of (170) to every pair of $\overline{\mathcal{P}}$. This results in the following graph.



The problem is that now there is a connection between (167) and (168) without using (169). And indeed

$$(167), (170), (168), (170), (167), \ldots$$

is an infinite $(\overline{\mathcal{P}} \cup \mathcal{D}ecr_{\mathcal{H}}, \underline{\mathcal{Q}}, \overline{\mathcal{R}} \cup \mathcal{D}ecr_{\neg\mathcal{H}})$-chain although the original problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$ is finite.

So, in Example 7.20 we have seen that adding all decreasing rules to $\overline{\mathcal{R}}$ – although complete – is not a good idea. The partitioning into $\mathcal{D}ecr_{\mathcal{H}}$ and $\mathcal{D}ecr_{\neg\mathcal{H}}$ helps to prevent the problem of Example 7.20 that the labeled versions of head symbols are no head symbols any more. But when adding $\mathcal{D}ecr_{\mathcal{H}}$ it can happen that we introduce connections that are not possible in the unlabeled DP problem resulting in an incomplete processor. This was demonstrated in Example 7.21.

The final solution will be to use the partitioning into $\mathcal{D}ecr_{\mathcal{H}}$ and $\mathcal{D}ecr_{\neg\mathcal{H}}$, but instead of building $\overline{\mathcal{P}} \cup \mathcal{D}ecr_{\mathcal{H}}$ we will directly combine the effect of $\mathcal{D}ecr_{\mathcal{H}}$ into the labeled version of $\mathcal{P}$. The idea is to built labeled pairs where the head symbols of the right-hand side are directly labeled with all smaller variants. So instead of creating the labeled pair $Lab(\beta, s) \to f_l(\ldots)$ and having the decreasing pair $f_l(\ldots) \to f_{l'}(\ldots)$ in $\mathcal{D}ecr_{\mathcal{H}}$, we now directly built the pair $Lab(\beta, s) \to f_{l'}(\ldots)$. Then we do not have to use $\mathcal{D}ecr_{\mathcal{H}}$ any more. This new construction is captured more formally in the following definition.

Definition 7.22 (Quasi-Labeled Pair-Graph). *Let the $\mathcal{F}$-algebra, the partial order $\geq$, the labels, and the labeling maps be as in Definition 7.16. Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$ be a DP problem with head symbols $\mathcal{H}$ and $\mathcal{P} = (N, E)$. We define the* quasi-labeled pair-graph *as $\underline{\mathcal{P}} = (\underline{N}, \underline{E})$ where*

- $\underline{N} = \{Lab(\beta, s) \to Lab(\beta, t) \mid s \to t \in N, root(t) \notin \mathcal{H}, \beta : \mathcal{V} \to M\} \cup$
  $\{Lab(\beta, s) \to f_{l'}(Lab(\beta, t_1), \ldots, Lab(\beta, t_n)) \mid s \to f(t_1, \ldots, t_n) \in N,$
  $f \in \mathcal{H}, \beta : \mathcal{V} \to M, l' \in L_f, \lambda_f([\beta](t_1), \ldots, [\beta](t_n)) \geq l'\}$
- $\underline{E} = \{(s, t) \in \underline{N} \times \underline{N} \mid (Unlab(s), Unlab(t)) \in E\}$

Now we can formulate an improved version of the semantic labeling processor for quasi-models in Theorem 7.19.

Theorem 7.23 (Processors Based on Semantic Labeling with Quasi-Models and Head Symbols). *Let $\mathcal{M}$, L_f, λ_f, and $\geq$ be as in Definition 7.16. The following processor Proc is sound. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, Proc returns*

- $\{(\underline{\mathcal{P}}, \underline{\mathcal{Q}}, \overline{\mathcal{R}} \cup \mathcal{D}ecr_{\neg\mathcal{H}}, f)\}$, *if* $\mathcal{M}$ *is a quasi-model of* $\mathcal{R}$, *if* $\mathcal{Q}$ *is linear, and if* $\mathcal{H}$ *is the set of head symbols of* $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$. *In this case Proc is even complete.*
- $\{(\underline{\mathcal{P}}, \underline{\mathcal{Q}}, \overline{\mathcal{R}} \cup \mathcal{D}ecr_{\neg\mathcal{H}}, \mathbf{a})\}$, *if* $\mathcal{M}$ *is a quasi-model of* $\mathcal{R}$, *if* $\mathcal{Q}$ *is not linear, and if* $\mathcal{H}$ *is the set of head symbols of* $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$.
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$, *otherwise.*

We first show that now the problems mentioned in Examples 7.20 and 7.21 are solved when using Theorem 7.23.

Example 7.24. If we apply the semantic labeling processor of Theorem 7.23 on the DP problem of Example 7.20 with the same quasi-model and the same labeling then we obtain the following TRS $\overline{\mathcal{R}} \cup \mathcal{D}ecr_{\neg\mathcal{H}}$.

$$\begin{array}{rcl}
\mathsf{g}_0(x) & \to & x \\
\mathsf{g}_1(x) & \to & x \\
\mathsf{g}_1(\mathsf{a}) & \to & \mathsf{c} \\
\mathsf{g}_1(x) & \to & \mathsf{g}_0(x)
\end{array}$$

Note that there are no decreasing rules for the head symbol F any more. Therefore, we now obtain two pairs in $\underline{\mathcal{P}}$.

$$\begin{array}{rcl}
\mathsf{F}_2(\mathsf{a}) & \to & \mathsf{F}_1(\mathsf{g}_1(\mathsf{b})) \\
\mathsf{F}_2(\mathsf{a}) & \to & \mathsf{F}_0(\mathsf{g}_1(\mathsf{b}))
\end{array}$$

For this resulting DP problem termination is trivially proven by the dependency graph which was not the case if we would have used Theorem 7.19.

Example 7.25. In this example we apply Theorem 7.23 on the problematic DP problem of Example 7.21, again using the same quasi-model and labeling function. There are no decreasing rules for the head symbols any more and we obtain the following new pairs.

$$\mathsf{F}_0(x, \mathsf{a}, \mathsf{c}) \to \mathsf{F}_1(x, \mathsf{b}, \mathsf{d}) \tag{171}$$

$$\mathsf{F}_0(x, \mathsf{a}, \mathsf{c}) \to \mathsf{F}_0(x, \mathsf{b}, \mathsf{d}) \tag{172}$$

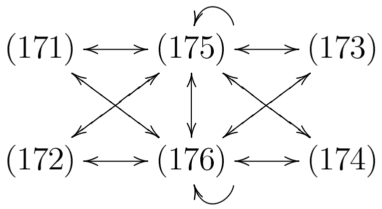
$$\mathsf{F}_0(x, \mathsf{b}, \mathsf{c}) \to \mathsf{F}_1(x, \mathsf{a}, \mathsf{d}) \tag{173}$$

$$\mathsf{F}_0(x, \mathsf{b}, \mathsf{c}) \to \mathsf{F}_0(x, \mathsf{a}, \mathsf{d}) \tag{174}$$

$$\mathsf{F}_0(\mathsf{s}(x), y, \mathsf{c}) \to \mathsf{F}_1(x, y, \mathsf{d}) \tag{175}$$

$$\mathsf{F}_0(\mathsf{s}(x), y, \mathsf{c}) \to \mathsf{F}_0(x, y, \mathsf{d}) \tag{176}$$

The graph of the new DP problem looks as follows.



Note that as in the input DP problem, in the labeled DP problem the only way to come from a pair on the left-hand side to one of the right-hand side (or vice versa) is if one also visits a pair in the middle. Thus, after removing the pairs (175) and (176) with the reduction pair processor, the processors based on the dependency graph finally prove finiteness. This is in contrast to labeled DP problem in Example 7.21 which is not finite.

The processor of Theorem 7.23 using quasi-models clearly corresponds to the processor of Theorem 7.15 for models. Both require that $\mathcal{Q}$ is linear for completeness and to carry over minimality. However, in the case of models we also had the processor based on full labelings of Theorem 7.10 which always carries over minimality regardless of $\mathcal{Q}$. Of course, it would be nice to have a corresponding processor for quasi-models, too. The following example shows that this is not possible.

Example 7.26. Let $\mathcal{R}$ consist of the following rules and let $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$.

$$
\begin{array}{rcl}
\mathsf{f}(x) & \to & \mathsf{g}(\mathsf{h}(x,x)) \\
\mathsf{g}(\mathsf{h}(x,y)) & \to & \mathsf{f}(x) \\
\mathsf{h}(x,x) & \to & \mathsf{i}(x)
\end{array}
$$

One can prove that $\mathcal{R}$ is innermost terminating which is equivalent to $\mathcal{Q}$-termination. The intuitive reason is that for an infinite reduction the right-hand side of the first rule will directly reduce to $\mathsf{g}(\mathsf{i}(x))$ and no instance of this term can be reduced by the second rule. As in the previous examples of this chapter we can also do the proof in the DP framework. One would first apply the rewriting processor of Theorem 5.10 on the dependency pair $\mathsf{F}(x) \to \mathsf{G}(\mathsf{h}(x,x))$ and replace this by $\mathsf{F}(x) \to \mathsf{G}(\mathsf{i}(x))$ and then the processors based on the dependency graph finish the termination proof.

We consider the infinite DP problem $(\{\mathsf{F}(x) \to \mathsf{F}(\mathsf{f}(\mathsf{i}(\mathsf{a})))\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$. The reason for being infinite is that it is possible to build an infinite chain. Due to innermost termination of $\mathcal{R}$ the term $\mathsf{f}(\mathsf{i}(\mathsf{a}))$ is $\mathcal{Q}$-terminating. And thus, one can instantiate the variable x of the left-hand side of the pair by some normal form of $\mathsf{f}(\mathsf{i}(\mathsf{a}))$ to obtain the infinite chain.

Now consider the quasi-model $\mathcal{M}$ with $M = \{0,1\}$, $\mathsf{a}_{\mathcal{M}} = \mathsf{h}_{\mathcal{M}}(\dots) = 1$, and $f_{\mathcal{M}}(\dots) = 0$ for all remaining function symbols f. If we use full labeling then the set $\underline{\mathcal{P}}$ consists of the pairs

$$
\begin{array}{rcl}
\mathsf{F}_0(x) & \to & \mathsf{F}_0(\mathsf{f}_0(\mathsf{i}_1(\mathsf{a}))) \\
\mathsf{F}_1(x) & \to & \mathsf{F}_0(\mathsf{f}_0(\mathsf{i}_1(\mathsf{a})))
\end{array}
$$

and $\overline{\mathcal{R}}$ contains the following rules.

$$
\begin{array}{rcl}
\mathsf{f}_0(x) & \to & \mathsf{g}_1(\mathsf{h}_{0,0}(x,x)) \\
\mathsf{f}_1(x) & \to & \mathsf{g}_1(\mathsf{h}_{1,1}(x,x)) \\
\mathsf{g}_1(\mathsf{h}_{0,0}(x,y)) & \to & \mathsf{f}_0(x) \\
\mathsf{g}_1(\mathsf{h}_{0,1}(x,y)) & \to & \mathsf{f}_0(x) \\
\mathsf{g}_1(\mathsf{h}_{1,0}(x,y)) & \to & \mathsf{f}_1(x) \\
\mathsf{g}_1(\mathsf{h}_{1,1}(x,y)) & \to & \mathsf{f}_1(x) \\
\mathsf{h}_{0,0}(x,x) & \to & \mathsf{i}_0(x) \\
\mathsf{h}_{1,1}(x,x) & \to & \mathsf{i}_1(x)
\end{array}
$$

The decreasing rules $\mathcal{D}ecr_{\neg\mathcal{H}}$ are

$$
\begin{array}{rcl}
\mathsf{f}_1(x) & \to & \mathsf{f}_0(x) \\
\mathsf{g}_1(x) & \to & \mathsf{g}_0(x) \\
\mathsf{h}_{0,1}(x) & \to & \mathsf{h}_{0,0}(x) \\
\mathsf{h}_{1,0}(x) & \to & \mathsf{h}_{0,0}(x)
\end{array}
$$

$$\begin{array}{rcl}
\mathsf{h}_{1,1}(x) & \to & \mathsf{h}_{0,0}(x) \\
\mathsf{h}_{1,1}(x) & \to & \mathsf{h}_{1,0}(x) \\
\mathsf{h}_{1,1}(x) & \to & \mathsf{h}_{0,1}(x) \\
\mathsf{i}_1(x) & \to & \mathsf{i}_0(x)
\end{array}$$

and the set $\underline{\mathcal{Q}}$ contains the following terms.

$$\begin{array}{r}
\mathsf{f}_0(x), \mathsf{f}_1(x), \\
\mathsf{g}_0(\mathsf{h}_{0,0}(x,y)), \mathsf{g}_0(\mathsf{h}_{0,1}(x,y)), \mathsf{g}_0(\mathsf{h}_{1,0}(x,y)), \mathsf{g}_0(\mathsf{h}_{1,1}(x,y)), \\
\mathsf{g}_1(\mathsf{h}_{0,0}(x,y)), \mathsf{g}_1(\mathsf{h}_{0,1}(x,y)), \mathsf{g}_1(\mathsf{h}_{1,0}(x,y)), \mathsf{g}_1(\mathsf{h}_{1,1}(x,y)), \\
\mathsf{h}_{0,0}(x,x), \mathsf{h}_{0,1}(x,x), \mathsf{h}_{1,0}(x,x), \mathsf{h}_{1,1}(x,x)
\end{array}$$

As the term $\mathsf{f}_0(\mathsf{i}_1(\mathsf{a}))$ is not $\underline{\mathcal{Q}}$ terminating w.r.t. $\overline{\mathcal{R}} \cup \mathcal{D}ecr_{\neg\mathcal{H}}$, there obviously is no minimal $(\underline{\mathcal{P}}, \underline{\mathcal{Q}}, \overline{\mathcal{R}} \cup \mathcal{D}ecr_{\neg\mathcal{H}})$-chain and hence, even with full-labeling and with $\underline{\mathcal{Q}}$ we cannot keep minimality. The problem is that in the infinite reduction

$$\begin{array}{rll}
\mathsf{f}_0(\mathsf{i}_1(\mathsf{a})) & \overset{\underline{\mathcal{Q}}}{\to}_{\overline{\mathcal{R}}} & \mathsf{g}_1(\mathsf{h}_{0,0}(\mathsf{i}_1(\mathsf{a}), \mathsf{i}_1(\mathsf{a}))) \\
& \overset{\underline{\mathcal{Q}}}{\to}_{\mathcal{D}ecr_{\neg\mathcal{H}}} & \mathsf{g}_1(\mathsf{h}_{0,0}(\mathsf{i}_1(\mathsf{a}), \mathsf{i}_0(\mathsf{a}))) \\
& \overset{\underline{\mathcal{Q}}}{\to}_{\overline{\mathcal{R}}} & \mathsf{f}_0(\mathsf{i}_1(\mathsf{a})) \\
& \overset{\underline{\mathcal{Q}}}{\to}_{\overline{\mathcal{R}}} & \ldots
\end{array}$$

we get an ill-labeled term $\mathsf{g}_1(\mathsf{h}_{0,0}(\mathsf{i}_1(\mathsf{a}), \mathsf{i}_0(\mathsf{a})))$ using the decreasing rules. Then the non-linear term $\mathsf{h}_{0,0}(x,x)$ does not prohibit the reduction to $\mathsf{f}_0(\mathsf{i}_1(\mathsf{a}))$ as it does in the unlabeled case. Hence, the main advantage of full-labeling – the fact that no ill-labeled terms are created – is destroyed by the decreasing rules.

7.3. Semantic Labeling and Unlabeling

In the previous two sections we have seen the technique of semantic labeling for models and quasi-models. The main benefit is that different occurrences of the same function symbols may become different symbols due to the labeling. And then processors based on the dependency graph and those based on orders often succeed although they have failed on the unlabeled DP problem.

However, there is one major drawback with semantic labeling: the labeled DP problems tend to get large, even when the carrier set size is only 2 or 3. This problem is magnified if one wants to use several different models in sequence. That this can be useful is demonstrated in the following example.

Example 7.27. We consider the same idea as in Example 7.6 where we needed to count modulo 2. In this example we need to count modulo 2 and modulo 3. The rules of $\mathcal{R}$ are as follows.

$$\begin{array}{rcl}
\mathsf{f}(\mathsf{false}, x) & \to & \mathsf{f}(\mathsf{even}(\mathsf{six}(x)), x) \\
\mathsf{f}(\mathsf{false}, x) & \to & \mathsf{f}(\mathsf{divThree}(\mathsf{six}(x)), x) \\
\mathsf{six}(\mathsf{0}) & \to & \mathsf{0} \\
\mathsf{six}(\mathsf{s}(x)) & \to & \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{six}(x))))))) \\
\mathsf{even}(\mathsf{s}(\mathsf{0})) & \to & \mathsf{false} \\
\mathsf{even}(\mathsf{s}(\mathsf{s}(x))) & \to & \mathsf{even}(x) \\
\mathsf{divThree}(\mathsf{s}(\mathsf{0})) & \to & \mathsf{false} \\
\mathsf{divThree}(\mathsf{s}(\mathsf{s}(\mathsf{s}(x)))) & \to & \mathsf{divThree}(x)
\end{array}$$

Since $\mathcal{R}$ belongs to a class where innermost termination implies termination ([GA01, Theorem 16]) we choose $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$. As in Example 7.6 all pairs but those for the f-rules can easily be removed. In the problematic DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}', \mathbf{m})$ the TRS $\mathcal{R}'$ consists of all but the f-rules and we have two pairs:

$$\mathsf{F}(\mathsf{false}, x) \to \mathsf{F}(\mathsf{even}(\mathsf{six}(x)), x) \tag{177}$$

$$\mathsf{F}(\mathsf{false}, x) \to \mathsf{F}(\mathsf{divThree}(\mathsf{six}(x)), x) \tag{178}$$

To solve this DP problem, one can first use semantic labeling with a modulo-2-counter to get rid of (177), and afterwards use a modulo-3-counter for (178). Of course, in this example one can directly use a modulo-6-counter but then the search space for suitable interpretations is rather large.

So, we start with the algebra $\mathcal{M}$ with carrier $M = \{0, 1\}$ and the interpretation given by $\mathsf{s}_{\mathcal{M}}(x) = (x + 1)\ \mathit{mod}\ 2$, $\mathsf{false}_{\mathcal{M}} = \mathsf{divThree}_{\mathcal{M}}(x) = 1$, $\mathsf{even}_{\mathcal{M}}(x) = x$, and $f_{\mathcal{M}}(\dots) = 0$ for all remaining function symbols f.

If we use full labeling we obtain the following new pairs $\overline{\mathcal{P}}$ and rules $\overline{\mathcal{R}'}$.

$$\begin{array}{rcl}
\mathsf{F}_{1,0}(\mathsf{false}, x) & \to & \mathsf{F}_{0,0}(\mathsf{even}_0(\mathsf{six}_0(x)), x) \\
\mathsf{F}_{1,1}(\mathsf{false}, x) & \to & \mathsf{F}_{0,1}(\mathsf{even}_0(\mathsf{six}_1(x)), x) \\
\mathsf{F}_{1,0}(\mathsf{false}, x) & \to & \mathsf{F}_{1,0}(\mathsf{divThree}_0(\mathsf{six}_0(x)), x) \\
\mathsf{F}_{1,1}(\mathsf{false}, x) & \to & \mathsf{F}_{1,1}(\mathsf{divThree}_0(\mathsf{six}_1(x)), x)
\end{array}$$

$$\begin{array}{rcl}
\mathsf{six}_0(\mathsf{0}) & \to & \mathsf{0} \\
\mathsf{six}_1(\mathsf{s}_0(x)) & \to & \mathsf{s}_1(\mathsf{s}_0(\mathsf{s}_1(\mathsf{s}_0(\mathsf{s}_1(\mathsf{s}_0(\mathsf{six}_0(x))))))) \\
\mathsf{six}_0(\mathsf{s}_1(x)) & \to & \mathsf{s}_1(\mathsf{s}_0(\mathsf{s}_1(\mathsf{s}_0(\mathsf{s}_1(\mathsf{s}_0(\mathsf{six}_1(x))))))) \\
\mathsf{even}_1(\mathsf{s}_0(\mathsf{0})) & \to & \mathsf{false} \\
\mathsf{even}_0(\mathsf{s}_1(\mathsf{s}_0(x))) & \to & \mathsf{even}_0(x) \\
\mathsf{even}_1(\mathsf{s}_0(\mathsf{s}_1(x))) & \to & \mathsf{even}_1(x) \\
\mathsf{divThree}_1(\mathsf{s}_0(\mathsf{0})) & \to & \mathsf{false} \\
\mathsf{divThree}_1(\mathsf{s}_0(\mathsf{s}_1(\mathsf{s}_0(x)))) & \to & \mathsf{divThree}_0(x) \\
\mathsf{divThree}_0(\mathsf{s}_1(\mathsf{s}_0(\mathsf{s}_1(x)))) & \to & \mathsf{divThree}_1(x)
\end{array}$$

As in Example 7.6 we can easily remove the first two labeled pairs by the processors based on the dependency graph, and moreover we can delete all labeled even-rules, since they are no longer usable. But we still have a problem with the remaining two labeled pairs. Of course, it is now possible to use a semantic labeling processor with a modulo-3-counter on the remaining labeled DP problem, but this results a huge search problem as now one has to find interpretations for twice as many symbols. In essence, then we could have directly searched for a model with carrier size six for the modulo-6-counter. To conclude, repeated application of a semantic labeling processor results in huge DP problems and huge search problems.

An alternative approach is to remove the labels again after we have simplified the labeled DP problem.[34] Here this would result in the DP problem $(\{(178)\}, \mathcal{Q}, \mathcal{R}'', \mathbf{m})$ where $\mathcal{R}''$ is like $\mathcal{R}'$ without the even-rules. Then we do not have the problem that the DP problems grow larger and larger, but we always result in DP problems that are smaller than the original DP problem. Additionally, the search-space for new models is not increased, too.

---

[34] Note that a corresponding idea was independently developed in [Zan05b, Section 7.3] which removes labels on the level of string rewrite systems.

So, in our example we can again apply semantic labeling, but this time we use the algebra $\mathcal{M}'$ with the carrier $M' = \{0, 1, 2\}$ and the interpretation given by $\mathsf{s}_{\mathcal{M}'}(x) = (x + 1)\ mod\ 3$, $\mathsf{false}_{\mathcal{M}'} = 1$, $\mathsf{divThree}_{\mathcal{M}'}(x) = x$, and $f_{\mathcal{M}'}(\dots) = 0$ for all remaining function symbols f. For full labeling we obtain the following new pairs.

$$\begin{array}{rcl}
\mathsf{F}_{1,0}(\mathsf{false}, x) & \rightarrow & \mathsf{F}_{0,0}(\mathsf{divThree}_0(\mathsf{six}_0(x)), x) \\
\mathsf{F}_{1,1}(\mathsf{false}, x) & \rightarrow & \mathsf{F}_{0,1}(\mathsf{divThree}_0(\mathsf{six}_1(x)), x) \\
\mathsf{F}_{1,2}(\mathsf{false}, x) & \rightarrow & \mathsf{F}_{0,2}(\mathsf{divThree}_0(\mathsf{six}_2(x)), x)
\end{array}$$

Now obviously the resulting DP problem can be solved by the processors based on the dependency graph.

Formulating the alternative approach as one combined processor would consist of three steps.

(i) Given a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ find a model $\mathcal{M}$ of $\mathcal{R}$ and choose some labeling.

(ii) Use processors to simplify the labeled DP problem $(\overline{\mathcal{P}}, \overline{\mathcal{Q}}, \overline{\mathcal{R}}, f)$[35] and obtain new DP problems $\mathcal{D}_1, \dots, \mathcal{D}_n$.

(iii) Return $\{\mathit{Unlab}(\mathcal{D}_1), \dots, \mathit{Unlab}(\mathcal{D}_n)\}$.

In this combined processor there is one major problem. Note that a processor may transform DP problems arbitrarily, i.e., it may introduce new function symbols which do not possess a label. (Examples are given by the needed rules processor of Theorem 4.12 and the $\mathcal{A}$-transformation processor of Theorem 6.8.) Moreover, a processor may attach additional labels like the semantic labeling processor. So, in general it is completely unclear how to remove the labels of the DP problems $\mathcal{D}_1, \dots, \mathcal{D}_n$.

To this end, we must not allow arbitrary processors in step (ii). A first requirement is quite natural. We do not allow processors which introduce new function symbols. But even if we require the harder condition that the processors in step (ii) may only reduce $\mathcal{P}$ and $\mathcal{R}$, and that they may not modify $\mathcal{Q}$, then still the alternative approach is unsound. This is demonstrated in the following example.

Example 7.28. We adapt the TRS of Example 7.11.

$$\begin{array}{rcl}
\mathsf{f}(x, y) & \rightarrow & \mathsf{f}(x, y) \\
\mathsf{f}(\mathsf{a}, x) & \rightarrow & \mathsf{f}(\mathsf{g}(x), x) \\
\mathsf{g}(\mathsf{h}(x)) & \rightarrow & \mathsf{i}(x) \\
\mathsf{i}(\mathsf{a}) & \rightarrow & \mathsf{a} \\
\mathsf{h}(\mathsf{a}) & \rightarrow & \mathsf{a}
\end{array}$$

Obviously, $\mathcal{R}$ is not innermost-terminating due to the first rule. We can easily simplify the initial DP problem to $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ with the following pairs in $\mathcal{P}$.

$$\begin{array}{rcl}
\mathsf{F}(x, y) & \rightarrow & \mathsf{F}(x, y) \\
\mathsf{F}(\mathsf{a}, x) & \rightarrow & \mathsf{F}(\mathsf{g}(x), x)
\end{array}$$

---

[35] Of course, the labeled DP problem depends on the theorem that is used for the labeling, e.g., we obtain different problems when we use Theorem 7.10, Theorem 7.15, or Theorem 7.23.

If we use the same $\mathcal{F}$-algebra, labeled signature, and labeling map as in Examples 7.7 and 7.11, we obtain the following labeled rules $\overline{\mathcal{R}}$, the labeled version of $\mathcal{Q}$ is $\overline{\mathcal{Q}} = \mathit{lhs}(\overline{\mathcal{R}})$, and $\overline{\mathcal{P}} = \mathcal{P}$.

$$\begin{array}{rcl} \mathsf{f}(x,y) & \to & \mathsf{f}(x,y) \\ \mathsf{f}(\mathsf{a},x) & \to & \mathsf{f}(\mathsf{g}(x),x) \\ \mathsf{g}(\mathsf{h}_0(x)) & \to & \mathsf{i}(x) \\ \mathsf{g}(\mathsf{h}_1(x)) & \to & \mathsf{i}(x) \\ \mathsf{i}(\mathsf{a}) & \to & \mathsf{a} \\ \mathsf{h}_0(\mathsf{a}) & \to & \mathsf{a} \end{array}$$

Now we simplify the labeled DP problem $(\overline{\mathcal{P}}, \overline{\mathcal{Q}}, \overline{\mathcal{R}}, f)$ by the sound processor which removes the first pair from $\overline{\mathcal{P}}$. Note that this really is sound according to the semantics of a processor. It is only required that whenever there is an infinite (minimal) chain in the input DP problem then there must be *some* infinite (minimal) chain in the output problem. As the resulting DP problem has the infinite minimal chain

$$\mathsf{F}(\mathsf{a},\mathsf{h}_1(\mathsf{a})) \to_{\overline{\mathcal{P}}} \mathsf{F}(\mathsf{g}(\mathsf{h}_1(\mathsf{a})),\mathsf{h}_1(\mathsf{a})) \xrightarrow{\overline{\mathcal{Q}}}_{\overline{\mathcal{R}}} \mathsf{F}(\mathsf{i}(\mathsf{a})),\mathsf{h}_1(\mathsf{a})) \xrightarrow{\overline{\mathcal{Q}}}_{\overline{\mathcal{R}}} \mathsf{F}(\mathsf{a},\mathsf{h}_1(\mathsf{a})) \to_{\overline{\mathcal{P}}} \dots$$

the processor obviously satisfies this requirement.

Unlabeling the DP problem $(\{\mathsf{F}(\mathsf{a},x) \to \mathsf{F}(\mathsf{g}(x),x)\}, \overline{\mathcal{Q}}, \overline{\mathcal{R}}, \mathbf{a})$ results in the DP problem $(\{\mathsf{F}(\mathsf{a},x) \to \mathsf{F}(\mathsf{g}(x),x)\}, \mathcal{Q}, \mathcal{R}, \mathbf{a})$ which is finite, cf. Example 7.11.

So we need an even stronger correspondence between the input DP problem and the output DP problems of the processor which is used in step (ii). The main idea is that we use processors with $\mathit{Proc}((\mathcal{P},\mathcal{Q},\mathcal{R},f)) = \{(\mathcal{P}_1,\mathcal{Q},\mathcal{R}_1,f_1),\dots,(\mathcal{P}_k,\mathcal{Q},\mathcal{R}_k,f_k)\}$ which transform every infinite $(\mathcal{P},\mathcal{Q},\mathcal{R})$-chain $s_1 \to t_1, s_2 \to t_2, \dots$ into a *corresponding* new $(\mathcal{P}_i,\mathcal{Q},\mathcal{R}_i)$-chain by just omitting some initial part of the chain. Here, the substitution and the rewrite steps that are used to build the $(\mathcal{P},\mathcal{Q},\mathcal{R})$-chain may not be changed. Thus, these processor can only *identify* which pairs and rules cannot occur infinitely often in an infinite chain.

For example, the processor used for the simplification in Example 7.28 does not have this property: it is possible to build an infinite chain using only the pair $\mathsf{F}(x,y) \to \mathsf{F}(x,y)$ and there is no possibility to build a corresponding chain with infinitely many occurrences of $\mathsf{F}(x,y) \to \mathsf{F}(x,y)$ from the resulting DP problem.

Definition 7.29 (Chain Identifying Processor). *A processor Proc is* chain identifying *iff for every DP problem $(\mathcal{P},\mathcal{Q},\mathcal{R},f)$ the processor returns a set of new DP problems $\{(\mathcal{P}_1,\mathcal{Q},\mathcal{R}_1,f),\dots,(\mathcal{P}_k,\mathcal{Q},\mathcal{R}_k,f)\}$ with $\mathcal{P}_i \subseteq \mathcal{P}$ and $\mathcal{R}_i \subseteq \mathcal{R}$ such that for every infinite $(\mathcal{P},\mathcal{Q},\mathcal{R},f)$-chain $s_1 \to t_1, s_2 \to t_2, \dots$ using the substitution σ with*

$$t_i\sigma \xrightarrow{\mathcal{Q}}_{\ell_{i,1}\to r_{i,1}} \dots \xrightarrow{\mathcal{Q}}_{\ell_{i,j_i}\to r_{i,j_i}} s_{i+1}\sigma$$

there is an $n \in \mathbb{N}$ and $m \in \{1,\dots,k\}$ such that $s_n \to t_n, s_{n+1} \to t_{n+1}, \dots$ is an infinite path in $\mathcal{P}_m$ and all rules $\ell_{i,j} \to r_{i,j}$ with $i \geq n$ are in $\mathcal{R}_m$.

First note that chain identifying processors are always sound and complete, but not every sound and complete processor is chain identifying. However, many processors are indeed chain identifying. (Theorems 3.3, 3.4, 3.25, 4.2, 4.18, 4.20, 4.22, 4.27, 4.32, 4.39, 4.41, 6.17 (B) and (D), 6.22, and 7.30) Further note that it is not a hard requirement that a chain identifying processor may neither change $\mathcal{Q}$ nor the minimality flag. The reason is that whenever it is identified that every infinite chain from some point onwards only uses certain pairs and rules then of course, if the original chain is minimal and respects the strategy, then this will also hold for the shortened chain which is like the original chain just without some initial part.

Now we present the semantic labeling processor that is used in Example 7.27. It first labels the DP problem, then it simplifies the labeled DP problem by some chain identifying processor and finally it removes all labels.

Theorem 7.30 (Semantic Labeling and Unlabeling Processors)**.** *Let $\mathit{Proc}_{\mathit{id}}$ be a chain identifying processor, let $\mathit{Proc}_{\mathit{model}}$ be a processor of Theorem 7.5, Theorem 7.10, or of Theorem 7.15, let $\mathit{Proc}_{\mathit{quasi}}$ be a processor of Theorem 7.19 or of Theorem 7.23. Then the following processor Proc is chain identifying and therefore sound and complete. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, Proc returns*

- $\{(\mathit{Unlab}(\mathcal{P}_1), \mathcal{Q}, \mathit{Unlab}(\mathcal{R}_1), f), \dots, (\mathit{Unlab}(\mathcal{P}_k), \mathcal{Q}, \mathit{Unlab}(\mathcal{R}_k), f)\}$,
  if $\mathcal{M}$ is a model of $\mathcal{R}$,
  $\mathit{Proc}_{\mathit{model}}((\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)) = \{(\mathcal{P}', \mathcal{Q}', \mathcal{R}', f')\}$, *and*
  $\mathit{Proc}_{\mathit{id}}((\mathcal{P}', \mathcal{Q}', \mathcal{R}', f')) = \{(\mathcal{P}_1, \mathcal{Q}', \mathcal{R}_1, f'), \dots, (\mathcal{P}_k, \mathcal{Q}', \mathcal{R}_k, f')\}$

- $\{(\mathit{Unlab}(\mathcal{P}_1), \mathcal{Q}, \mathit{Unlab}(\mathcal{R}_1 \setminus \mathcal{D}\mathit{ecr}), f), \dots,$
  $(\mathit{Unlab}(\mathcal{P}_k), \mathcal{Q}, \mathit{Unlab}(\mathcal{R}_k \setminus \mathcal{D}\mathit{ecr}), f)\}$,
  if $\mathcal{M}$ is a quasi-model of $\mathcal{R}$,
  $\mathit{Proc}_{\mathit{quasi}}((\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)) = \{(\mathcal{P}', \mathcal{Q}', \mathcal{R}', f')\}$, *and*
  $\mathit{Proc}_{\mathit{id}}((\mathcal{P}', \mathcal{Q}', \mathcal{R}', f')) = \{(\mathcal{P}_1, \mathcal{Q}', \mathcal{R}_1, f'), \dots, (\mathcal{P}_k, \mathcal{Q}', \mathcal{R}_k, f')\}$

- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$, *otherwise*

Beside the previously mentioned advantage of Theorem 7.30 that we remain with small DP problems there is another benefit. Note that as a chain identifying processor, the processor of Theorem 7.30 is always complete and it preserves the minimality flag. This is in contrast to the previous processors based on semantic labeling which are used as ingredient to this new processor. There, only under certain conditions completeness is obtained and minimality can be preserved.

There seems to be a slight restriction in Theorem 7.30. Up to now we only allow *one* application of *one* chain identifying processor between labeling and unlabeling. However, this is not a severe restriction as we can always compress an *arbitrary* application of *arbitrary* chain identifying processor into one chain identifying processor using the following lemma. This was already needed in Example 7.27, since there we applied three chain identifying processors on the labeled DP problem before unlabeled it.

Lemma 7.31 (Combining Chain Identifying Processors)**.** *Let Proc_0, Proc_1, …, Proc_n be chain identifying processors, and let $\mathcal{D}$ be a DP problem. If $\mathit{Proc}_0(\mathcal{D}) = \{\mathcal{D}_1, \dots, \mathcal{D}_n\}$ and if for all $1 \leq i \leq n$ the result of $\mathit{Proc}_i(\mathcal{D}_i)$ is $\{\mathcal{D}_{i,1}, \dots, \mathcal{D}_{i,k_i}\}$ then the following processor Proc is chain identifying. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, Proc returns*

- $\{\mathcal{D}_{1,1}, \dots, \mathcal{D}_{1,k_1}, \dots, \mathcal{D}_{n,1}, \dots, \mathcal{D}_{n,k_n}\}$, *if* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f) = \mathcal{D}$

- $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, *otherwise*

By Lemma 7.31 we know that applying the processors of Theorems 3.3 and 3.4 in sequence results in a chain identifying processor *Proc*. And applying Lemma 7.31 a second time, we conclude that the sequential application of *Proc* and Theorem 3.25 also is a chain identifying processor $Proc'$. Hence, the sequential application of all three processors indeed corresponds to one application of the (combined) chain identifying processor $Proc'$. This finally allows us to apply Theorem 7.30 like we did in Example 7.27.

Summary of Chapter 7

In this chapter we have shown various unpublished new results about how to extend semantic labeling, a technique usually applied on TRSs, to the DP framework. To this end, we have lifted semantic labeling from full rewriting to $\mathcal{Q}$-restricted rewriting for both models and quasi-models, and we have presented ways to preserve minimality and to achieve completeness for a semantic labeling processor. Finally, if one wants to label, simplify, and then unlabel a DP problem, we have figured out the class of chain-identifying processors which may be used for the intermediate simplifying step. In this way, one can always achieve completeness and minimality is never lost. This final approach has the additional advantage that it never increases the resulting DP problem. Therefore, we propose this approach for the automation of semantic labeling in combination with the chain identifying processors of Chapters 3 and 4.

For the automation of course one has to find suitable (quasi-)models. How to solve this problem is investigated in [KM07, KZ06, Zan05b]. In [Zan05b] one can also find our idea of labeling and unlabeling, which was independently developed. However, there are two major differences: in [Zan05b] labeling and unlabeling is performed on the level of string rewrite systems and only one specific technique (rule removal as in Section 4.3) is applied between labeling and unlabeling, whereas we have presented the results in the DP framework and we have identified a whole class of processors that may be used in between.

The combination of semantic labeling and dependency pairs has already been done in [Ohl01], but this work is completely subsumed by our work, as one can use Theorem 7.30 with the basic reduction pair processor of Theorem 4.2 to simulate [Ohl01], for example. However, in contrast to [Ohl01] we investigate minimality and completeness, we have a more complex structure of DP problems which contain a strategy and a graph-component, and we can combine every chain identifying processor with semantic labeling, and not only the basic reduction pair processor.

Semantic labeling under strategies has already been investigated by us in [TM07] where we considered innermost termination for semantic labeling with models. This thesis completely extends these results as we deal with the more general case of $\mathcal{Q}$-restricted rewriting. Moreover, we even perform semantic labeling with evaluation strategy and quasi-models. This is again an improvement, since in [TM07] it is shown that one cannot use semantic labeling with quasi-models if one considers innermost rewriting. And finally, in [TM07] we only worked on the level of TRSs, whereas in this thesis we integrated semantic labeling to the DP framework.

However, in the work of [HM06a, TM07] a variant of semantic labeling, *predictive labeling*, is presented. The advantage of predictive labeling is that only a (quasi-)model of the usable rules (which are defined w.r.t. a given labeling) has to be found. But since in these papers predictive labeling again works on TRSs, and not on DP problems it is incomparable to our work. Moreover, only ordinary and innermost rewriting have been considered.

The work of [KM07] integrates predictive labeling to the DP framework for full rewriting, it corresponds to the processor of Theorem 7.19 for $\mathcal{Q} = \varnothing$ where only the needed rules have to satisfy the quasi-model condition. This clearly extends our results in some way, but certain aspects like completeness and minimality are not covered in that work. Therefore, it would be an interesting future work to integrate that work into ours, but there may be problems. For example for non-linear sets $\mathcal{Q}$, here we used full labeling to preserve minimality. But for full labeling semantic labeling and predictive labeling coincide.

We show by an adaptation of Example 7.6 that certain benefits of predictive labeling are already present in our processors.

Example 7.32. Let $\mathcal{R}$ consist of the following rules.

$$\begin{array}{rcl}
\mathsf{f}(\mathsf{true},x) & \to & \mathsf{random}(\mathsf{f}(\mathsf{odd}(\mathsf{double}(x)),x)) \\
\mathsf{double}(\mathsf{0}) & \to & \mathsf{0} \\
\mathsf{double}(\mathsf{s}(x)) & \to & \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) \\
\mathsf{odd}(\mathsf{0}) & \to & \mathsf{false} \\
\mathsf{odd}(\mathsf{s}(\mathsf{0})) & \to & \mathsf{true} \\
\mathsf{odd}(\mathsf{s}(\mathsf{s}(x))) & \to & \mathsf{odd}(x) \\
\mathsf{random}(x) & \to & \mathsf{s}(x) \\
\mathsf{random}(x) & \to & x
\end{array}$$

Note that $\mathcal{R}$ is not terminating, but innermost terminating. To prove innermost termination as in Example 7.6, one would like to label the f-symbol. If one uses semantic labeling directly on $\mathcal{R}$ then one has problems finding a suitable model due to the random-rules. The benefit of *predictive* labeling is that one only has to find a model for the double- and odd-rules, but not for the problematic random-rules. However, performing the proof in the DP framework is also possible with *semantic* labeling. Using the standard processors we can remove all pairs in $DP(\mathcal{R})$ except for

$$\mathsf{F}(\mathsf{true},x) \;\to\; \mathsf{F}(\mathsf{odd}(\mathsf{double}(x)),x)$$

and we can replace $\mathcal{R}$ by the usable rules, which are the double- and odd-rules. Then, one can use the same algebra to prove finiteness of the DP problem as one would use to prove innermost termination of $\mathcal{R}$ by predictive labeling (e.g., one can use the algebra of Example 7.6): in both cases the same rules have to satisfy the model condition.

There is an improved variant of predictive labeling where one only has to consider the usable or needed rules w.r.t. an argument filter. With that technique one can even prove innermost termination of the TRS where one has replaced the first rule of $\mathcal{R}$ by the following.

$$\mathsf{f}(\mathsf{true},x,y) \;\to\; \mathsf{f}(\mathsf{odd}(\mathsf{double}(x)),x,\mathsf{random}(y))$$

Here, the random-rules are called inside the recursive call in an accumulator, which makes them usable. But if the interpretations and the labelings ignore the third argument of f then predictive labeling does not demand that the algebra is a model of the random-rules.

However, this can again be simulated by our methods. Applying the argument filter processor of Theorem 4.38 with $\pi(\mathsf{F}) = [1,2]$ deletes the accumulator of the corresponding dependency pair

$$\mathsf{F}(\mathsf{true}, x, y) \quad \to \quad \mathsf{F}(\mathsf{odd}(\mathsf{double}(x)), x, \mathsf{random}(y))$$

and afterwards the random-rules are not usable any more. Then one can prove finiteness as before.

Finally, to use predictive labeling in the termination case with quasi-models, there is a certain condition on the order. However, this condition ensures that if we apply the needed rules processor of Theorem 4.12, then the additional rules of $\mathcal{C}_\varepsilon$ always satisfy the quasi-model condition. Thus, again there is no large difference to predictive labeling, since often the needed rules are exactly the usable rules w.r.t. the labeling.

Nevertheless, a combination of our results with [KM07] would be helpful, since the processors of Theorems 4.38 and Theorem 4.12 are incomplete and the latter additionally destroys minimality.

Another interesting future work would be to answer the open question whether full labeling (Theorem 7.10) is complete.

8. Processors for Non-Termination Analysis

Almost all techniques for automated termination analysis try to *prove termination* and there are hardly any methods to *prove non-termination*. But detecting non-termination automatically would be very helpful when debugging programs.

We show that the DP framework is particularly suitable for combining both termination and *non*-termination analysis. We introduce a processor which tries to detect infinite DP problems in order to answer "`no`". Then, if all previous processors were complete, we can conclude non-termination of the original TRS. An important advantage of the DP framework is that it can couple the search for a proof and a disproof of termination:

Processors which try to prove termination are also helpful for the non-termination proof because they transform the initial DP problem into sub-problems, where most of them can easily be proved finite. So they detect those sub-problems which could cause non-termination. Therefore, the non-termination processors should only operate on these sub-problems and thus, they only have to regard a subset of the rules when searching for non-termination.

On the other hand, processors that try to disprove termination are also helpful for the termination proof, even if some of the previous processors were incomplete. The reason is that there are many indeterminisms in a termination proof attempt, since usually many processors can be applied to a DP problem. Thus, if one can find out that a DP problem is infinite, very often one has reached a "dead end" and should backtrack if one has applied incomplete processors before.[36]

Our criteria to detect infiniteness of a DP problem are based on looping DP problems which were already introduced in [GTS05b]. However, in that work it has not been investigated how to detect loops in the context of innermost or $\mathcal{Q}$-restricted rewriting. Since these problems often occur, there is urgent need to check whether a loop (for full rewriting) also respects the strategy. To this end, in [GTS05c] sufficient, but incomplete criteria have been presented. However, in this thesis we strictly extend the results of [GTS05c] as we will show and explain a novel decision procedure for this question. Although the procedure is easy to implement its termination proof is quite involved.

This chapter is organized as follows. To prove non-termination within the DP framework, in Section 8.1 we recall the notion of a looping problem and generalize it from full rewriting to $\mathcal{Q}$-restricted rewriting. Then in Section 8.2 we show a new processor which can switch from innermost termination to termination which will make disproving $\mathcal{Q}$-termination easier. That processor will even allow us to disprove innermost termination of non-innermost looping problems by finding a loop which disregards the innermost strategy. Finally, in Section 8.3 we will present the new decision procedure for detecting loops for $\mathcal{Q}$-restricted rewriting.

---

[36] It might still be possible to prove finiteness, since there are examples like Example 2.12 which are both finite and infinite.

8.1. Looping Problems

An obvious approach to find infinite reductions is to search for a term s which reduces to a term $C[s\mu]$ containing an instance of s. Such a reduction is called a *loop* and a TRS $\mathcal{R}$ with such a reduction is called *looping*. A loop has three desirable properties. First, each loop can be finitely represented, e.g., by giving the rules and corresponding positions that are used in the reduction from s to $C[s\mu]$. Second, once one has found the rules and positions for the rewrite sequence, it is easily possible to verify whether it is really possible to reduce s to $C[s\mu]$. (For full rewriting this is trivial.) And most importantly, a loop gives rise to the following infinite reduction which proves non-termination.

$$s \to_{\mathcal{R}}^{+} C[s\mu] \to_{\mathcal{R}}^{+} C[C\mu[s\mu^2]] \to_{\mathcal{R}}^{+} C[C\mu[C\mu^2[s\mu^3]]] \to_{\mathcal{R}}^{+} \dots \tag{179}$$

Unfortunately, if one does not consider full rewriting but innermost rewriting or $\mathcal{Q}$-restricted rewriting, then loopingness does not imply non-termination, since neither $\overset{\mathrm{i}}{\to}_{\mathcal{R}}$ nor $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ is stable if $\mathcal{Q} \neq \varnothing$. The reason is that from $s \overset{\mathcal{Q}}{\to}{}_{\mathcal{R}}^{+} C[s\mu]$ one cannot deduce $s\mu \overset{\mathcal{Q}}{\to}{}_{\mathcal{R}}^{+} C\mu[s\mu^2]$. And even if this is possible, then it might be the problem that later on for some larger n the reduction $s\mu^n \overset{\mathcal{Q}}{\to}{}_{\mathcal{R}}^{+} C\mu^n[s\mu^{n+1}]$ is not possible.

As an example consider the TRS $\mathcal{R} = \{\mathsf{f}(\mathsf{g}(x)) \to \mathsf{f}(\mathsf{g}(\mathsf{g}(x))), \mathsf{g}(\mathsf{g}(\mathsf{g}(x))) \to \mathsf{a}\}$. By choosing $s = \mathsf{f}(\mathsf{g}(x))$, $C = \Box$, and $\mu = \{x/\mathsf{g}(x)\}$ we obtain the following reduction.

$$s \overset{\mathrm{i}}{\to}_{\mathcal{R}} C[s\mu] = \mathsf{f}(\mathsf{g}(\mathsf{g}(x))) \overset{\mathrm{i}}{\to}_{\mathcal{R}} C[C\mu[s\mu^2]] = \mathsf{f}(\mathsf{g}(\mathsf{g}(\mathsf{g}(x))))$$

Now the only possible next reduction step yields the normal form $\mathsf{f}(\mathsf{a})$, such that one cannot build the infinite reduction (179) as in the termination case. And indeed, the TRS $\mathcal{R}$ is innermost terminating.

To solve this problem, one can define that a TRS $\mathcal{R}$ is *$\mathcal{Q}$-looping* iff there is a term s, a substitution μ, and a context C such that $s\mu^n \overset{\mathcal{Q}}{\to}{}_{\mathcal{R}}^{+} C\mu^n[s\mu^{n+1}]$ for every natural number n. A similar definition was already used in [GTS05b, Footnote 6]. Then indeed, $\mathcal{Q}$-loopingness implies non-termination for the $\mathcal{Q}$-restricted rewrite relation. However the following example shows that this definition does not really correspond to a loop, e.g., it does not have the first two of the three desired properties of a loop.

Example 8.1. Consider the TRS $\mathcal{R}$ with the following rules.

$$\begin{array}{rcl}
\mathsf{f}(x, y) & \to & \mathsf{f}(\mathsf{s}(x), \mathsf{g}(\mathsf{h}(x, \mathsf{0}))) \\
\mathsf{h}(\mathsf{s}(x), y) & \to & \mathsf{h}(x, \mathsf{s}(y)) \\
\mathsf{g}(\mathsf{h}(x, y)) & \to & \mathsf{i}(y)
\end{array}$$

Then for $\mathcal{Q} = \mathit{lhs}(\mathcal{R})$ the TRS $\mathcal{R}$ is $\mathcal{Q}$-looping, as for $s = \mathsf{f}(\mathsf{s}(x), \mathsf{g}(\mathsf{h}(x, \mathsf{0})))$, $C = \Box$, and $\mu = \{x/\mathsf{s}(x)\}$ there are the following reductions.

$$\begin{array}{rcl}
s\mu^n & = & \mathsf{f}(\mathsf{s}(x), \mathsf{g}(\mathsf{h}(x, \mathsf{0})))\mu^n \\
 & = & \mathsf{f}(\mathsf{s}^{n+1}(x), \mathsf{g}(\mathsf{h}(\mathsf{s}^n(x), \mathsf{0}))) \\
 & \overset{\mathcal{Q}}{\to}{}_{\mathcal{R}}^{n} & \mathsf{f}(\mathsf{s}^{n+1}(x), \mathsf{g}(\mathsf{h}(x, \mathsf{s}^n(\mathsf{0})))) \\
 & \overset{\mathcal{Q}}{\to}_{\mathcal{R}} & \mathsf{f}(\mathsf{s}^{n+1}(x), \mathsf{i}(\mathsf{s}^n(\mathsf{0}))) \\
 & \overset{\mathcal{Q}}{\to}_{\mathcal{R}} & \mathsf{f}(\mathsf{s}^{n+2}(x), \mathsf{g}(\mathsf{h}(\mathsf{s}^{n+1}(x), \mathsf{0}))) \\
 & = & C\mu^n[s\mu^{n+1}]
\end{array}$$

The problem is that the reductions from $s\mu^n$ to $C\mu^n[s\mu^{n+1}]$ depend on n. In this example it might still be possible to represent and check the reductions in a finite way as they have a regular structure, but in general the problem is not even semi-decidable.

Suppose we want to solve the undecidable problem whether a function over the naturals is total. Since term rewriting is Turing-complete we can assume that there are corresponding rules for i which compute that function by innermost evaluation strategy. But then we can add the three rules of $\mathcal{R}$ and totality of i is equivalent to the question whether s, μ, and C as above form an innermost-loop, since we obtain a loop iff all terms $\mathsf{i}(\mathsf{s}^n(\mathsf{0}))$ for $n \in \mathbb{N}$ have a normal form.

So, the problem with the requirement $s\mu^n \overset{\mathcal{Q}}{\rightarrow}{}^+_{\mathcal{R}} C\mu^n[s\mu^{n+1}]$ is that although there are some intermediate terms in the infinite reduction that have a regular structure, it is possible that for every n the reduction from $s\mu^n$ to $C\mu^n[s\mu^{n+1}]$ is completely different.

However, in the infinite reduction (179) for standard rewriting there is a strong regularity in the reductions from $s\mu^n$ to $C\mu^n[s\mu^{n+1}]$. For every n one can apply exactly the same rules in exactly the same order at exactly the same positions. Hence, one only has to give the reduction of s to $C[s\mu]$ to know how to continue for $s\mu, s\mu^2, \ldots$. This gives rise to the our final definition of $\mathcal{Q}$-looping.

Definition 8.2 ($\mathcal{Q}$-Looping TRS). *Let $\mathcal{R}$ be a TRS, and let $\mathcal{Q}$ be a set of terms. Then $\mathcal{R}$ is* $\mathcal{Q}$-looping *iff there is a term s, a context C, a substitution μ, a number $m \geq 1$, and if there are rules $\ell_1 \to r_1, \ldots, \ell_m \to r_m \in \mathcal{R}$ and positions $p_1, \ldots, p_m$ such that for all $n \in \mathbb{N}$ there is the following reduction.*

$$s\mu^n \overset{\mathcal{Q}}{\rightarrow}_{\ell_1 \to r_1, p_1} \circ \overset{\mathcal{Q}}{\rightarrow}_{\ell_2 \to r_2, p_2} \circ \cdots \circ \overset{\mathcal{Q}}{\rightarrow}_{\ell_m \to r_m, p_m} C[s\mu^{n+1}]$$

Note that Definition 8.2 generalizes the definition of a looping TRS, since a TRS $\mathcal{R}$ is looping iff $s \rightarrow^+_{\mathcal{R}} C[s\mu]$ iff $\mathcal{R}$ is $\varnothing$-looping. Moreover, this definition of $\mathcal{Q}$-loopingness has the desired first property that a loop can be finitely represented, and also the desired third property is satisfied.

Theorem 8.3 (Loops and Non-Termination). *Let $\mathcal{R}$ be a TRS and $\mathcal{Q}$ be a set of terms. If $\mathcal{R}$ is $\mathcal{Q}$-looping then $\mathcal{R}$ is not $\mathcal{Q}$-terminating.*

Theorem 8.3 is easily proven, since from $\mathcal{Q}$-loopingness of $\mathcal{R}$ we get the same reduction as in (179) where one just replaces $\rightarrow_{\mathcal{R}}$ by $\overset{\mathcal{Q}}{\rightarrow}_{\mathcal{R}}$. Notice that it is not at all obvious whether the second desired property of a loop is satisfied: there might be a problem to check whether a given rewrite sequence really is a loop, since one has to check the looping reduction for the possibly infinite set of terms $s, s\mu, \ldots$. Before we show the novel result in Section 8.3 how this problem can be decided, we first generalize loopingness to DP problems.

Our definition of a looping DP problem generalizes [GTS05b, Definition 22] from the termination case to $\mathcal{Q}$-restricted rewriting, in particular for $\mathcal{Q} = \varnothing$ both definitions are equivalent. However, our definition is different for the innermost case since we do not allow that the reductions may depend on n as it is done in [GTS05b, Footnote 6].

Definition 8.4 (Looping DP Problem). *Let $\mathcal{D} = (\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ be a DP problem. Then $\mathcal{D}$ is* looping *iff there is a term s, a substitution μ, a number $m \geq 1$, and if there are rules $\ell_1 \to r_1, \ldots, \ell_m \to r_m \in \mathcal{P} \cup \mathcal{R}$ and positions $p_1, \ldots, p_m$ such that for all $n \in \mathbb{N}$ there is the following reduction sequence.*

$$s\mu^n \overset{\mathcal{Q}}{\rightarrow}_{\ell_1 \to r_1, p_1} \circ \overset{\mathcal{Q}}{\rightarrow}_{\ell_2 \to r_2, p_2} \circ \cdots \circ \overset{\mathcal{Q}}{\rightarrow}_{\ell_m \to r_m, p_m} s\mu^{n+1}$$

Moreover, these reductions have to satisfy three additional requirements: there must be at least one $\mathcal{P}$-reduction, for every $\mathcal{P}$-reduction the corresponding position must be the root position, and for every $\mathcal{P}$-reduction the redex must be in $\mathcal{Q}$-normal form.[37]

The additional requirement on the $\mathcal{P}$-reductions corresponds to the fact that in chains, every pair in $\mathcal{P}$ may only be applied at the root and all instantiated left-hand sides of the pairs must be in $\mathcal{Q}$-normal form. Moreover, we need at least one pair to build a chain.

Theorem 8.5 (Processor Based on Loop-Detection)**.** *The following processor Proc is sound and complete. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, Proc returns*

- no*, if $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ is looping,*
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$*, otherwise*

Comparing Theorem 8.3 with Theorem 8.5, one can see two advantages of Theorem 8.5. First, there is no need to search for a context and instead of taking an arbitrary term s to start the loop of a TRS, one can always start the search with an instance of a left-hand side of a pair. The reason for the latter is that one can w.l.o.g. assume that the one necessary step with the pair is the first step in a loop. The proof of the following Theorem 8.6 reveals that one will find all looping TRSs, even if one starts the loops with an instance of a left-hand side of the TRS.

The second advantage is more important. Due to the previous processors usually there are only few rules and pairs remaining which may be non-terminating. Hence, the modularity of the DP framework helps to reduce the search space for a non-termination proof. We can apply every sound and complete processor – which often was designed to *prove* termination – before trying to *disprove* termination.

The following theorem shows that one can detect all $\mathcal{Q}$-looping TRSs by looking for loops in the corresponding DP problems.

Theorem 8.6 (Looping TRSs and Looping DP Problems)**.** *A TRS $\mathcal{R}$ is $\mathcal{Q}$-looping iff $(DP(\mathcal{R}), \mathcal{Q}, \mathcal{R}, f)$ is looping.*

Note that the correspondence between looping TRSs and looping DP problems was already proven in [GTS05b, Theorem 23], but only for full rewriting. Thus, Theorem 8.6 generalizes that result to $\mathcal{Q}$-restricted rewriting.

Example 8.7. Let $\mathcal{R} = \{\mathsf{f}(x,y,z,u) \to \mathsf{g}(x, \mathsf{f}(\mathsf{c}(x,y),z,z,y))\}$. For the initial DP problem $(\mathcal{P}, \varnothing, \mathcal{R}, \mathbf{m})$ with $\mathcal{P} = DP(\mathcal{R})$ there is the following reduction with the substitution $\mu = \{x/\mathsf{c}(x,y), y/z, u/y\}$.

$$s = \mathsf{F}(x,y,z,u) \to_{\mathcal{P},\varepsilon} \mathsf{F}(\mathsf{c}(x,y),z,z,y) = s\mu$$

Since full rewriting is closed under substitutions this directly implies $s\mu^n \to_{\mathcal{P},\varepsilon} s\mu^{n+1}$ for all n. Hence, $(\mathcal{P}, \varnothing, \mathcal{R}, \mathbf{m})$ is looping and infinite.

If we want to directly prove that $\mathcal{R}$ is a looping TRS, we can choose the same μ, a similar term $s = \mathsf{f}(x,y,z,u)$, but additionally we have to provide the context $C = \mathsf{g}(x, \Box)$.

In the next section we show that Theorem 8.6 only tells us half of the truth, and in Section 8.3 we investigate how to find loops.

---

[37] To improve readability in this chapter we disregard the graph structure of $\mathcal{P}$. However, all the presented results can be easily extended to graphs as well.

8.2. Switching to Termination

Usually, to analyze termination of a TRS we first apply various processors on the initial DP problem $(DP(\mathcal{R}), \mathcal{Q}, \mathcal{R}, \mathbf{m})$ before we try to disprove termination. Then, if all these processors have been complete we can conclude non-termination. But it may be the case that the initial DP problem is looping whereas the resulting DP problems are still infinite, but not looping any more. Therefore, we would not only like to know that our processors are complete, but they should also be *loop preserving*: using loop preserving processors has the advantage that they do not harm for disproving termination, whereas the application of other complete processors may prevent a successful non-termination proof.

And indeed, nearly all of our processors are loop preserving, e.g., every chain identifying processor (cf. Definition 7.29) is loop preserving. However, there is one processor that is often used and that is not loop preserving: the processor of Theorem 3.14 to switch to innermost termination. Consider the following example.

Example 8.8. Let $\mathcal{R}$ consist of the following rules and let $\mathcal{Q} = \varnothing$.

$$\begin{array}{rcl} \mathsf{f}(x) & \to & \mathsf{f}(\mathsf{g}(\mathsf{0}, \mathsf{true})) \\ \mathsf{g}(x, \mathsf{true}) & \to & \mathsf{g}(\mathsf{s}(x), \mathsf{isnat}(x)) \\ \mathsf{isnat}(\mathsf{0}) & \to & \mathsf{true} \\ \mathsf{isnat}(\mathsf{s}(x)) & \to & \mathsf{isnat}(x) \end{array}$$

Obviously, $\mathcal{R}$ is looping due to the first rule where one can choose $s = \mathsf{f}(x)$ and $\mu = \{x/\mathsf{g}(\mathsf{0}, \mathsf{true})\}$. However, the rules of the remaining TRS do not admit a looping reduction: The non-terminating reduction

$$\begin{array}{rcrl} \mathsf{g}(\mathsf{0}, \mathsf{true}) & \to_{\mathcal{R}} & \mathsf{g}(\mathsf{s}(\mathsf{0}), \mathsf{isnat}(\mathsf{0})) & \to_{\mathcal{R}}^{1} \\ \mathsf{g}(\mathsf{s}(\mathsf{0}), \mathsf{true}) & \to_{\mathcal{R}} & \mathsf{g}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{isnat}(\mathsf{s}(\mathsf{0}))) & \to_{\mathcal{R}}^{2} \\ \mathsf{g}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{true}) & \to_{\mathcal{R}} & \mathsf{g}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0}))), \mathsf{isnat}(\mathsf{s}(\mathsf{s}(\mathsf{0})))) & \to_{\mathcal{R}}^{3} \ \ldots \end{array}$$

is not a loop. Due to Theorem 8.6 also the initial DP problem contains a loop, using the dependency pair $\mathsf{F}(x) \to \mathsf{F}(\mathsf{g}(\mathsf{0}, \mathsf{true}))$ of the first rule. However, once we apply the usual processors including the processor to switch to innermost termination, we get the DP problem $(\mathcal{P}, \mathit{lhs}(\mathcal{R}), \mathcal{R}, \mathbf{m})$ with $\mathcal{P} = \{\mathsf{F}(x) \to \mathsf{F}(\mathsf{g}(\mathsf{0}, \mathsf{true})), \mathsf{G}(x, \mathsf{true}) \to \mathsf{G}(\mathsf{s}(x), \mathsf{isnat}(x))\}$. Then, there is no loop any more, since one cannot even build an infinite chain with the pair $\mathsf{F}(x) \to \mathsf{F}(\mathsf{g}(\mathsf{0}, \mathsf{true}))$ as $\mathsf{g}(\mathsf{0}, \mathsf{true})$ has no normal form, and since the other pair of $\mathcal{P}$ – as shown before – does not admit a loop.

To handle this problem, in this section we develop an inverse processor to Theorem 3.14 which switches to full termination. Then the problem of Example 8.8 is solved, as after its application we can again find the loop as for the initial DP problem. Moreover, by this method we can also prove that the TRS in Example 8.8 is not innermost terminating although it is not innermost looping.

Of course, when developing a processor which switches to full termination the difficult task is to prove that it is complete. But it turns out that completeness of such a processor is quite similar to soundness of the processor of Theorem 3.14. The reason is that in both ways we have to construct an infinite chain with large strategy component $\mathcal{Q}_{\mathit{large}}$ from an infinite chain with smaller strategy component $\mathcal{Q}_{\mathit{small}}$. However, there are minor differences: in Theorem 3.14 $\mathcal{Q}_{\mathit{small}} = \mathcal{Q}$ and $\mathcal{Q}_{\mathit{large}} = \mathit{lhs}(\mathcal{R})$ whereas for a processor to switch to full termination we have $\mathcal{Q}_{small} = \varnothing$ and $\mathcal{Q}_{large} = \mathcal{Q}$ where in both cases $\mathcal{Q}$ is the strategy component of the input DP problem. Another important difference is that for the completeness of a processor to switch to full termination one also needs to conclude $\mathcal{Q}$-restricted non-termination of $\mathcal{R}$ from non-termination of $\mathcal{R}$, i.e., one does not only have to consider infinite chains. One reason is that a DP problem $(\mathcal{P}, \varnothing, \mathcal{R}, f)$ is already infinite if $\mathcal{R}$ is not terminating. However, that this is not the only reason is illustrated in Example 8.10.

But let us first formally introduce the processor to switch to full termination in the following theorem, where we see that the requirements are quite similar to those of Theorem 3.14.

Theorem 8.9 (Processor to Switch to Termination). *Let Proc be the processor which returns* $\{(\mathcal{P}, \varnothing, \mathcal{R}, \mathbf{a})\}$ *for a given DP problem* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$*. Then, Proc is sound and it is complete, if*

- *for all* $s \to t \in \mathcal{P}$*, non-variable subterms of* s *do not unify with left-hand sides of rules from* $\mathcal{R}$*,*
- $\mathcal{R}$ *is locally confluent,*
- $\mathit{NF}(\mathcal{R}) \subseteq \mathit{NF}(\mathcal{Q})$*, and*
- $\mathcal{Q}$*-restricted termination of* $\mathcal{R}$ *implies termination of* $\mathcal{R}$

One question is how to guarantee the last requirement that $\mathcal{Q}$-restricted termination of $\mathcal{R}$ implies termination of $\mathcal{R}$ without doing the termination proof of $\mathcal{R}$. Our solution is to alternatively require that $\mathcal{R}$ belongs to a class of TRSs where innermost termination and termination are equivalent. Then the last requirement is indeed satisfied, since by Lemma 2.4 one can conclude innermost termination of $\mathcal{R}$ from $\mathcal{Q}$-termination of $\mathcal{R}$.

We now show that the last requirement of Theorem 8.9 is essential, even if one would define that a DP problem is infinite iff it admits an infinite chain.

Example 8.10. Consider the DP problem $(\varnothing, \mathit{lhs}(\mathcal{R}), \mathcal{R}, \mathbf{a})$ where $\mathcal{R}$ consists of the following rules.

$$\begin{aligned} \mathsf{g}(\mathsf{a}) &\to \mathsf{g}(\mathsf{b}) \\ \mathsf{g}(\mathsf{b}) &\to \mathsf{g}(\mathsf{a}) \\ \mathsf{a} &\to \mathsf{c} \\ \mathsf{b} &\to \mathsf{d} \end{aligned}$$

Then $(\varnothing, \mathit{lhs}(\mathcal{R}), \mathcal{R}, \mathbf{a})$ is not infinite as $\mathcal{R}$ is innermost terminating and as there obviously is no infinite innermost chain. However, $\mathcal{R}$ is locally confluent, $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$ is even confluent, and $(\varnothing, \varnothing, \mathcal{R}, \mathbf{a})$ is infinite since $\mathcal{R}$ is not terminating. Thus, without the requirement that innermost termination of $\mathcal{R}$ should imply termination of $\mathcal{R}$, Theorem 8.9 becomes unsound.

But even if one considers the DP problems $(\mathcal{P}, \mathit{lhs}(\mathcal{R}), \mathcal{R}, \mathbf{a})$ and $(\mathcal{P}, \varnothing, \mathcal{R}, \mathbf{a})$ with $\mathcal{P} = \{\mathsf{F}(\mathsf{g}(\mathsf{c}), \mathsf{g}(\mathsf{d}), x) \to \mathsf{F}(x, x, x)\}$ one obtains a counterexample. Although there is an infinite $(\mathcal{P}, \varnothing, \mathcal{R})$-chain in addition to non-termination of $\mathcal{R}$ there still is no infinite $(\mathcal{P}, \mathit{lhs}(\mathcal{R}), \mathcal{R})$-chain.

Another way to reduce the set $\mathcal{Q}$ is to apply the $\mathcal{Q}$-reduction processors of Theorems 3.34 and 3.36. Since for disproving termination minimality is irrelevant, one should always prefer Theorem 3.34 over Theorem 3.36 as the processor of the former theorem can delete more terms from $\mathcal{Q}$ and it is more often applicable.

Since Theorem 3.34 and Theorem 8.9 are incomparable one should apply them both as a first step to disprove termination. Nevertheless, these processors just simplify the DP problem for a later non-termination proof, but they will never say “no”. Hence, to disprove termination we afterwards still have to apply Theorem 8.5. However, we still have not presented a way to detect loops. This is the topic of the next section.

8.3. Detecting Looping Problems

In [GTS05b] it is investigated how loops can be detected by narrowing. Another approach to detect loops is based on unfolding and is presented in [Pay06]. Moreover, for the special class of string rewrite systems, in [GZ99] a decision procedure is presented, which can answer the question whether there is a loop up to a given length. However, in all these papers ([GTS05b, GZ99, Pay06]) only loops for full rewriting are detected and there is no technique to detect whether such a loop also respects the strategy given by $\mathcal{Q}$. This is partially solved in [GTS05c] where *approximations* are presented by which one can detect innermost loops.

In this section we extend the results of [GTS05c] and develop a novel method to *decide* whether a loop respects the strategy. If there is a loop for full rewriting then we have found a reduction of the following form.

$$s\mu^n \to_{\ell_1\to r_1,p_1} \circ \to_{\ell_2\to r_2,p_2} \circ \cdots \circ \to_{\ell_m\to r_m,p_m} s\mu^{n+1}$$

Since every time the same rules are applied at the same positions, all intermediate terms have the form $s_i\mu^n$. Hence, the reductions look at follows.

$$s\mu^n = s_1\mu^n \to_{\ell_1\to r_1,p_1} s_2\mu^n \to_{\ell_2\to r_2,p_2} \dots s_m\mu^n \to_{\ell_m\to r_m,p_m} s\mu^{n+1}$$

Note that this reduction is $\mathcal{Q}$-looping iff every direct subterm $s_i|_{p_ij}\mu^n$ of every redex $s_i\mu^n|_{p_i}$ is in $\mathcal{Q}$-normal form (and if one considers looping DP problems then additionally for every i with $\ell_i \to r_i \in \mathcal{P}$ the terms $s_i\mu^n$ have to be in $\mathcal{Q}$-normal form). Since a term t is in $\mathcal{Q}$-normal form iff t does not contain a redex w.r.t. $\mathcal{Q}$, we can reformulate the question about $\mathcal{Q}$-loopingness in terms of so-called *redex problems*.

Definition 8.11 (Redex, Matching, and Identity Problems)**.** *Let s and q be terms, let μ be a substitution with finite domain. Then a* redex problem *is a triple $(s \mathrel{|}\!\gtrdot q, \mu)$, a* matching problem *is a triple $(s \gtrdot q, \mu)$, and an* identity problem *is a triple $(s \approxeq q, \mu)$.*

A redex problem $(s \mathrel{|}\!\gtrdot q, \mu)$ is solvable *iff there is a natural number n, a position p, and a substitution σ such that $s\mu^n|_p = q\sigma$, a matching problem is* solvable *iff there is a natural number n and a substitution σ such that $s\mu^n = q\sigma$, and an identity problem is* solvable *iff there is a natural number n such that $s\mu^n = q\mu^n$.*

Obviously, for every term s the redex problem $(s \mathrel{|}\!\gtrdot \mu, q)$ is not solvable for any $q \in \mathcal{Q}$ iff $s\mu^n \in NF(\mathcal{Q})$ for all $n \in \mathbb{N}$.

Example 8.12. We consider the TRS of Example 8.7 where now we want to analyze $\mathcal{Q}$-termination for $\mathcal{Q} = \{q\}$ with $q = \mathsf{c}(\mathsf{c}(\mathsf{c}(x_1, x_1), x_2), x_3)$. Then the loop of Example 8.7 (with $s = \mathsf{F}(x, y, z, u)$ and $\mu = \{x/\mathsf{c}(x, y), y/z, u/y\}$) respects the strategy iff $(s \mathrel{|}\!\gtrdot q, \mu)$ is not solvable.

To answer the question whether a redex problem $(s \mid\!> q, \mu)$ is solvable, we have to look for three unknowns: the position p, the substitution σ, and the number n. We will now eliminate these unknowns one by one and start with the position p. This will result in matching problems. Then in a second step we will further transform matching problems into identity problems where only the number n is unknown. Finally, we will present a novel algorithm to decide identity problems. Therefore, at the end of this section we will have a decision procedure for redex problems, and thus also for the question whether a given loop respects the strategy.

To start with simplifying a redex problem $(s \mid\!> q, \mu)$ into a finite disjunction of matching problems, note that since the position can be chosen freely from all terms $s, s\mu, s\mu^2, \ldots$ it is not possible to just unroll the possible choices for the position. But the following theorem shows that it is indeed possible to reduce redex problems to matching problems.

Theorem 8.13 (Solving Redex Problems). *Let* $(s \mid\!> q, \mu)$ *be a redex problem. Let* $\mathcal{W} = \bigcup_{i \in \mathbb{N}} \mathcal{V}(s\mu^i)$. *Then* $(s \mid\!> q, \mu)$ *is solvable iff* q *is a variable or the matching problem* $(u > q, \mu)$ *is solvable for some non-variable subterm* u *of a term in* $\{s\} \cup \{x\mu \mid x \in \mathcal{W}\}$.

Note that the set $\mathcal{W}$ is finite since it is a subset of the finite set of variables $\mathcal{V}(s) \cup \bigcup_{x \in \mathit{Dom}(\mu)} \mathcal{V}(x\mu)$. Hence, Theorem 8.13 can easily be automated.

Example 8.14. We use Theorem 8.13 for the redex problem $(s \mid\!> q, \mu)$ of Example 8.12.

i	$s\mu^i$	$\mathcal{V}(s\mu^i)$
0	$\mathsf{F}(x, y, z, u)$	$\{x, y, z, u\}$
1	$\mathsf{F}(\mathsf{c}(x, y), z, z, y)$	$\{x, y, z\}$

Since there is no new variable in the second iteration we can stop and know that $\mathcal{W} = \{x, y, z, u\}$. Thus, $(s \mid\!> q, \mu)$ is solvable iff one of the following two matching problems is solvable.

$$(\mathsf{F}(\mathsf{c}(x, y), z, z, y) > q, \mu) \qquad (\mathsf{c}(x, y) > q, \mu)$$

Now the question whether a given matching problem is solvable remains. This amounts to detecting the unknown number n and the matcher σ. Our next aim is again to reduce this problem to a conjunction of identity problems where there is no matcher σ any more. However, we first have to generalize the notion of a matching problem $(s > q, \mu)$ which includes one pair of terms $s > q$ into a matching problem which allows a set of pairs of terms.

Definition 8.15 (Matching Problem). *A* matching problem *is a pair* $(\mathcal{M}, \mu)$ *of a set* $\mathcal{M}$ *of pairs* $\{s_1 > q_1, \ldots, s_k > q_k\}$ *together with a substitution* μ. *A matching problem for* $(\mathcal{M}, \mu)$ *is* solvable *iff there is a substitution* σ *and a number* $n \in \mathbb{N}$ *such that for all* $1 \le j \le k$ *the equality* $s_j\mu^n = q_j\sigma$ *is valid.*

If $\mathcal{M}$ *only contains one pair* $s > q$ *then we identify* $(\mathcal{M}, \mu)$ *with* $(s > q, \mu)$, *and if* μ *is clear from the context we write* $\mathcal{M}$ *as an abbreviation for* $(\mathcal{M}, \mu)$.

We now give a set of four transformation rules which either detect that a matching problem is not solvable (indicated by $\bot$), or which transform a matching problem into *solved form*. And once we have reached a matching problem in solved form, it is possible to translate it into identity problems.

Definition 8.16 (Transformation of Matching Problems)**.** *Let* $\mathcal{V}$ *be the set of variables and let* $\mu = \{x_1/t_1, \ldots, x_m/t_m\}$ *be a substitution. We define* $\mathcal{V}_{\mathit{incr}} = \{x \in \mathcal{V} \mid$ *there is some* $n \in \mathbb{N}$ *with* $x\mu^n \notin \mathcal{V}\}$ *as the set of* increasing variables.

For each matching problem $(\mathcal{M}, \mu)$ *with* $\mathcal{M} = \mathcal{M}' \uplus \{s \gtrdot q\}$ *with* $q \notin \mathcal{V}$ *there is a corresponding transformation rule.*

(i) $\mathcal{M} \Rightarrow \{s'\mu \gtrdot q' \mid s' \gtrdot q' \in \mathcal{M}\}$, *if* $s \in \mathcal{V}_{\mathit{incr}}$

(ii) $\mathcal{M} \Rightarrow \bot$, *if* $s \in \mathcal{V} \setminus \mathcal{V}_{\mathit{incr}}$

(iii) $\mathcal{M} \Rightarrow \bot$, *if* $s = f(\ldots)$, $q = g(\ldots)$, *and* $f \neq g$

(iv) $\mathcal{M} \Rightarrow \mathcal{M}' \cup \{s_1 \gtrdot q_1, \ldots, s_k \gtrdot q_k\}$, *if* $s = f(s_1, \ldots, s_k)$, $q = f(q_1, \ldots, q_k)$

Otherwise, a matching problem is in solved form.

The following theorem shows that every matching problem $(s \gtrdot q, \mu)$ can be reduced to a set of identity problems.

Theorem 8.17 (Solving Matching Problems)**.** *Let* $(\mathcal{M}, \mu)$ *be a matching problem.*

(i) The transformation rules of Definition 8.16 are confluent and terminating.

(ii) If $\mathcal{M} \Rightarrow \bot$ *then* $\mathcal{M}$ *is not solvable.*

(iii) If $\mathcal{M} \Rightarrow \mathcal{M}'$ *then* $\mathcal{M}$ *is solvable iff* $\mathcal{M}'$ *is solvable.*

(iv) $\mathcal{M}$ *is solvable iff* $\mathcal{M} \Rightarrow^* \mathcal{M}'$ *for some matching problem* $\mathcal{M}' = \{s_1 \gtrdot x_1, \ldots, s_k \gtrdot x_k\}$ *in solved form, such that for all* $i \neq j$ *with* $x_i = x_j$ *the identity problem* $(s_i \approxeq s_j, \mu)$ *is solvable.*

Example 8.18. We illustrate the transformation rules of Definition 8.16 by continuing Example 8.14, where at the end we had to analyze two matching problems for $q = \mathsf{c}(\mathsf{c}(\mathsf{c}(x_1, x_1), x_2), x_3)$ and $\mu = \{x/\mathsf{c}(x, y), y/z, u/y\}$.

- The matching problem $(\mathsf{F}(\mathsf{c}(x, y), z, z, y) \gtrdot q, \mu)$ can be reduced to $\bot$ by rule (iii).
- The other matching problem can be transformed by rules (i) and (iv) into solved form as follows.

$$\begin{aligned}
(\mathsf{c}(x, y) \gtrdot q, \mu) &= \{\mathsf{c}(x, y) \gtrdot \mathsf{c}(\mathsf{c}(\mathsf{c}(x_1, x_1), x_2), x_3)\} \\
&\Rightarrow \{x \gtrdot \mathsf{c}(\mathsf{c}(x_1, x_1), x_2), y \gtrdot x_3\} \\
&\Rightarrow \{\mathsf{c}(x, y) \gtrdot \mathsf{c}(\mathsf{c}(x_1, x_1), x_2), z \gtrdot x_3\} \\
&\Rightarrow \{x \gtrdot \mathsf{c}(x_1, x_1), y \gtrdot x_2, z \gtrdot x_3\} \\
&\Rightarrow \{\mathsf{c}(x, y) \gtrdot \mathsf{c}(x_1, x_1), z \gtrdot x_2, z \gtrdot x_3\} \\
&\Rightarrow \{x \gtrdot x_1, y \gtrdot x_1, z \gtrdot x_2, z \gtrdot x_3\}
\end{aligned}$$

Hence, by Theorem 8.17 all matching problems are not solvable iff the identity problem $(x \approxeq y, \mu)$ is not solvable.

```
Input:  An identity problem $(s \approxeq t, \mu)$.
Output: "Yes", the identity problem is solvable, or "No", it is not.

(i)    While $\mu$ contains a cycle of length $n > 1$ do $\mu := \mu^n$
(ii)   $S := \varnothing$
(iii)  If $s = t$ then stop with result "Yes"
(iv)   If there is a shared position $p$ of $s$ and $t$ such that $s|_p = f(\dots)$ and $t|_p = g(\dots)$
       and $f \neq g$ then stop with result "No"
(v)    If there is a shared position $p$ of $s$ and $t$ such that $s|_p = x$, $t|_p = g(\dots)$, and $x$ is
       not an increasing variable then stop with result "No".
       Repeat this step with $s$ and $t$ exchanged.
(vi)   If there is a shared position $p$ of $s$ and $t$ such that $s|_p = x$, $t|_p = y$, $x \neq y$, and
       $x, y \notin Dom(\mu)$ then stop with result "No"
(vii)  Add the triple $(x, p, t|_p)$ to $S$ for all shared positions $p$ of $s$ and $t$ such that
       $x = s|_p \neq t|_p$ where $x$ is an increasing variable
       Repeat this step with $s$ and $t$ exchanged.
(viii) If $(x, p_1, u_1) \in S$ and $(x, p_2, u_2) \in S$ where
           (a) $u_1$ and $u_2$ are not unifiable or where
           (b) $u_1 = u_2$ and $p_1 < p_2$,
       then stop with result "No"
(ix)   $s := s\mu, t := t\mu$
(x)    Continue with step (iii)
```

Figure 8.19.: An algorithm to decide solvability of identity problems

It only remains to give an algorithm which decides solvability of an identity problem. The full algorithm is presented in Figure 8.19, and we will explain the performed steps one by one.

First we replace the substitution μ by μ^n such that μ^n does not contain *cycles*. Here, a substitution δ contains a cycle of length n iff $\delta = \{x_1/x_2,\ x_2/x_3,\ \dots,\ x_n/x_1,\ \dots\}$ where the x_i are pairwise different variables. Obviously, if δ contains a cycle of length n then in δ^n all variables $x_1, \dots, x_n$ do not belong to the domain any more. Thus, step (i) will terminate and afterwards μ does not contain cycles of length 2 or more.

Note that like in Theorem 8.13 where we replaced μ by $\delta = \mu^{j-i}$, the identity problem $(s \approxeq t, \mu)$ is solvable iff $(s \approxeq t, \mu^n)$ is solvable. Hence, after step (i) we still have to decide solvability of $(s \approxeq t, \mu)$ for the modified μ. The advantage is that now μ has a special structure. For all $x \in Dom(\mu)$ either x is an increasing variable or for some n the term $x\mu^n$ is a variable which is not in the domain of μ. And for such substitutions μ the terms $s, s\mu, s\mu^2, \dots$ finally become *stationary* at each position, i.e., for every position p there is some n such that either all terms $s\mu^n|_p, s\mu^{n+1}|_p, s\mu^{n+2}|_p, \dots$ are of the form $f(\dots)$, or all these terms are the same variable $x \notin \mathit{Dom}(\mu)$. Therefore, it is possible to define $s\mu^\infty$ as the limit of all terms $s, s\mu, s\mu^2, \ldots$.

If the identity problem is solvable then there is some n such that $s\mu^n = t\mu^n$ which is detected in step (iii). The reason is that with steps (ix) and (x) it is iterated over all term pairs (s,t), $(s\mu, t\mu)$, $(s\mu^2, t\mu^2)$, $\ldots$.

On the other hand it might be the case that an identity problem has a *stationary conflict*, i.e., $s\mu^\infty \neq t\mu^\infty$. Then the identity problem is unsolvable since $s\mu^n = t\mu^n$ implies $s\mu^\infty = t\mu^\infty$. However, if the terms $s\mu^\infty$ and $t\mu^\infty$ differ, then there is some position p such that the symbols at position p differ, or $s\mu^\infty|_p$ is a variable and $t\mu^\infty|_p$ is not a variable (or vice versa), or both $s\mu^\infty|_p$ and $t\mu^\infty|_p$ are different variables Hence, if we choose n high enough then p is stationary for both $s\mu^n|_p$ and $t\mu^n|_p$. But then one of three rules (iv)-(vi) will trigger. The reason is that all variables in $s\mu^\infty$ and $t\mu^\infty$ are not from the domain of μ.

Up to now we can detect all solvable identity problems and all identity problems which are not solvable due to a stationary conflicts. However, there remain other identity problems which are neither solvable nor do they possess a stationary conflict. As an example consider $(x \approxeq y, \{x/\mathsf{f}(x), y/\mathsf{f}(y)\})$. Then $s\mu^\infty = \mathsf{f}(\mathsf{f}(\mathsf{f}(\ldots))) = t\mu^\infty$ but this identity problem is not solvable since $x\mu^n = \mathsf{f}^n(x) \neq \mathsf{f}^n(y) = y\mu^n$. We call these identity problems *infinite*.

The remaining steps (ii), (vii), and (viii) are used to detect infinite identity problems. In the set S we store subproblems (x, p, u) such that whenever the identity problem is solvable, then $x\mu^m = u\mu^m$ must hold for some m to make the terms $s\mu^n$ and $t\mu^n$ equal at position p. We give some intuitive arguments why the two abortion criteria in step (viii) are correct. For (viii–a) notice that if u_1 and u_2 are not unifiable then $x\mu^m$ cannot be both $u_1\mu^m$ and $u_2\mu^m$, which would be a conflict. And if in the problem to make $x\mu^m$ equal to $u_1\mu^m$ we again produce the same problem at a lower position, then this will continue forever. Hence, the problem is not solvable, which will be detected by step (viii–b).

The following theorem shows that indeed all answers of the algorithm are correct and it also shows that we will always get an answer. However, especially the termination proof is quite involved since we have to show that the criteria in step (viii) suffice to detect all infinity identity problems.

Theorem 8.20 (Solving Identity Problems)**.** *The algorithm in Figure 8.19 to decide solvability of identity problems is correct and it terminates.*

Example 8.21. We illustrate the algorithm with the identity problem $(x \approxeq y, \mu)$ where $\mu = \{x/\mathsf{f}(y, u_0), y/\mathsf{f}(z, u_0), z/\mathsf{f}(x, u_0), u_0/u_1, u_1/u_0\}$.

As μ contains a cycle of length 2 we replace μ by $\mu^2 = \{x/\mathsf{f}(\mathsf{f}(z, u_0), u_1)$, $y/\mathsf{f}(\mathsf{f}(x, u_0), u_1)$, $z/\mathsf{f}(\mathsf{f}(y, u_0), u_1)\}$. Since $x\mu^\infty = \mathsf{f}(\mathsf{f}(\mathsf{f}(\ldots, u_1), u_0), u_1) = y\mu^\infty$ we know that the problem is solvable or infinite. Hence, the steps (iv)-(vi) will never succeed. We start with $s = x$ and $t = y$. Since the terms are different we add (x, ε, y) and (y, ε, x) to S. In the next iteration we have $s = \mathsf{f}(\mathsf{f}(z, u_0), u_1)$ and $t = \mathsf{f}(\mathsf{f}(x, u_0), u_1)$. Again, the terms are different and we add $(x, 11, z)$ and $(z, 11, x)$ to S. The next iteration yields the new triples $(y, 1111, z)$ and $(z, 1111, y)$, and after having applied μ three times we get the two last triples $(x, 111111, y)$ and $(y, 111111, x)$. Then due to (viii–b) the algorithm terminates with "No".

By simply combining the theorems of this section we have finally obtained a decision procedure which can solve the question whether a loop is also a loop when regarding the strategy given by $\mathcal{Q}$.

Corollary 8.22 (Deciding $\mathcal{Q}$-loops)**.** *For every looping reduction of a TRS or of a DP problem it is decidable whether that reduction is a loop when regarding the strategy* $\mathcal{Q}$*.*

Example 8.23. At the end of Example 8.18 we knew that the given loop respects the strategy iff $(x \approxeq y, \mu)$ is not solvable where $\mu = \{x/\mathsf{c}(x,y), y/z, u/y\}$. We apply the algorithm of Figure 8.19 to show that this identity problem is indeed not solvable, and hence the loop is also $\mathcal{Q}$-looping.

Since μ only contains cycles of length 1, we skip step (i). So, let $s = x$ and $t = y$. Then none of the steps (iii)-(vi) are applicable. Hence, we add (x, ε, y) to S and continue with $s = \mathsf{c}(x,y)$ and $t = z$. Then, in step (v) the algorithm is stopped with answer "No" due to a stationary conflict.

Summary of Chapter 8

In this chapter we have presented various methods to prove that a TRS is non-terminating. First, we have seen that every *loop* gives rise to non-termination. Then we have presented a new contribution, where we have generalized the concept of a loop from full rewriting to $\mathcal{Q}$-restricted rewriting and from TRSs to DP problems. Moreover, we have proven that a TRS is looping iff its initial DP problem is looping. This result is important since it implies that one can only benefit from disproving termination in the DP framework: If a TRS is looping, one will be able to detect that loop in a corresponding DP problem, and often one can detect the loop more easily, since the processors to prove termination often narrow the search space to find a loop considerably.

To further simplify disproving termination, we have introduced a novel processor which can transform infinite non-looping DP problems into looping DP problems by switching from innermost termination to termination.

All of the above methods have already been published by us in a basic version in [GTS05b] where we only considered full and innermost rewriting instead of $\mathcal{Q}$-restricted rewriting. Moreover, the equivalence result between looping TRSs and looping DP problems is only available for full rewriting in [GTS05b].

Finally, we have presented a new algorithm to decide whether a loop for full rewriting is also a loop for $\mathcal{Q}$-restricted rewriting. This extends our work in [GTS05c] which can only approximate this question.

To automate disproving, we propose the following strategy. First one should apply all complete processors of the previous chapters as if one would try to prove termination. This will usually simplify the initial DP problem considerably and it will detect the possibly non-terminating parts. Then one should really try to disprove termination. To obtain a small set $\mathcal{Q}$ one should apply the processor of Theorem 3.34 to remove all terms of $\mathcal{Q}$ which cannot block a reduction any more, and one should apply the processor of Theorem 8.9 to switch to termination, if possible. Note that both these processors drop minimality, but this does not matter, since minimality is not needed when disproving termination. Finally one should try to disprove termination by Theorem 8.5, i.e., by searching for a loop. To this end, one can use one of the methods in [GTS05b, GZ99, Pay06] for finding loops for full termination, and then check whether that loop respects the strategy by our new algorithm where of course, one should guide this search by the evaluation strategy as far as possible.

In the related work of [WS06] a larger class of non-terminating problems has been identified, so called *inner-loops*. These still have some regular structure but up to now, no technique is available which can automatically detect inner-loops that are not already loops. It would be interesting to develop these techniques and to generalize inner-loops from full rewriting to $\mathcal{Q}$-restricted rewriting.

Other related work can be found in [Pla86]. There a TRS $\mathcal{R}$ is already called "non-terminating" if it allows a self-embedding reduction sequence. A reduction sequence $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \dots$ is called self-embedding if there are i and $j > i$ such that $t_j \succsim_{emb} t_i$. Thus, this criterion can be used to detect that $\mathcal{R}$ is not simply-terminating. This is used to avoid infinite reductions in the reduction step of Knuth-Bendix style completion. However these results are incomparable to our work, since their criterion does not imply non-termination with the usual meaning of non-termination.

9. Conclusion

We introduced the DP framework for termination proofs (Chapter 2) which generalizes the classical DP approach [AG00] into a general basis for automated termination proofs. Since it turned out that the classical evaluation strategies of full rewriting and innermost rewriting are not expressive enough – especially if one wants to obtain completeness – we used the more general notion of $\mathcal{Q}$-restricted rewriting.

Then we showed how to formulate the existing components of the DP approach as processors within this framework (Chapters 3–5). In contrast to the DP approach, now these components can be applied at any time during the termination proof and their applicability conditions only concern the current DP problem, not the whole TRSs. For example, we designed a modular and more powerful version of the technique of [Gra95] to switch to innermost termination. Moreover, we developed several new processors and extended existing techniques ([Urb01]) to simplify DP problems, such that far less constraints have to be satisfied, or less new pairs and rules will be generated. Additionally, we introduced semantic notions (of *Cap* and of usable rules) which encompass all known syntactic definitions, and our estimations yield better results than all previous versions. And even more importantly, since our processors only make use of these semantic notions, every future improvement of the estimations can be integrated without having to recheck or adapt a single proof of any of the processors.

Afterwards, we presented several new processors to handle applicative TRSs, which are first-order TRSs of a special form that can be used to represent higher-order functions (Chapter 6). Here, we extended the existing work about transforming applicative TRSs to standard functional form ([KKSV96]) to DP problems, which entailed the new problem of considering the evaluation strategy in that transformation. There we detected that some of the desired processors would be unsound, but we solved this problem by combining the transformation with those processors that are based on well-founded orders. This is sufficient since most of the other processors are equally powerful on transformed and untransformed DP problems.

As next step we adapted the powerful technique of semantic labeling [Zan95] to the DP framework (Chapter 7). Our work included the integration of strategies for both models and quasi-models, we investigated completeness of semantic labeling, and we detected which processors may be used between labeling and unlabeling.

Finally, we illustrated how to disprove termination (Chapter 8). A major contribution was already made in the previous chapters since for all processors in Chapters 3–7, we also investigated their completeness which allows us to use them also when proving non-termination. To finally detect non-termination we try to find loops, a notion we have generalized from full to $\mathcal{Q}$-restricted rewriting and from TRSs to DP problems. To this end, we designed a novel algorithm which can decide whether a loop for full rewriting respects a given evaluation strategy, and additionally other techniques were presented that allow to switch to a more liberal strategy before trying to detect a loop.

Of course, there are other techniques to analyze termination that have not yet been adapted to the DP framework like match-bounds [GHW03, GHWZ07] or the size-change principle [LJB01, TG03, TG05]. Therefore, in [GTS05a, Theorem 36] it is shown how to integrate arbitrary existing termination techniques into the DP framework. The main idea is to prove $\mathcal{Q}$-termination of $\mathcal{P} \cup \mathcal{R}$ in order to show that $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ is finite. In this way, these techniques can benefit from other processors which were applied before. This increases their applicability and power considerably. However, it would of course be more advantageous to integrate these techniques more closely into the DP framework like it was done for semantic labeling. This would be an interesting future work.

All of our techniques not only work in theory but they can be efficiently mechanized, too. This can be done in two steps.

First, one needs an efficient implementation of each individual processor. For the more challenging processors – those of Chapters 4, 6, 7, and 8 – one can find corresponding techniques in our work [CSL$^+$06, FGM$^+$07, GTSF03, GTSF06, STA$^+$07] and in the work of [CLS06, CMTU05, GZ99, Pay06, HM05, KM07, KZ06, Zan05b, ZHM07]. Here, the most recent techniques often use SAT encodings and benefit from modern SAT solvers.

And second, one definitely needs a strategy when to apply each technique. One can take the following general purpose strategy as starting point, but of course one might want to adapt it for certain classes of input problems, or if new techniques become available.

Our strategy works as follows. We start with the tree that only contains the initial DP problem as node. And whenever we detect an unprocessed DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ in this tree, we try to successfully apply a processor in the following sequence, i.e., we apply the first processor *Proc* which satisfies $\mathit{Proc}((\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)) \neq \{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$.

(i) Apply very fast and complete processors
   (a) Delete all edges in the pair-graph due to the dependency graph (Theorem 3.3 with estimation of Definition 3.9)
   (b) Decompose pair-graph (Theorem 3.4)
   (c) Delete all edges in the pair-graph due to the dependency graph (Theorem 3.3 with star-estimation of Definition 3.31)
   (d) Switch to innermost termination (Theorem 3.14)
   (e) Remove all unusable rules (Theorem 3.25)
   (f) Remove non-blocking terms in $\mathcal{Q}$ (Theorem 3.36)
   (g) Switch from applicative to functional form in the innermost case (Theorem 6.8)

(ii) Apply fast and complete processors
   (a) Remove pairs by the subterm criterion (Theorem 4.41)
   (b) Switch from applicative to functional form in the termination case by using monotonic reduction pairs (Theorem 6.17 (C))
   (c) Remove all unneeded rules by using monotonic reduction pairs (Theorems 4.20 and 6.17 (B))
   (d) Remove further rules by using monotonic reduction pairs (Theorems 4.22 and 6.17 (B))

If one has found a reduction order for one of the above processors one should directly try to simplify the resulting DP problem further with the help of this reduction order.

For example, the constraints to successfully apply (ii–b) do not require a strict decrease. However, if it turns out that some pairs or rules are strictly decreasing one can directly simplify the DP problem further by (ii–d).

(iii) Apply powerful and complete processors

Remove pairs by using "simple" reduction pairs (such as RPO, KBO, linear polynomial orders with low coefficients)

- (a) For applicative DP problems use reduction pair processor (Theorem 6.22), or if a DP problem is not even π-proper, try combination with argument filter processor (Theorem 4.38)
- (b) For non-applicative DP problems use reduction pair processors (Theorems 4.27 and 4.32)

(iv) Apply complete transformations

- (a) Rewrite pairs (Theorem 5.10)
- (b) Narrow pairs (Theorem 5.19)
- (c) Instantiate pairs (Theorems 5.3 and 5.5)

Here, the processors in steps (iv–a) and (iv–b) should only be applied if their application is complete. Moreover, one should directly apply all pair transformations before restarting at the beginning of our strategy. The reason is that often many transformation have to be performed, before a reduction pair processor can be applied afterwards. However, the processors of step (i) should be tried after every transformation.

Since the transformations can often be applied infinitely many times, we have identified "safe" transformations which are guaranteed to terminate. More details can be found in [GTSF06, Definition 33].

(v) Apply very powerful and complete processors

Repeat step (iii) with "complex" reduction pairs (such as negative or non-linear polynomial orders and matrix interpretations)

(vi) Try to disprove termination

- (a) Remove all non-blocking terms from $\mathcal{Q}$ (Theorem 3.34)
- (b) Switch to termination (Theorem 8.9)
- (c) Find a loop (Theorem 8.5)

After steps (vi–a) and (vi–b) have been applied, one should not restart with the global strategy in step (i), but directly try step (vi–c). If one cannot find a loop, one should not apply any step of (vi) at all, since otherwise minimality would be lost.

(vii) Apply incomplete transformations

Reapply step (iv) but even allow incomplete transformations

(viii) Apply semantic labeling

Use the technique of labeling and unlabeling (Theorem 7.30). For the labeling always use full labeling (Theorems 7.10 and 7.23) and for quasi-models and linear sets $\mathcal{Q}$ combine it with Theorem 7.15 to preserve minimality. For the simplification between labeling and unlabeling use the chain-identifying processors of steps (i–a)–(i–c) and (i–e) as well as the processors of steps (ii–c), (ii–d), and (iii) in combination with linear polynomial orders.

Our strategy has two global properties. All DP problems *in the tree* have their minimality flag enabled and the roots of the pairs are always head symbols. (Note that minimality is lost *locally* when trying to disprove termination and when semantic labeling is used with quasi-models and non-linear sets $\mathcal{Q}$ (steps (vi) and (viii)). To achieve the former property, we did not allow the processors of Theorems 4.12, 6.12, and 6.17 (A). And because of the latter property we did not integrate Theorem 4.39.

All other processors of this thesis that cannot be found in our strategy are not integrated, because they are subsumed by some strictly more powerful processor.

A large number of processors (including all that have been presented in this thesis) have been implemented in our automated termination tool AProVE [GST06]. Moreover, we have designed a dedicated strategy similar to the one above for every class of termination problems, i.e., there is one strategy for TRSs, one for Prolog-programs, one for Haskell-programs, etc.

The accumulated effect[38] of our contributions can be seen at the annual international Termination Competition [MZ07]. Due to the DP framework, since the beginning of the competition in the year 2004, AProVE was the most powerful system in the term rewriting category, both for proving and for disproving termination. Here, the tools were tested on the examples from the termination problem data base (TPDB) [TPDB], a collection of termination problems from several sources and different areas of computer science. To give some numbers, in the competition of this year where every tool had two minutes to analyze each TRS, AProVE could detect that 723 of the 975 TRSs are terminating and it could disprove termination of 128 systems, whereas the second most powerful tools could only prove termination of 574 TRSs and disprove termination of 117 systems.

This demonstrates that the DP framework is indeed very well suited for automation and for application in practice. The fact that the most successful of the other termination tools Cariboo [FGK02], CiME,[39] Jambox,[40] Matchbox [Wal04], MultumNonMulta,[41] MU-TERM [AGIL07], NTI [Pay06], TEPARLA,[42] TERMPTATION,[43] TORPA [Zan05b], TPA [Kop06], TTT [HM07], TTT2,[44] and TTTbox[45] also use dependency pairs, additionally fortifies this claim.

To summarize, without our contributions, AProVE would not be the winner of the competition in the last four years.

---

[38] To empirically illustrate the advantages of a particular technique, we refer to the large amount of experimental data which is available in [GTS05b, GTSF03, GTSF04, GTSF06, TGS04] and at `http://aprove.informatik.rwth-aachen.de/eval/#experiments`.

[39] Available at `http://cime.lri.fr/`.

[40] Available at `http://joerg.endrullis.de/`.

[41] Available at `http://www.theory.informatik.uni-kassel.de/~dieter/multum/`.

[42] Available at `http://www.win.tue.nl/~hzantema/torpa.html`.

[43] Available at `http://www.lsi.upc.es/~albert/term.html`.

[44] Available at `http://colo6-c703.uibk.ac.at/ttt2/`.

[45] Available at `http://cl-informatik.uibk.ac.at/~mkorp/TTTbox.html`.

Since the combination of techniques within the DP framework leads to a very modular, flexible, and powerful approach, the DP framework is particularly suitable as a basis for future research on automated termination proving. We see four main directions of research:

While there already exist several powerful processors, these processors are not yet sufficient to handle all termination problems occurring in practice. Therefore, one important topic for further work is the improvement of the existing processors and the development of new processors which are particularly fast or particularly powerful for certain classes of DP problems. (Some more detailed ideas have already been given in the summaries of each of the Chapters 3 – 8.)

Another important line of research is the development of strategies to decide which processor should be applied next on a particular DP problem. We have designed such strategies for the current set of processors, but with every new technique, the development becomes more complex: Even in the presented strategy one should use time-outs in specific steps, because otherwise not even step (vi) would be reached if one choses certain classes of reduction pairs in step (v). (Exchanging these two steps would not help since the current step (vi) – if used without resource limits – would then block step (v).) Thus, distributing the resource time on the available processors in such a way that one benefits from all available techniques is a major problem that needs to be investigated.

Since current termination provers become more and more complicated, the chances are high that not every generated proof is correct. And indeed, in the last years some provers returned wrong answers due to a bug in the implementation of the theorems. Therefore, verifying termination proofs becomes a more and more important task. Here, the research has just started recently with the CoLoR and A3PAT projects [BDC$^+$06, CCF$^+$07].

The last direction is about the connection between term rewriting and programming languages. Although there already exist transformations from logic- and functional-programming into TRSs that work well in practice, they can still be improved to handle and exploit more features of the language, e.g., built-in data-structures, types, etc. Moreover, successful transformational approaches to prove termination for the important class of imperative programs are still missing. On the other hand it would also be interesting, to adapt the powerful termination techniques of term rewriting such that they become applicable directly on programs, and – vice versa – to integrate termination techniques from other areas into term rewriting.

Bibliography

[Abe04] A. Abel. Termination checking with types. *RAIRO – Theoretical Informatics and Applications*, 38(4):277–319, 2004.

[AG00] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.

[AGIL07] B. Alarcón, R. Gutiérrez, J. Iborra, and S. Lucas. Proving termination of context-sensitive rewriting with MU-TERM. In *Proceedings of the 6th Spanish Conference on Programming and Computer Languages (PROLE '06)*, Electronic Notes in Theoretical Computer Science, 2007. To appear. Tool available at `http://www.dsic.upv.es/~slucas/csr/termination/muterm/`.

[AM93] G. Aguzzi and U. Modigliani. Proving termination of logic programs by transforming them into equivalent term rewriting systems. In *Proceedings of the 13th Conference on Foundations of Software Technology and Theoretical Computer Science (FST & TCS '93)*, volume 761 of *Lecture Notes in Computer Science*, pages 114–124, 1993.

[AY03] T. Aoto and T. Yamada. Termination of simply typed term rewriting systems by translation and labelling. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications (RTA '03)*, volume 2706 of *Lecture Notes in Computer Science*, pages 380–394, 2003.

[AY04] T. Aoto and T. Yamada. Termination of simply-typed applicative term rewriting systems. In *Proceedings of the 2nd International Workshop on Higher-Order Rewriting (HOR '04)*, pages 61–65, 2004.

[AY05] T. Aoto and T. Yamada. Dependency pairs for simply typed term rewriting. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications (RTA '05)*, volume 3467 of *Lecture Notes in Computer Science*, pages 120–134, 2005.

[AZ95] T. Arts and H. Zantema. Termination of logic programs using semantic unification. In *Proceedings of the 5th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR '95)*, volume 1048 of *Lecture Notes in Computer Science*, pages 219–233, 1995.

[BCDO06] J. Berdine, B. Cook, D. Distefano, and P. O'Hearn. Automatic termination proofs for programs with shape-shifting heaps. In *Proceedings of the 18th International Conference on Computer Aided Verification (CAV '06)*, volume 4144 of *Lecture Notes in Computer Science*, pages 386–400, 2006.

[BDC$^+$06] F. Blanqui, W. Delobel, S. Coupet-Grimal, S. Hinderer, and A. Koprowski. CoLoR, a Coq library on rewriting and termination. In *Proceedings of the 8th*

International Workshop on Termination (WST 06'), pages 69–73, 2006. See also `http://color.loria.fr/`.

[BFG⁺04] G. Barthe, M. J. Frade, E. Giménez, L. Pinto, and T. Uustalu. Type-based termination of recursive definitions. *Mathematical Structures in Computer Science*, 14(1):1–45, 2004.

[BFR00] C. Borralleras, M. Ferreira, and A. Rubio. Complete monotonic semantic path orderings. In *Proceedings of the 17th International Conference on Automated Deduction (CADE '00)*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 346–364, 2000.

[Bla04] F. Blanqui. A type-based termination criterion for dependently-typed higher-order rewrite systems. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA '04)*, volume 3091 of *Lecture Notes in Computer Science*, pages 24–39, 2004.

[BMS05] A. R. Bradley, Z. Manna, and H. B. Sipma. Termination of polynomial programs. In *Proceedings of the 6th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI '05)*, volume 3385 of *Lecture Notes in Computer Science*, pages 113–129, 2005.

[BN98] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge, 1998.

[BR01] C. Borralleras and A. Rubio. A monotonic higher-order semantic path ordering. In *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR '01)*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 531–547, 2001.

[CCF⁺07] E. Contejean, P. Courtieu, J. Forest, O. Pons, and X. Urbain. Certification of automated termination proofs. In *Proceedings of the 6th International Symposium on Frontiers of Combining Systems (FroCoS '07)*, Lecture Notes in Artificial Intelligence, 2007. To appear. See also `http://www3.iie.cnam.fr/~urbain/a3pat/`.

[CLS05] M. Codish, V. Lagoon, and P. Stuckey. Testing for termination with monotonicity constraints. In *Proceedings of the 21th International Conference on Logic Programming (ICLP '05)*, volume 3668 of *Lecture Notes in Computer Science*, pages 326–340, 2005.

[CLS06] M. Codish, V. Lagoon, and P. J. Stuckey. Solving partial order constraints for LPO termination. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, volume 4098 of *Lecture Notes in Computer Science*, pages 4–18, 2006.

[CMTU05] E. Contejean, C. Marché, A. P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning*, 34(4):325–363, 2005.

[CPR06] B. Cook, A. Podelski, and A. Rybalchenko. Termination proofs for systems code. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '06)*, pages 415–426. ACM Press, 2006.

[CS02] M. Colon and H. B. Sipma. Practical methods for proving program termination. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV '02)*, volume 2034 of *Lecture Notes in Computer Science*, pages 442–454, 2002.

[CSL$^+$06] M. Codish, P. Schneider-Kamp, V. Lagoon, R. Thiemann, and J. Giesl. SAT solving for argument filterings. In *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '06)*, volume 4246 of *Lecture Notes in Artificial Intelligence*, pages 30–44, 2006.

[DD94] D. De Schreye and S. Decorte. Termination of logic programs: The never-ending story. *Journal of Logic Programming*, 19/20:199–260, 1994.

[Der87] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.

[Der04] N. Dershowitz. Termination by abstraction. In *Proceedings of the 20th International Conference on Logic Programming (ICLP '04)*, volume 3132 of *Lecture Notes in Computer Science*, pages 1–18, 2004.

[DS02] D. De Schreye and A. Serebrenik. Acceptability with general orderings. In *Computational Logic: Logic Programming and Beyond*, volume 2407 of *Lecture Notes in Computer Science*, pages 187–210, 2002.

[EWZ06] J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR '06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 574–588, 2006.

[FGK02] O. Fissore, I. Gnaedig, and H. Kirchner. Cariboo: An induction based proof tool for termination with strategies. In *Proceedings of the 4th International Conference on Principles and Practice of Declarative Programming (PPDP '02)*, pages 62–73. ACM Press, 2002. Tool available at `http://protheo.loria.fr/softwares/cariboo`.

[FGM$^+$07] C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT solving for termination analysis with polynomial interpretations. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT '07)*, volume 4501 of *Lecture Notes in Computer Science*, pages 340–354, 2007.

[GA01] J. Giesl and T. Arts. Verification of Erlang processes by dependency pairs. *Applicable Algebra in Engineering, Communication and Computing*, 12(1,2):39–72, 2001.

[GAO02] J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *Journal of Symbolic Computation*, 34(1):21–58, 2002.

[GHW03] A. Geser, D. Hofbauer, and J. Waldmann. Match-bounded string rewriting systems. In *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS '03)*, volume 2747 of *Lecture Notes in Computer Science*, pages 449–459, 2003.

[GHWZ07] A. Geser, D. Hofbauer, J. Waldmann, and H. Zantema. On tree automata that certify termination of left-linear term rewriting systems. *Information and Computation*, 205(4):512–534, 2007.

[Gie95] J. Giesl. Termination analysis for functional programs using term orderings. In *Proceedings of the 2nd International Static Analysis Symposium (SAS '95)*, volume 983 of *Lecture Notes in Computer Science*, pages 154–171, 1995.

[Gra95] B. Gramlich. Abstract relations between restricted termination and confluence properties of rewrite systems. *Fundamenta Informaticae*, 24:3–23, 1995.

[GSST06] J. Giesl, S. Swiderski, P. Schneider-Kamp, and R. Thiemann. Automated termination analysis for Haskell: From term rewriting to programming languages. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, volume 4098 of *Lecture Notes in Computer Science*, pages 297–312, 2006.

[GST06] J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the DP framework. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR '06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 281–286, 2006. Tool available at `http://aprove.informatik.rwth-aachen.de/`.

[GTS05a] J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '04)*, volume 3452 of *Lecture Notes in Artificial Intelligence*, pages 301–331, 2005.

[GTS05b] 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 (FroCoS '05)*, volume 3717 of *Lecture Notes in Artificial Intelligence*, pages 216–231, 2005.

[GTS05c] J. Giesl, R. Thiemann, and P. Schneider-Kamp. Disproving termination of term rewriting. Talk given at the *Workshop on Disproving – Non-Theorems, Non-Validity, Non-Provability*, 2005.

[GTSF03] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Improving dependency pairs. In *Proceedings of the 10th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '03)*, volume 2850 of *Lecture Notes in Artificial Intelligence*, pages 167–182, 2003.

[GTSF04] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA '04)*, volume 3091 of *Lecture Notes in Computer Science*, pages 210–220, 2004.

[GTSF06] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.

[GTSS07] J. Giesl, R. Thiemann, S. Swiderski, and P. Schneider-Kamp. Proving termination by bounded increase. In *Proceedings of the 21st Conference on Automated Deduction (CADE '07)*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 443–459, 2007.

[GW93] H. Ganzinger and U. Waldmann. Termination proofs of well-moded logic programs via conditional rewrite systems. In *Proceedings of the 3rd International Workshop on Conditional Term Rewriting (CTRS '93)*, volume 656 of *Lecture Notes in Computer Science*, pages 216–222, 1993.

[GWB98] J. Giesl, C. Walther, and J. Brauburger. Termination analysis for functional programs. *Automated Deduction – A Basis for Applications*, 3:135–164. Kluwer Academic Publishers, 1998.

[GZ99] A. Geser and H. Zantema. Non-looping string rewriting. *RAIRO Theoretical Informatics and Applications*, 33(3):279–302, 1999.

[HM05] N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1,2):172–199, 2005.

[HM06a] N. Hirokawa and A. Middeldorp. Predictive labeling. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, volume 4098 of *Lecture Notes in Computer Science*, pages 313–327, 2006.

[HM06b] N. Hirokawa and A. Middeldorp. Uncurrying for termination. In *Proceedings of the 3rd International Workshop on Higher-Order Rewriting (HOR '06)*, pages 19–24, 2006.

[HM07] N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool: Techniques and features. *Information and Computation*, 205(4):474–511, 2007. Tool available at `http://colo6-c703.uibk.ac.at/ttt/`.

[HW06] D. Hofbauer and J. Waldmann. Termination of string rewriting with matrix interpretations. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, volume 4098 of *Lecture Notes in Computer Science*, pages 328–342, 2006.

[JR07] J.-P. Jouannaud and A. Rubio. Polymorphic higher-order recursive path orderings. *Journal of the ACM*, 54(1):1–48, 2007.

[KB70] D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon, 1970.

[KKS98] M. R. K. Krishna Rao, D. Kapur, and R. Shyamasundar. Transformational methodology for proving termination of logic programs. *Journal of Logic Programming*, 34(1):1–42, 1998.

[KKSV96] R. Kennaway, J. W. Klop, R. Sleep, and F.-J. de Vries. Comparing curried and uncurried rewriting. *Journal of Symbolic Computation*, 21(1):15–39, 1996.

[KL80] S. Kamin and J. J. Lévy. Two generalizations of the recursive path ordering. Unpublished Manuscript, University of Illinois, IL, USA, 1980.

[KM07] A. Koprowski and A. Middeldorp. Predictive labeling with dependency pairs using SAT. In *Proceedings of the 21st Conference on Automated Deduction (CADE '07)*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 410–425, 2007.

[KNT99] K. Kusakari, M. Nakamura, and Y. Toyama. Argument filtering transformation. In *Proceedings of the 1st International Conference on Principles and Practice of Declarative Programming (PPDP '99)*, volume 1702 of *Lecture Notes in Computer Science*, pages 48–62, 1999.

[Kop06] A. Koprowski. TPA: Termination Proved Automatically. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, volume 4098 of *Lecture Notes in Computer Science*, pages 257–266, 2006. Tool available at `http://www.win.tue.nl/tpa/`.

[Kru60] J. B. Kruskal. Well-quasiorderings, the Tree Theorem, and Vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95:210–223, 1960.

[Kus01] K. Kusakari. On proving termination of term rewriting systems with higher-order variables. *IPSJ Transactions on Programming*, 42(SIG 7 (PRO 11)):35–45, 2001.

[KZ06] A. Koprowski and H. Zantema. Automation of recursive path ordering for infinite labelled rewrite systems. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR '06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 332–346, 2006.

[Lan79] D. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.

[LB98] M. Lifantsev and L. Bachmair. An LPO-based termination ordering for higher-order terms without λ-abstraction. In *Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics (TPHOLs '98)*, volume 1479 of *Lecture Notes in Computer Science*, pages 277–293, 1998.

[LJB01] C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. In *ACM Symposium on Principles of Programming Languages (POPL '01)*, pages 81–92, 2001.

[LMS03] V. Lagoon, F. Mesnard, and P. J. Stuckey. Termination analysis with types is more accurate. In *Proceedings of the 19th International Conference on Logic Programming (ICLP '03)*, volume 2916 of *Lecture Notes in Computer Science*, pages 254–268, 2003.

[Luc05] S. Lucas. Polynomials over the reals in proofs of termination: from theory to practice. *RAIRO Theoretical Informatics and Applications*, 39(3):547–586, 2005.

[Mar94] M. Marchiori. Logic programs as term rewriting systems. In *Proceedings of the 4th International Conference on Algebraic and Logic Programming (ALP '94)*, volume 850 of *Lecture Notes in Computer Science*, pages 223–241, 1994.

[Mar96] M. Marchiori. Proving existential termination of normal logic programs. In *Proceedings of the 5th International Conference on Algebraic Methodology and Software Technology (AMAST '96)*, volume 1101 of *Lecture Notes in Computer Science*, pages 375–390, 1996.

[Mid94] A. Middeldorp. A simple proof to a result of Bernhard Gramlich. Presentation given at the 5th Japanese Term Rewriting Meeting, Tsukuba, 1994. Available at `http://cl-informatik.uibk.ac.at/~ami/research/publications/misc/bg.pdf`.

[Mid01] A. Middeldorp. Approximating dependency graphs using tree automata techniques. In *Proceedings of the 1st International Joint Conference on Automated Reasoning (IJCAR '01)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 593–610, 2001.

[Mid02] A. Middeldorp. Approximations for strategies and termination. In *Proceedings of the 2nd International Workshop on Reduction Strategies in Rewriting and Programming (WRS '02)*, volume 70(6) of *Electronic Notes in Theoretical Computer Science*, pages 1–20, 2002.

[MR03] F. Mesnard and S. Ruggieri. On proving left termination of constraint logic programs. *ACM Transactions on Computational Logic*, 4(2):207–259, 2003.

[MZ07] C. Marché and H. Zantema. The termination competition. In *Proceedings of the 18th International Conference on Rewriting Techniques and Applications (RTA '07)*, volume 4533 of *Lecture Notes in Computer Science*, pages 303–313, 2007. See also `http://www.lri.fr/~marche/termination-competition/`.

[Ohl01] E. Ohlebusch. Semantic labeling meets dependency pairs. In *Proceedings of the 5th International Workshop on Termination (WST '01)*, pages 36–38, 2001.

[Pay06] Étienne Payet. Detecting non-termination of term rewriting systems using an unfolding operator. In *Proceedings of the 16th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR '06)*, volume 4407 of *Lecture Notes in Computer Science*, pages 177–193, 2006. Tool available at `http://www2.univ-reunion.fr/~epayet/Research/TRS/TRSanalyses.html`.

[Pla86] D. A. Plaisted. A simple non-termination test for the Knuth-Bendix method. In *Proceedings of the 8th International Conference on Automated Deduction (CADE '86)*, volume 230 of *Lecture Notes in Computer Science*, pages 79–88, 1986.

[Pol96] J. van de Pol. *Termination of higher-order rewrite systems.* PhD thesis, Utrecht, 1996.

[PR04a] A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. In *Proceedings of the 5th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI '04)*, volume 2937 of *Lecture Notes in Computer Science*, pages 239–251, 2004.

[PR04b] A. Podelski and A. Rybalchenko. Transition invariants. In *Proceedings of the 19th IEEE Symposium on Logic in Computer Science (LICS '04)*, pages 32–41. IEEE Computer Society, 2004.

[PS97] S. E. Panitz and M. Schmidt-Schauss. TEA: Automatically proving termination of programs in a non-strict higher-order functional language. In *Proceedings of the 4th International Static Analysis Symposium (SAS '97)*, volume 1302 of *Lecture Notes in Computer Science*, pages 345–360, 1997.

[Raa97] F. van Raamsdonk. Translating logic programs into conditional rewriting systems. In *Proceedings of the 14th International Conference on Logic Programming (ICLP '97)*, pages 168–182. MIT Press, 1997.

[SGST06] P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated termination analysis for logic programs by term rewriting. In *Proceedings of the 16th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR '06)*, volume 4407 of *Lecture Notes in Computer Science*, pages 177–193, 2006.

[SK05] M. Sakai and K. Kusakari. On dependency pair method for proving termination of higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E88-D(3):583–593, 2005.

[Sma04] J.-G. Smaus. Termination of logic programs using various dynamic selection rules. In *Proceedings of the 20th International Conference on Logic Programming (ICLP '04)*, volume 3132 of *Lecture Notes in Computer Science*, pages 43–57, 2004.

[STA$^+$07] P. Schneider-Kamp, R. Thiemann, E. Annov, M. Codish, and J. Giesl. Proving termination using recursive path orders and SAT solving. In *Proceedings of the 6th International Symposium on Frontiers of Combining Systems (FroCoS '07)*, Lecture Notes in Artificial Intelligence, 2007. To appear.

[Ste95] J. Steinbach. Simplification orderings: History of results. *Fundamenta Informaticae*, 24:47–87, 1995.

[SWS01] M. Sakai, Y. Watanabe, and T. Sakabe. An extension of dependency pair method for proving termination of higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E84-D(8):1025–1032, 2001.

[Ter03] Terese. *Term Rewriting Systems.* Cambridge Tracts in Theoretical Computer Science 55. Cambridge University Press, 2003.

[TG03] R. Thiemann and J. Giesl. Size-change termination for term rewriting. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications (RTA '03)*, volume 2706 of *Lecture Notes in Computer Science*, pages 264–278, 2003.

[TG05] R. Thiemann and J. Giesl. The size-change principle and dependency pairs for termination of term rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 16(4):229–270, 2005.

[TGS04] R. Thiemann, J. Giesl, and P. Schneider-Kamp. Improved modular termination proofs using dependency pairs. In *Proceedings of the 2nd International Joint Conference on Automated Reasoning (IJCAR '04)*, volume 3097 of *Lecture Notes in Artificial Intelligence*, pages 75–90, 2004.

[Tiw04] A. Tiwari. Termination of linear programs. In *Proceedings of the 16th International Conference on Computer Aided Verification (CAV '04)*, volume 3097 of *Lecture Notes in Artificial Intelligence*, pages 70–82, 2004.

[TM07] R. Thiemann and A. Middeldorp. Innermost termination of rewrite systems by labeling. In *Proceedings of the 7th International Workshop on Reduction Strategies in Rewriting and Programming (WRS '07)*, Electronic Notes in Theoretical Computer Science, 2007. To appear.

[Toy87] Y. Toyama. Counterexamples to the termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, 1987.

[Toy04] Y. Toyama. Termination of S-expression rewriting systems: Lexicographic path ordering for higher-order terms. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA '04)*, volume 3091 of *Lecture Notes in Computer Science*, pages 40–54, 2004.

[TPDB] The termination problem data base (TPDB). Available at `http://www.lri.fr/~marche/tpdb/`.

[TT00] A. Telford and D. Turner. Ensuring termination in ESFP. *Journal of Universal Computer Science*, 6(4):474–488, 2000.

[TZGS07] R. Thiemann, H. Zantema, J. Giesl, and P. Schneider-Kamp. Adding constants to string rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 2007. To appear.

[Urb01] X. Urbain. Automated incremental termination proofs for hierarchically defined term rewriting systems. In *Proceedings of the 1st International Joint Conference on Automated Reasoning (IJCAR '01)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 485–498, 2001.

[Wal94] C. Walther. On proving the termination of algorithms by machine. *Artificial Intelligence*, 71(1):101–157, 1994.

[Wal04] J. Waldmann. Matchbox: A tool for match-bounded string rewriting. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA '04)*, volume 3091 of *Lecture Notes in Computer Science*, pages 85–94, 2004. Tool available at `http://dfa.imn.htwk-leipzig.de/matchbox/`.

[WS06] Y. Wang and M. Sakai. On non-looping term rewriting. In *Proceedings of the 8th International Workshop on Termination (WST '06)*, pages 17–21, 2006.

[Xi02] H. Xi. Dependent types for program termination verification. *Higher-Order and Symbolic Computation*, 15(1):91–131, 2002.

[Zan95] H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.

[Zan05a] H. Zantema. Reducing right-hand sides for termination. In *Processes, Terms and Cycles: Steps on the Road to Infinity*, volume 3838 of *Lecture Notes in Computer Science*, pages 173–197, 2005.

[Zan05b] H. Zantema. Termination of string rewriting proved automatically. *Journal of Automated Reasoning*, 34:105–139, 2005. Tool available at `http://www.win.tue.nl/~hzantema/torpa.html`.

[ZHM07] H. Zankl, N. Hirokawa, and A. Middeldorp. Constraints for argument filterings. In *Proceedings of the 33rd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '07)*, volume 4362 of *Lecture Notes in Computer Science*, pages 579–590, 2007.

A. Proofs

A.2. Proofs of Chapter 2

Proof of Lemma 2.4. Obvious. ☐

Proof of Lemma 2.6. We show both directions separately. First consider that there is some $q \in \mathcal{Q}' \cap NF(\mathcal{Q})$. As $q \in \mathcal{Q}'$, obviously $q \notin NF(\mathcal{Q}')$ and hence, $NF(\mathcal{Q}) \nsubseteq NF(\mathcal{Q}')$.

For the other direction let there be some t with $t \in NF(\mathcal{Q})$ and $t \notin NF(\mathcal{Q}')$. From the latter we know that t contains a subterm $q\sigma$ with $q \in \mathcal{Q}'$. Note that $q \in NF(\mathcal{Q})$ as $t \in NF(\mathcal{Q})$ and as rewriting is closed under contexts and substitutions. But this shows $\mathcal{Q}' \cap NF(\mathcal{Q}) \neq \varnothing$. ☐

Proof of Theorem 2.11. (i) $\Rightarrow$ (ii): We define $^\flat$ as the inverse operation to $^\sharp$, i.e., $(f^\sharp(t_1, \ldots, t_n))^\flat = f(t_1, \ldots, t_n)$. Let $s_1 \to t_1, s_2 \to t_2, \ldots$ be an infinite $(DP(\mathcal{R}), \mathcal{Q}, \mathcal{R})$-chain with $t_1\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} s_2\sigma, t_2\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} s_2\sigma, \ldots$. Thus, $s_1\sigma^\flat = \ell_1\sigma \xrightarrow{\mathcal{Q}}_{\mathcal{R}} r_1\sigma = C_1[t_1\sigma^\flat] \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} C_1[s_2\sigma^\flat] = C_1[\ell_2\sigma] \xrightarrow{\mathcal{Q}}_{\mathcal{R}} C_1[r_2\sigma] = C_1[C_2[t_3\sigma^\flat]] \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} C_1[C_2[s_3\sigma^\flat]] \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \ldots$ is an infinite $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ reduction.

(ii) $\Rightarrow$ (iii): Obvious, as every minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain is also a $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain.

(iii) $\Rightarrow$ (i): Suppose, we have an infinite $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$-reduction. Then, there is minimal (w.r.t. $\rhd$) term t_0 that starts an infinite reduction where all its proper subterms are $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ terminating. Thus, $t_0 \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R},>\varepsilon} s_1 = \ell_1\sigma \xrightarrow{\mathcal{Q}}_{\mathcal{R},\varepsilon} r_1\sigma$ and $r_1\sigma$ is $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ non-terminating. Again, there must be a minimal non-terminating subterm t_1 of $r_1\sigma$. By minimality of t_1 the root of t_1 must be defined. Moreover, as all proper subterms of $\ell_1\sigma$ are $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ terminating the term r_1 cannot be a proper subterm of ℓ_1. Hence, the step from s_1 to t_1 must correspond to a dependency pair of $\mathcal{R}$. Thus, $s_1^\sharp \to_{DP(\mathcal{R})} t_1^\sharp$ is a reduction at the root position. Continuing in this way, we get an infinite sequence of dependency pairs. To show that this sequence really is a chain we have to show that all $s_i^\sharp$ are in $\mathcal{Q}$-normal form. First note that all proper subterms of all s_i are in $\mathcal{Q}$-normal form. Hence, all proper subterms of all $s_i^\sharp$ are in $\mathcal{Q}$-normal form, too. Furthermore, the root of each $s_i^\sharp$ is a tuple symbol, thus no s_i can be matched by a term of $\mathcal{Q}$, too.

Finally, as all t_i are chosen minimal and as all $t_i^\sharp$ have a tuple symbol as root that is not contained in $\mathcal{Q}$, all $t_i^\sharp$ are $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$-terminating. ☐

Proof of Lemma 2.17. Let some $(\mathcal{P}', \mathcal{Q}', \mathcal{R}', f') \in Proc((\mathcal{P}, \mathcal{Q}, \mathcal{R}, f))$ be infinite and suppose that $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ is not infinite. Thus, $\mathcal{R}$ is $\mathcal{Q}$-terminating and $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ is finite. Due to $\mathcal{Q}$-termination of $\mathcal{R}$, every $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain is minimal and thus, there is no infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain, even if $f = \mathbf{m}$.

Note that by Lemma 2.4 $\mathcal{Q}'$-termination of $\mathcal{R}'$ follows from $\mathcal{Q}$-termination of $\mathcal{R}$. So if $(\mathcal{P}', \mathcal{Q}', \mathcal{R}', f')$ is infinite, there must be an infinite $(\mathcal{P}', \mathcal{Q}', \mathcal{R}')$-chain. But then this is also a $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain which contradicts the observation above. The reason is that if the edge $s_i \to t_i, s_{i+1} \to t_{i+1}$ is present in $\mathcal{P}'$ then the same edge is contained in $\mathcal{P}$, that $t_i\sigma \xrightarrow{\mathcal{Q}'}{}^*_{\mathcal{R}'} s_{i+1}\sigma$ implies $t_i\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} s_{i+1}\sigma$ by Lemma 2.4, and that $s_i\sigma \in \mathit{NF}(\mathcal{Q}')$ implies $s_i\sigma \in \mathit{NF}(\mathcal{Q})$. $\square$

A.3. Proofs of Chapter 3

Proof of Theorem 3.3. Completeness follows from Lemma 2.17, since $\mathcal{P} \cap \mathcal{P}' \subseteq \mathcal{P}$. *Proc* is sound because we only drop those edges $(s \to t, u \to v)$ in $\mathcal{P}$ which cannot form a chain. Hence, whenever $s_1 \to t_1, s_2 \to t_2, \dots$ is an infinite (minimal) $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain then it is also an infinite (minimal) $(\mathcal{P}', \mathcal{Q}, \mathcal{R})$-chain and thus, an infinite (minimal) $(\mathcal{P} \cap \mathcal{P}', \mathcal{Q}, \mathcal{R})$-chain. $\square$

Proof of Theorem 3.4. Completeness follows from Lemma 2.17, since $\mathcal{P}_i \subseteq \mathcal{P}$ for all i. *Proc* is sound since after a finite number of nodes in the beginning, any infinite (minimal) $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain only contains nodes and edges from a single SCC. Hence, there also is an infinite (minimal) $(\mathcal{P}_i, \mathcal{Q}, \mathcal{R})$-chain for some $\mathcal{P}_i$. $\square$

Proof of Lemma 3.8. Let $t\sigma \xrightarrow{\mathcal{Q}}_{\mathcal{R},p_1} u_1 \xrightarrow{\mathcal{Q}}_{\mathcal{R},p_2} \dots \xrightarrow{\mathcal{Q}}_{\mathcal{R},p_n} u_n = u$ be a reduction such that $\mathcal{S}\sigma \subseteq \mathit{NF}(\mathcal{Q})$. Let $\{q_1, \dots, q_m\}$ be the set of positions where *ECap* replaces the subterms of t by corresponding fresh variables $x_1, \dots, x_m$. By the definition of *Cap* and by the definition of an estimated *Cap* function for each p_i there is a higher position $q_j \leq p_i$. Hence, u can at most differ from $t\sigma$ on positions below a q_j. We define μ to be like σ but on the variables $x_1, \dots, x_n$ we define $\mu(x_i) = u|_{q_i}$. Then by construction $\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)\mu = u$ and μ and σ differ only on the fresh variables. $\square$

Proof of Theorem 3.10. Suppose $s \to t, u \to v$ is a $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain, i.e., there is a substitution σ such that $s\sigma$ and $u\sigma$ are $\mathcal{Q}$-normal forms and $t\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} u\sigma$. From the latter we conclude by Lemma 3.8 that $u\sigma = \mathit{ECap}^{\{s,u\}}_{\mathcal{R},\mathcal{Q}}(t)\mu$ for some substitution μ that differs from σ only on the fresh variables that are introduced by *ECap*. Hence, u and $\mathit{ECap}^{\{s,u\}}_{\mathcal{R},\mathcal{Q}}(t)$ are unifiable by μ where $s\mu$ and $u\mu$ are in $\mathcal{Q}$-normal form. But then then $s\delta$ and $u\delta$ are obviously also $\mathcal{Q}$-normal forms where δ is the mgu of u and $\mathit{ECap}^{\{s,u\}}_{\mathcal{R},\mathcal{Q}}(t)$. $\square$

Proof of Lemma 3.12. We prove by structural induction that for every term *ICap* replaces subterms by fresh variables at higher positions than *Cap*. As the generalized TRS $\mathcal{R}$ and the sets $\mathcal{Q}$ and $\mathcal{S}$ are fixed throughout this proof, we write $\mathit{ICap}(t)$ instead of $\mathit{ICap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$ and similarly $\mathit{Cap}(t)$ instead of $\mathit{Cap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$.

If $\mathit{ICap}(t)$ is a fresh variable then there is nothing to prove. Otherwise, if t is a variable x and $\mathit{ICap}(x) = x$ then $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$ and x occurs in $\mathcal{S}$. If σ is a substitution with $\mathcal{S}\sigma \subseteq \mathit{NF}(\mathcal{Q})$ then $x\sigma$ is a $\mathcal{R}$-normal form and hence $x\sigma$ cannot be reduced by $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$. Thus, $\mathit{Cap}(x) = x$.

Finally, if $t = f(t_1, \dots, t_n)$ and $\mathit{ICap}(t) = f(\mathit{ICap}(t_1), \dots, \mathit{ICap}(t_n))$ then by induction we know that in every $\mathit{ICap}(t_i)$ the fresh variables have been introduced at higher positions than in $\mathit{Cap}(t_i)$. Hence, if in every reduction of $t\sigma$ with $\mathcal{S}\sigma \subseteq \mathit{NF}(\mathcal{Q})$ there is no reduction at root position, then by Definition 3.7 $\mathit{Cap}(t) = f(\mathit{Cap}(t_1), \dots, \mathit{Cap}(t_n))$ and we are done. Otherwise, there is a substitution σ with $\mathcal{S}\sigma \subseteq \mathit{NF}(\mathcal{Q})$ with a reduction of the form $t\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} f(s_1, \dots, s_n) = \ell\tau \xrightarrow{\mathcal{Q}}_{\mathcal{R}} r\tau = u$ where $\ell \to r \in \mathcal{R}$ and the last reduction step is the first reduction step at the root position. Hence, every direct subterm of $\ell\tau$ is in $\mathcal{Q}$-normal form. Moreover, $t_i\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} s_i$ and by Lemma 3.8 we know that every $s_i = \mathit{ICap}(t_i)\mu$ for a substitution μ that differs from σ only on the fresh variables that have been introduced by *ICap*. W.l.o.g. we assume that μ is equal to τ on the variables of the rule $\ell \to r$. Thus, $f(\mathit{ICap}(t_1), \ldots, \mathit{ICap}(t_n))\mu = f(\mathit{ICap}(t_1)\mu, \ldots, \mathit{ICap}(t_n)\mu) = f(s_1, \ldots, s_n) = \ell\tau = \ell\mu$ shows that $f(\mathit{ICap}(t_1), \ldots, \mathit{ICap}(t_n))$ and ℓ are unifiable by μ. As μ is identical to τ on the variables of ℓ and as μ is identical to σ on all variables but those that have been introduced by *ICap* we know that all terms in $\mathcal{S}\mu \cup \{\ell\mu|_1, \ldots, \ell\mu|_n\}$ are in $\mathcal{Q}$-normal form. Thus, also for the mgu δ of $f(\mathit{ICap}(t_1), \ldots, \mathit{ICap}(t_n))$ and ℓ all terms in $\mathcal{S}\delta \cup \{\ell\delta|_1, \ldots, \ell\delta|_n\}$ are in $\mathcal{Q}$-normal form. This finally proves that $\mathit{ICap}(t)$ is a fresh variable in contradiction to the assumption. □

Proof of Theorem 3.14. Completeness follows from Lemma 2.17. For soundness, we prove that under the conditions of the first case in Theorem 3.14, every minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain also results in a minimal $(\mathcal{P}, \mathit{lhs}(\mathcal{R}), \mathcal{R})$-chain, i.e., an innermost chain.

If $s_1 \to t_1, s_2 \to t_2, \ldots$ is a minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain then there is a substitution σ such that we have the following conditions for all i:

(a) $t_i\sigma \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} s_{i+1}\sigma$

(b) $s_i\sigma$ is in $\mathcal{Q}$-normal form

(c) $t_i\sigma$ is terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$

By (a) and (c), $\sigma(x)$ is terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ for all $x \in \mathcal{V}(s_2) \cup \mathcal{V}(s_3) \cup \ldots$ Since $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ is locally confluent on these terms, every $\sigma(x)$ has a unique normal form $\sigma(x)\!\downarrow$ w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by Newman's lemma. Let σ' be the substitution with $\sigma'(x) = \sigma(x)\!\downarrow$ for all $x \in \mathcal{V}(s_2) \cup \mathcal{V}(s_3) \cup \ldots$ and $\sigma'(x) = \sigma(x)$ otherwise. For all $i > 1$ we obtain:

(i) For all terms t we have $t\sigma \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} t\sigma'$.

(ii) If non-variable subterms of s_i do not unify with left-hand sides of rules from $\mathcal{R}$, then $s_i\sigma'$ is a normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$.

(iii) A term is an $\mathcal{R}$-normal form iff it is a normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$.

The observations (i) and (ii) are obvious. For (iii), the "only if" direction follows from $\overset{\mathcal{Q}}{\to}_{\mathcal{R}} \subseteq \to_{\mathcal{R}}$ (by Lemma 2.4). For the "if" direction, let t be a normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ and assume that t contains $\mathcal{R}$-redexes. Let t' be an "innermost" $\mathcal{R}$-redex, i.e., all proper subterms of t' are in $\mathcal{R}$-normal form. Since $\mathit{NF}(\mathcal{R}) \subseteq \mathit{NF}(\mathcal{Q})$, they are also in $\mathcal{Q}$-normal form. But then t' is also a redex w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$. This contradicts the assumption that t is a normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$.

Now we show that $s_2 \to t_2, s_3 \to t_3, \ldots$ is also a minimal $(\mathcal{P}, \mathit{lhs}(\mathcal{R}), \mathcal{R})$-chain. To this end, we use the substitution σ' instead of σ. For all $i > 1$ we have to prove:

(a′) $t_i\sigma' \overset{\mathrm{i}}{\to}_{\mathcal{R}}{}^{*} s_{i+1}\sigma'$

(b′) $s_i\sigma'$ is in normal form w.r.t. $\mathcal{R}$

(c′) $t_i\sigma'$ is innermost terminating

For (a′), note that $t_i\sigma \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} s_{i+1}\sigma \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} s_{i+1}\sigma'$ by (a) and (i), where $s_{i+1}\sigma'$ is a normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by (ii). Moreover, since $t_i\sigma$ is terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ and since $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ is locally confluent on $\mathcal{Q}$-terminating terms w.r.t. $\mathcal{R}$, $s_{i+1}\sigma'$ is the *unique* normal form of $t_i\sigma$ w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by Newman's lemma. Since $t_i\sigma \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} t_i\sigma'$ by (i), $t_i\sigma'$ is terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by (c) and since $\overset{\mathrm{i}}{\to}_{\mathcal{R}} \subseteq \overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by Lemma 2.4, $t_i\sigma'$ is also innermost terminating. Let w be a normal form of $t_i\sigma'$ w.r.t. $\overset{\mathsf{i}}{\to}_{\mathcal{R}}$. As $t_i\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} w$ and as w is also a normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by (iii), w must be the unique normal form $s_{i+1}\sigma'$. Hence, $t_i\sigma' \overset{\mathsf{i}}{\to}{}^*_{\mathcal{R}} s_{i+1}\sigma'$.

For (b$'$), $s_i\sigma'$ is a normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by (ii). Thus, (iii) implies that it is also a normal form w.r.t. $\mathcal{R}$.

For (c$'$), we have $t_i\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} t_i\sigma'$ by (i). Thus, $t_i\sigma'$ is terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by (c). Hence, $t_i\sigma'$ is also terminating w.r.t. $\overset{\mathsf{i}}{\to}_{\mathcal{R}}$ since $\overset{\mathsf{i}}{\to}_{\mathcal{R}} \subseteq \overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by Lemma 2.4. ☐

Proof of Lemma 3.19. Let $t \overset{\mathcal{Q}}{\to}_{\mathcal{R}} t_i$ for $i \in \{1,2\}$ with $t_1 \neq t_2$. As $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ is closed under contexts we only have to look at the case where $t = \ell\sigma \overset{\mathcal{Q}}{\to}_{\mathcal{R}} r\sigma = t_1$. If the reduction from $\ell\sigma$ to t_2 gives rive to a critical pair then we are done since $t_1 \neq t_2$ is a contradiction to the requirement that there are only trivial critical pairs. In the other case the reduction to t_2 is completely inside σ. Hence, there is some variable x at a position of ℓ such that $\sigma(x) \overset{\mathcal{Q}}{\to}_{\mathcal{R}} s$ and $t_2 = \ell\sigma[s]_p$. We define δ to be like σ on all variables except for x and $\delta(x) = s$. Then, clearly $t_2 \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} \ell\delta$ and $t_1 = r\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} r\delta$.

Unfortunately, we cannot guarantee $\ell\delta \overset{\mathcal{Q}}{\to}_{\mathcal{R}} r\delta$ as the subterms of $\ell\delta$ are not necessarily in $\mathcal{Q}$-normal form. But as t is terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ we know that for each $y \in \mathcal{V}(\ell) \supseteq \mathcal{V}(r)$ the term $y\delta$ can be rewritten to some normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$. Let μ be the corresponding substitution, i.e., $\mu(y)$ is defined as a normal form of $y\delta$ for each y. Then indeed we obtain $t_1 \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} r\delta \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} r\mu$ and $t_2 \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} \ell\delta \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} \ell\mu \overset{\mathcal{Q}}{\to}_{\mathcal{R}} r\mu$. For this last reduction step we show for every position p of ℓ that $\ell\mu \overset{\mathcal{Q}}{\to}_{\mathcal{R},\geqslant p} r\mu$ or $\ell\mu|_p \in \mathit{NF}(\mathcal{Q})$. Since, obviously $\ell\mu \notin \mathit{NF}(\mathcal{R}) \supseteq \mathit{NF}(\mathcal{Q})$ this will indeed prove the desired reduction $\ell\mu \overset{\mathcal{Q}}{\to}_{\mathcal{R}} r\mu$.

First note that there can be no redex in μ since $\mathit{NF}(\mathcal{R}) \subseteq \mathit{NF}(\mathcal{Q})$ implies that each normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ is also an $\mathcal{R}$-normal form. Hence, we only have to consider the case where $\ell|_p$ is not a variable. By induction there already is a reduction $\ell\mu \overset{\mathcal{Q}}{\to}_{\mathcal{R},p'} r\mu$ for some p' with $p > p'$ – then there is nothing to show – or every proper subterm of $\ell\mu|_p$ is in $\mathcal{Q}$-normal form. We consider two cases. If a reduction at position p is possible with $\mathcal{R}$ then this gives rise to a critical pair. Since all critical pairs are trivial, this reduction must result in $r\mu$ and we are done as all proper subterms of the redex are in $\mathcal{Q}$-normal form. In the other case no term of $\mathcal{Q}$ can match $\ell\mu|_p$ due to $\mathit{NF}(\mathcal{R}) \subseteq \mathit{NF}(\mathcal{Q})$, and since all proper subterms are in $\mathcal{Q}$-normal form, we have proven that $\ell\mu|_p \in \mathit{NF}(\mathcal{Q})$. ☐

Proof of Corollary 3.23. $\mathcal{R}$ terminates if the DP problem $(\mathit{DP}(\mathcal{R}), \varnothing, \mathcal{R}, \mathbf{m})$ is finite by Theorem 2.11. For overlay systems, no non-variable subterms of left-hand sides from $\mathit{DP}(\mathcal{R})$ unify with left-hand sides from $\mathcal{R}$. Hence, by using Theorem 3.14, it is sufficient if the DP problem $(\mathit{DP}(\mathcal{R}), \mathit{lhs}(\mathcal{R}), \mathcal{R}, \mathbf{m})$ is finite. This follows from innermost termination (i.e., $\mathit{lhs}(\mathcal{R})$-termination) of $\mathcal{R}$ by Theorem 2.11. ☐

Proof of Theorem 3.25. Completeness follows from Lemma 2.17 and soundness can be proven as follows: whenever the sequence $s \to t, u \to v$ occurs in an infinite chain, i.e., $t\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} u\sigma$ where $s\sigma \in \mathit{NF}(\mathcal{Q})$, then by Definition 3.24 all rules in this reduction are from $\mathcal{U}^{\{s\}}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{U}(\mathcal{P},\mathcal{Q},\mathcal{R}) \subseteq \mathcal{EU}(\mathcal{P},\mathcal{Q},\mathcal{R})$. Thus we obtain $t\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{EU}(\mathcal{P},\mathcal{Q},\mathcal{R})} u\sigma$ which shows that every infinite $(\mathcal{P},\mathcal{Q},\mathcal{R})$-chain is an infinite $(\mathcal{P},\mathcal{Q},\mathcal{EU}(\mathcal{P},\mathcal{Q},\mathcal{R}))$-chain. To carry over minimality of chains it suffices to state that $\overset{\mathcal{Q}}{\to}_{\mathcal{EU}(\mathcal{P},\mathcal{Q},\mathcal{R})}$ is a subrelation of $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by Lemma 2.4. ☐

Proof of Theorem 3.28. The theorem is a direct consequence of Theorem 4.30 where we choose the argument filter π defined by $\pi(f) = [1,\ldots,n]$ for all function symbols f with arity n.

☐

Proof of Theorem 3.32. Suppose $s \to t, u \to v$ is a $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain, i.e., there is a substitution σ such that $s\sigma$ and $u\sigma$ are $\mathcal{Q}$-normal forms and $t\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} u\sigma$. By Definition 3.24 we obtain $t\sigma \to^{*}_{\mathcal{R}''} u\sigma$ for the TRS $\mathcal{R}'' = \mathcal{EU}^{\{s,u\}}_{\mathcal{R},\mathcal{Q}}(t)$ and hence, $u\sigma \to^{*}_{\mathcal{R}'} t\sigma$. Using Lemma 3.8 we conclude $t\sigma = ECap^{\varnothing}_{\mathcal{R}',\varnothing}(u)\mu$ for some substitution μ that differs from σ only on the fresh variables that are introduced by $ECap$. Hence, t and $ECap^{\varnothing}_{\mathcal{R}',\varnothing}(u)$ are unifiable by μ where $s\mu$ and $u\mu$ are in $\mathcal{Q}$-normal form. But then obviously for the mgu δ of t and $ECap^{\varnothing}_{\mathcal{R}',\varnothing}(u)$ we also know that $s\delta$ and $u\delta$ are $\mathcal{Q}$-normal forms. □

Proof of Theorem 3.34. Let $\mathcal{Q}' = \mathcal{Q} \cap \mathcal{T}(\mathcal{F}, \mathcal{V})$. Obviously, by Lemma 2.4 every $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain is a $(\mathcal{P}, \mathcal{Q}', \mathcal{R})$-chain proving soundness. For completeness, we can just replace any symbol occurring in $\mathcal{Q} \setminus \mathcal{Q}'$ by a fresh symbol not occurring in $\mathcal{P} \cup \mathcal{R} \cup \mathcal{Q}$. Thus, after this replacement everything is in normal form w.r.t. $\mathcal{Q} \setminus \mathcal{Q}'$, and every term in $\mathcal{Q}'$-normal form is now also in $\mathcal{Q}$-normal form. As all symbols that are replaced in this way are not contained in $\mathcal{F}$, no reduction of $\mathcal{P} \cup \mathcal{R}$ is destroyed by the replacement. Hence, in this way we obtain from any infinite $\xrightarrow{\mathcal{Q}'}_{\mathcal{R}}$ reduction an infinite $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ reduction and from an infinite $(\mathcal{P}, \mathcal{Q}', \mathcal{R})$ chain one obtains an infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain. □

Proof of Theorem 3.36. Let $\mathcal{Q}' = \{q \in \mathcal{Q} \mid \mathit{root}(q) \in \mathcal{F}\}$. Completeness is proven in the same way as in the proof of Theorem 3.34. For soundness we again can use Lemma 2.4 to handle infinite, non-minimal chains. However, to prove that the processor is sound even for minimal chains we have to show that whenever an instantiated right-hand side $t\sigma$ of $\mathcal{P}$ is terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ then $t\sigma$ is also terminating w.r.t. $\xrightarrow{\mathcal{Q}'}_{\mathcal{R}}$. Here, we can restrict σ to be a $\mathcal{Q}$-normal substitution. To this end, we prove the following property for all terms $s \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and all $\mathcal{Q}$-normal substitutions δ with $\mathit{root}(\delta(x)) \notin \mathcal{F}$ for all $x \in \mathit{Dom}(\delta)$:

If $s\delta \xrightarrow{\mathcal{Q}'}_{\mathcal{R}} u$ then $s\delta \xrightarrow{\mathcal{Q}}_{\mathcal{R}} u$ and $u = s'\delta'$ for some $s' \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and some $\mathcal{Q}$-normal substitution δ' with $\mathit{root}(\delta'(x)) \notin \mathcal{F}$ for all $x \in \mathit{Dom}(\delta)$. $(\star)$

Using $(\star)$ obviously allows us to prove termination of $s\delta$ w.r.t. $\xrightarrow{\mathcal{Q}'}_{\mathcal{R}}$ if $s\delta$ is terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$. However, as we are interested in the termination of $t\sigma$ we first have to get the corresponding s and δ with $t = s\delta$. To this end we partition σ into $\mu\delta$ where $\mu(x)$ is like $\sigma(x)$ but where all maximal subterms v of $\sigma(x)$ which are rooted with a symbol that is not in $\mathcal{F}$ are replaced by fresh variables x_v, and where δ replaces all these variables by the corresponding subterm, i.e., $\delta(x_v) = v$. Then we choose $s = t\mu$ which is a term of $\mathcal{T}(\mathcal{F}, \mathcal{V})$ since t and all $\mu(x)$ are elements of $\mathcal{T}(\mathcal{F}, \mathcal{V})$ by construction. Moreover, as σ is a $\mathcal{Q}$-normal substitution this must be the case for δ, too. Hence, by $(\star)$ we can then conclude termination of $t\sigma$ w.r.t. $\xrightarrow{\mathcal{Q}'}_{\mathcal{R}}$ from termination of $t\sigma$ w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$.

To prove $(\star)$ we perform structural induction on s. As δ is a $\mathcal{Q}$-normal substitution and as $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$, s can be no variable. So, let $s = f(s_1, \ldots, s_n)$. If the reduction is below the root then one just has to apply the induction hypothesis. The only interesting case is a reduction at the root, i.e., $s\delta = \ell\rho \xrightarrow{\mathcal{Q}'}_{\mathcal{R}} r\rho = u$ for some rule $\ell \to r \in \mathcal{R}$. To prove $(\star)$ it suffices to show that $s\delta \xrightarrow{\mathcal{Q}}_{\mathcal{R}} u$. The reason is that this implies ρ being a $\mathcal{Q}$-normal substitution, and then we can partition $r\rho$ into $(r\mu')\delta'$ in the same way as we partitioned $t\sigma$ into $(t\mu)\delta$ to apply $(\star)$. To prove $s\delta \xrightarrow{\mathcal{Q}}_{\mathcal{R}} u$ we show that all direct subterms of $s\delta$ are in $\mathcal{Q}$-normal form. Suppose $s_i\delta$ has a $\mathcal{Q}$-redex, i.e., $s_i\delta|_p = q\tau$ for some position p and some substitution τ. First note that $s_i\delta|_p$ cannot completely be in δ as δ is a $\mathcal{Q}$-normal substitution, so p points to a non-variable position of s_i. Note that then $\mathit{root}(s_i|_p) = \mathit{root}(q) \in \mathcal{F}$ since $s_i \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. Hence, by construction $q \in \mathcal{Q}'$ which contradicts $s\delta \xrightarrow{\mathcal{Q}'}_{\mathcal{R}} u$. □

A.4. Proofs of Chapter 4

Proof of Theorem 4.2. Completeness follows from Lemma 2.17 and soundness is proved as in [AG00, Theorems 7 and 11].

Let $s_1 \to t_1, s_2 \to t_2, \ldots$ be an infinite (minimal) $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain. We distinguish two cases. If there is some $n \in \mathbb{N}$ such that for every $i \geq n$ the pair $s_i \to t_i$ is contained in $\mathcal{P} \setminus \succ_\pi$ then $s_n \to t_n, s_{n+1} \to t_{n+1}, \ldots$ clearly is an infinite (minimal) $(\mathcal{P} \setminus \succ_\pi, \mathcal{Q}, \mathcal{R}, f)$ chain and we are done.

Otherwise, there is some $s \to t \in \succ_\pi$ which occurs infinitely often in the chain. As $s_1 \to t_1, s_2 \to t_2, \ldots$ is a chain there must a be a substitution σ such that $t_i\sigma \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} s_{i+1}\sigma$. As $\succsim_\pi$ contains $\mathcal{R}$ and as $\succsim_\pi$ is stable and monotonic we obtain $t_i\sigma \succsim_\pi s_{i+1}\sigma$. Moreover, as every pair in $\mathcal{P}$ is oriented with $\succsim_\pi$ or with $\succ_\pi$ and as both these relations are stable we obtain $s_1\sigma \mathrel{\overset{\succ}{\scriptstyle(\sim)}_{\pi}} t_1\sigma \succsim_\pi s_2\sigma \mathrel{\overset{\succ}{\scriptstyle(\sim)}_{\pi}} t_2\sigma \ldots$. As $s \to t$ occurs infinitely often in the chain and as $\succsim_\pi$ and $\succ_\pi$ are compatible this shows that $s_1\sigma$ starts an infinite decreasing sequence w.r.t. $\succ_\pi$. This is in contradiction to the well-foundedness of $\succ_\pi$. □

Proof of Lemma 4.7. Let $M = \{t_0, \ldots, t_n\}$ with $t_0 < \cdots < t_n$. Since the term $\mathit{Comp}(M) = \mathsf{c}(t_0, \mathsf{c}(t_2 \ldots \mathsf{c}(t_n, \bot) \ldots))$ is in $\mathcal{Q}$-normal form, we obtain for every i the reduction $\mathit{Comp}(M) \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{C}_\varepsilon} \mathsf{c}(t_i, \mathsf{c}(t_{i+1} \ldots \mathsf{c}(t_n, \bot) \ldots)) = s_i$ by applying i-times the second rule $\mathsf{c}(x, y) \to y$ of $\mathcal{C}_\varepsilon$. Each s_i must be in $\mathcal{Q}$-normal form and we can use the first rule $\mathsf{c}(x, y) \to x$ of $\mathcal{C}_\varepsilon$ to finally obtain $\mathit{Comp}(M) \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{C}_\varepsilon} s_i \overset{\mathcal{Q}}{\to}_{\mathcal{C}_\varepsilon} t_i$. □

Proof of Lemma 4.10. As first step we prove for every set $\mathcal{S}'$ of terms that if $\mathit{Cap}^{\mathcal{S}'}_{\mathcal{R},\mathcal{Q}}(t) = f(s_1, \ldots, s_n)$ then $t = f(t_1, \ldots, t_n)$ and $s_i = \mathit{Cap}^{\mathcal{S}'}_{\mathcal{R},\mathcal{Q}}(t_i)$. $(\star)$

If $\mathit{Cap}^{\mathcal{S}'}_{\mathcal{R},\mathcal{Q}}(t) = f(s_1, \ldots, s_n)$ then by Definition 3.7 obviously the term t can be no variable and moreover, t must be of the form $f(t_1, \ldots, t_n)$. As $\mathit{Cap}^{\mathcal{S}'}_{\mathcal{R},\mathcal{Q}}(t)$ is not a fresh variable for every substitution μ with $\mathcal{S}'\mu \subseteq \mathit{NF}(\mathcal{Q})$ the term $t\mu$ cannot be reduced at the root level. Thus, the only possible reductions of $t\mu$ are those that can be performed in the subterms $t_i\mu$. But also the converse result is valid: whenever we can reduce $t_i\mu$ then we can perform the same reduction in $t\mu$. By Definition 3.7 we conclude that for each position ip of t the subterm $t|_{ip}$ of t is replaced by a fresh variable iff the subterm $t_i|_p$ of t_i is replaced by a fresh variable when applying Cap. Thus, for each i we obtain $s_i = \mathit{Cap}^{\mathcal{S}'}_{\mathcal{R},\mathcal{Q}}(t_i)$ which finally proves $(\star)$.

We now prove the lemma. If $\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$ is a fresh variable x then we choose δ with $\delta(x) = \mathit{Cap}^{\mathcal{T}}_{\mathcal{R},\mathcal{Q}}(t\sigma)$. Then indeed we obtain $\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)\sigma\delta = x\sigma\delta = x\delta = \mathit{Cap}^{\mathcal{T}}_{\mathcal{R},\mathcal{Q}}(t\sigma)$ as desired.

If $\mathit{Cap}^{\mathcal{T}}_{\mathcal{R},\mathcal{Q}}(t\sigma)$ is a fresh variable then by Definition 3.7 there is a substitution μ such that $\mathcal{T}\mu \subseteq \mathit{NF}(\mathcal{Q})$ and $(t\sigma)\mu \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} \overset{\mathcal{Q}}{\to}_{\mathcal{R}} s$ where the last reduction is at the root position. We choose the substitution $\sigma\mu$ to show that $\mathit{Cap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$ must be a fresh variable, too. We obtain the same term $t(\sigma\mu)$ and $\mathcal{S}(\sigma\mu) = (\mathcal{S}\sigma)\mu \subseteq \mathcal{T}\mu \subseteq \mathit{NF}(\mathcal{Q})$ establishes the required normal form condition. Hence, for the remaining proof we can assume that neither $\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$ nor $\mathit{Cap}^{\mathcal{T}}_{\mathcal{R},\mathcal{Q}}(t\sigma)$ are fresh variables. We perform induction on t.

If $t = x$ then $\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) = x$. Hence, for every substitution μ with $\mathcal{S}\sigma\mu \subseteq \mathit{NF}(\mathcal{Q})$ the term $x\sigma\mu$ is in normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$. With the same argumentation as above we conclude that $\mathit{Cap}^{\mathcal{T}}_{\mathcal{R},\mathcal{Q}}(x\sigma)$ does not replace any subterm of $x\sigma$. Thus, $\mathit{Cap}^{\mathcal{T}}_{\mathcal{R},\mathcal{Q}}(t\sigma) = t\sigma = \mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)\sigma$ and we can choose δ to be the empty substitution.

Otherwise, $t = f(t_1, \ldots, t_n)$ and as $\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$ and $\mathit{Cap}^{\mathcal{T}}_{\mathcal{R},\mathcal{Q}}(t\sigma)$ are not fresh variables we know that $\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) = f(s_1, \ldots, s_n)$ and $\mathit{Cap}^{\mathcal{T}}_{\mathcal{R},\mathcal{Q}}(t\sigma) = f(\ldots)$. We define the function $ECap'$ which maps every t_i to s_i and every other term is mapped to a fresh variable. Then $ECap'$ must be an estimated Cap-function as otherwise $ECap$ would not estimate Cap. Thus, we can apply the induction hypothesis to obtain substitutions δ_i with $Cap^{\mathcal{T}}_{\mathcal{R},\mathcal{Q}}(t_i\sigma) = ECap'^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_i)\sigma\delta_i$. As each δ_i is only defined on the fresh variables that are introduced by $ECap'$ we can build a combined substitution δ which satisfies $Cap^{\mathcal{T}}_{\mathcal{R},\mathcal{Q}}(t_i\sigma) = ECap'^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_i)\sigma\delta$ for every i. We finally prove the lemma as follows.

$$\begin{array}{rcl}
Cap^{\mathcal{T}}_{\mathcal{R},\mathcal{Q}}(t\sigma) & = & Cap^{\mathcal{T}}_{\mathcal{R},\mathcal{Q}}(f(t_1\sigma,\ldots,t_n\sigma)) \\
\text{(by } (\star)\text{)} & = & f(Cap^{\mathcal{T}}_{\mathcal{R},\mathcal{Q}}(t_1\sigma),\ldots,Cap^{\mathcal{T}}_{\mathcal{R},\mathcal{Q}}(t_n\sigma)) \\
 & = & f(ECap'^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1)\sigma\delta,\ldots,ECap'^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n)\sigma\delta) \\
 & = & f(ECap'^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1),\ldots,ECap'^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n))\sigma\delta \\
 & = & f(s_1,\ldots,s_n)\sigma\delta \\
 & = & ECap^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)\sigma\delta
\end{array} \qquad \square$$

Proof of Lemma 4.11. Note that by construction of $\mathcal{S}_{all}$ and σ all terms in $\mathcal{S}_{all}\sigma$ are $\mathcal{Q}$-normal.

(i) We perform induction on $\xrightarrow{\mathcal{Q}}^{+}_{\mathcal{R}} \cup \rhd$. For a variable the result obviously holds, so let $t = f(t_1,\ldots,t_n)$. As $t \in NF(\mathcal{Q})$ implies $t_i \in NF(\mathcal{Q})$ for every i we know by the induction hypothesis that $\mathcal{I}(t_i) \in NF(\mathcal{Q})$. Then $t' = f(\mathcal{I}(t_1),\ldots,\mathcal{I}(t_n))$ must also be $\mathcal{Q}$-normal. The reason is that the only possible redex of t' w.r.t. $\mathcal{Q}$ can be at the root level. However, if $t' = q\mu$ for some $q \in \mathcal{Q}$ then t is also an instance of $\mathcal{Q}$ as $\mathcal{I}$ is injective and whenever a subterm of t is replaced by $\mathsf{c}(\ldots,\ldots)$ then this part can only be matched by a variable in q.

Using the induction hypothesis again we see that $\mathcal{I}(s) \in NF(\mathcal{Q})$ for all s with $t \xrightarrow{\mathcal{Q}}^{+}_{\mathcal{R}} s$ and $s \in NF(\mathcal{Q})$. Thus, the terms t' and $\mathsf{c}(t', Comp(Red(t)))$ are both $\mathcal{Q}$-normal. Here, we have used the observation that $Comp(M) \in NF(\mathcal{Q})$ whenever $M \subseteq NF(\mathcal{Q})$. Hence, regardless whether $\mathcal{I}(t) = t'$ or $\mathcal{I}(t) = \mathsf{c}(t', Comp(Red(t)))$ we obtain $\mathcal{I}(t) \in NF(\mathcal{Q})$.

(ii) We perform induction on t. If t is a variable x then by definition $\mathcal{I}(t\sigma) = \mathcal{I}(x\sigma) = x\mathcal{I}(\sigma) = t\mathcal{I}(\sigma)$. Otherwise, let $t = f(t_1,\ldots,t_n)$. Then we obtain $\mathcal{N}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_i) \subseteq \mathcal{N}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{N}$. Hence, by induction we conclude $\mathcal{I}(t_i\sigma) = t_i\mathcal{I}(\sigma)$. If $\mathcal{I}(t\sigma)$ is built by the second case we obtain $\mathcal{I}(t\sigma) = f(t_1\mathcal{I}(\sigma),\ldots,t_n\mathcal{I}(\sigma)) = t\mathcal{I}(\sigma)$. In the other case there must be a rule $\ell \to r \in \mathcal{R} \setminus \mathcal{N}$ such that $f(Cap^{\mathcal{S}_{all}\sigma}_{\mathcal{R},\mathcal{Q}}(t_1\sigma),\ldots,Cap^{\mathcal{S}_{all}\sigma}_{\mathcal{R},\mathcal{Q}}(t_n\sigma))$ unifies with ℓ by some mgu μ with $(\mathcal{S}_{all}\sigma \cup \{\ell|_1,\ldots,\ell|_n\})\mu \subseteq NF(\mathcal{Q})$. The requirement $\mathcal{S} \subseteq \mathcal{S}_{all}$ allows us to use Lemma 4.10 (where we choose $\mathcal{T} = \mathcal{S}_{all}\sigma$) and we obtain $Cap^{\mathcal{S}_{all}\sigma}_{\mathcal{R},\mathcal{Q}}(t_i\sigma) = ECap^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_i)\sigma\delta$ where δ only instantiates fresh variables introduced by Cap.

Now, we extend μ to behave like $\delta\mu$ on the fresh variables of $ECap$. In this way we do not change the set $(\mathcal{S}_{all}\sigma \cup \{\ell|_1,\ldots,\ell|_n\})\mu$ and obtain $ECap^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_i)\sigma\mu = ECap^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_i)\sigma\delta\mu = Cap^{\mathcal{S}_{all}\sigma}_{\mathcal{R},\mathcal{Q}}(t_i\sigma)\mu = \ell|_i\mu$. As ℓ and t are variable disjoint we know that $f(ECap^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1),\ldots,ECap^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n))$ and ℓ are unifiable by some mgu τ. By construction τ instantiates the terms in $\mathcal{S}_{all}$ less than $\sigma\mu$ does and τ instantiates the terms in $\{\ell|_1,\ldots,\ell|_n\}$ less than μ does. Therefore, we conclude $(\mathcal{S} \cup \{\ell|_1,\ldots,\ell|_n\})\tau \subseteq NF(\mathcal{Q})$ which further implies $\ell \to r \in \mathcal{N}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$ by Definition 4.5. This is a contradiction to $\ell \to r \notin \mathcal{N}$.

(iii) By (i) we know that $\mathcal{I}(t\sigma) \in \mathit{NF}(\mathcal{Q})$. We prove the result by induction on t. If t is a variable then we use (ii) with $\mathcal{S} = \varnothing$ to conclude $\mathcal{I}(t\sigma) = t\mathcal{I}(\sigma)$.

Otherwise, if $t = f(t_1, \ldots, t_n)$ then by the induction $\mathcal{I}(t_i\sigma) \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{C}_\varepsilon} t_i\mathcal{I}(\sigma) \in \mathit{NF}(\mathcal{Q})$ and hence, $f(\mathcal{I}(t_1\sigma), \ldots, \mathcal{I}(t_n\sigma)) \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{C}_\varepsilon} f(t_1\mathcal{I}(\sigma), \ldots, t_n\mathcal{I}(\sigma)) = t\mathcal{I}(\sigma)$. With the same argumentation as in (i) we conclude $t\mathcal{I}(\sigma) \in \mathit{NF}(\mathcal{Q})$.

Finally, if $\mathcal{I}(t\sigma)$ is built by the second case we are done. Otherwise, we first reduce $\mathcal{I}(t\sigma) = \mathsf{c}(f(\mathcal{I}(t_1\sigma), \ldots, \mathcal{I}(t_n\sigma)), \mathit{Comp}(\ldots))$ to the term $f(\mathcal{I}(t_1\sigma), \ldots, \mathcal{I}(t_n\sigma))$ in one step. Note that this first reduction step respects the evaluation strategy given by $\mathcal{Q}$ as $\mathcal{I}(t\sigma) \in \mathit{NF}(\mathcal{Q})$.

(iv) The proof is a consequence of the definitions of *Red* and *Inv*, and of Lemma 4.7. Using $\mathit{Inv}(\mathcal{I}(t))$ we conclude $\mathcal{I}(t) = \mathsf{c}(\ldots, \mathit{Comp}(\mathit{Red}(t))) \in \mathit{NF}(\mathcal{Q})$. And if $t \xrightarrow{\mathcal{Q}}{}^{+}_{\mathcal{R}} s$ with $s \in \mathit{NF}(\mathcal{Q})$ then $\mathcal{I}(s) \in \mathit{Red}(t) \cap \mathit{NF}(\mathcal{Q})$ due to (i). Thus, Lemma 4.7 yields $\mathcal{I}(t) \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{C}_\varepsilon} \mathsf{c}(\ldots, \mathcal{I}(s)) \xrightarrow{\mathcal{Q}}{}_{\mathcal{C}_\varepsilon} \mathcal{I}(s)$.

(v) Let $t = \ell\sigma \xrightarrow{\mathcal{Q}}{}_{\mathcal{R}} r\sigma = s$ for some rule $\ell = f(\ell_1, \ldots, \ell_n) \to r \in \mathcal{R}$. We first want to show that $\ell \to r \in \mathcal{N}$. Due to Definition 3.7 we easily obtain a substitution δ with $\mathit{Cap}^{\mathcal{S}_{\mathit{all}}\sigma}_{\mathcal{R},\mathcal{Q}}(\ell_i\sigma)\delta = \ell_i\sigma$: whenever *Cap* replaces a subterm by a fresh variable then δ replaces this fresh variable by the subterm. We rename the variables in $\ell \to r$ to fresh ones to obtain the rule $\ell' = f(\ell'_1, \ldots, \ell'_n) \to r'$ and we define δ on $\mathcal{V}(\ell')$ to be like σ is defined on $\mathcal{V}(\ell)$. In this way we achieve $f(\mathit{Cap}^{\mathcal{S}_{\mathit{all}}\sigma}_{\mathcal{R},\mathcal{Q}}(\ell_i\sigma), \ldots, \mathit{Cap}^{\mathcal{S}_{\mathit{all}}\sigma}_{\mathcal{R},\mathcal{Q}}(\ell_i\sigma))\delta = \ell'\delta$ which shows that these two terms are unifiable by some mgu μ. As δ is only defined on fresh variables and on the variables of ℓ' we conclude $(\mathcal{S}_{\mathit{all}}\sigma \cup \{\ell'_1, \ldots, \ell'_n\})\delta = \mathcal{S}_{\mathit{all}}\sigma \cup \{\ell_1\sigma, \ldots, \ell_n\sigma\} \subseteq \mathcal{S}_{\mathit{all}}\sigma \subseteq \mathit{NF}(\mathcal{Q})$. The set inclusions can be derived from the construction of $\mathcal{S}_{\mathit{all}}$ and σ. Hence, when using the mgu μ instead of δ we also obtain $(\mathcal{S}_{\mathit{all}}\sigma \cup \{\ell'_1, \ldots, \ell'_n\})\mu \subseteq \mathit{NF}(\mathcal{Q})$. Finally using the fact that $\mathcal{I}(t)$ is built by the second case we know that $\ell' \to r'$ and thus $\ell \to r$ is contained in $\mathcal{N}$.

Now by the closure properties of $\mathcal{N}$ we know that $\mathcal{N}^{\{\ell_1,\ldots,\ell_n\}}_{\mathcal{R},\mathcal{Q}}(r) \subseteq \mathcal{N}$. Moreover, by the construction of $\mathcal{S}_{\mathit{all}}$ we obtain $\{\ell_1, \ldots, \ell_n\} \subseteq \mathcal{S}_{\mathit{all}}$. Hence, (iii) and (ii) yield $\mathcal{I}(t) = \mathcal{I}(\ell\sigma) \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{C}_\varepsilon} \ell\mathcal{I}(\sigma) \xrightarrow{\mathcal{Q}}{}_{\mathcal{N}} r\mathcal{I}(\sigma) = \mathcal{I}(r\sigma)$. Note that really all reductions respect the evaluation strategy given by $\mathcal{Q}$. Finally, as $x\sigma \in \mathit{NF}(\mathcal{Q})$ for all $x \in \mathcal{V}(\ell) \supseteq \mathcal{V}(r)$ we know by (i) that $x\mathcal{I}(\sigma) \in \mathit{NF}(\mathcal{Q})$ for all $x \in \mathcal{V}(r)$. Together with $r\mathcal{I}(\sigma) = \mathcal{I}(r\sigma) = \mathcal{I}(s)$ this proves $\mathit{Inv}(\mathcal{I}(s))$ as every subterm of $\mathcal{I}(s)$ which is rooted by c is contained in $\mathcal{I}(\sigma)$.

(vi) We perform induction on first the length of the reduction and then on the structure of t. If $t = s$ then there is nothing to show. So, let $t \xrightarrow{\mathcal{Q}}{}_{\mathcal{R}} t' \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} s$. Obviously t cannot be a variable. If $\mathcal{I}(t)$ is built by the third case we use (iv). So, let $t = f(t_1, \ldots, t_n)$ where $\mathcal{I}(t) = f(\mathcal{I}(t_1), \ldots, \mathcal{I}(t_n))$. If we have a root reduction we use (v) to obtain $\mathcal{I}(t) \xrightarrow{\mathcal{Q}}{}^{+}_{\mathcal{N}\cup\mathcal{C}_\varepsilon} \mathcal{I}(t')$ and $\mathit{Inv}(\mathcal{I}(t'))$. Thus, by induction we conclude $\mathcal{I}(t') \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{N}\cup\mathcal{C}_\varepsilon} \mathcal{I}(s)$.

Otherwise, we know that for some i we have a reduction $t_i \xrightarrow{\mathcal{Q}}{}_{\mathcal{R}} t'_i$ and $t' = f(t_1, \ldots, t'_i, \ldots, t_n)$. W.l.o.g. in the reduction of t' to s we first reduce t'_i to a $\mathcal{Q}$-normal form s_i, i.e., $t' \xrightarrow{\mathcal{Q}}{}^{+}_{\mathcal{R}} f(t_1, \ldots, s_i, \ldots, t_n) = t'' \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} s$. We obtain $\mathcal{I}(t_i) \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{N}\cup\mathcal{C}_\varepsilon} \mathcal{I}(s_i)$ and thus, $\mathcal{I}(t) \to^{*}_{\mathcal{N}\cup\mathcal{C}_\varepsilon} f(\mathcal{I}(t_1), \ldots, \mathcal{I}(s_i), \ldots, \mathcal{I}(t_n)) = s'$.

Next, we show $\mathcal{I}(t'') = s'$, i.e., we have to prove that there is no rule $\ell \to r \in \mathcal{R} \setminus \mathcal{N}$ such that $f(\mathit{Cap}^{\mathcal{S}_{\mathit{all}}\sigma}_{\mathcal{R},\mathcal{Q}}(t_1), \ldots, \mathit{Cap}^{\mathcal{S}_{\mathit{all}}\sigma}_{\mathcal{R},\mathcal{Q}}(s_i), \ldots, \mathit{Cap}^{\mathcal{S}_{\mathit{all}}\sigma}_{\mathcal{R},\mathcal{Q}}(t_n))$ unifies with ℓ by an mgu μ such that $(\mathcal{S}_{\mathit{all}}\sigma \cup \{\ell|_1, \ldots, \ell|_n\})\mu \subseteq \mathit{NF}(\mathcal{Q})$. As $\mathcal{I}(t)$ is built by the second case it suffices to show that $\mathit{Cap}^{\mathcal{S}_{\mathit{all}}\sigma}_{\mathcal{R},\mathcal{Q}}(s_i)$ is an instance of $\mathit{Cap}^{\mathcal{S}_{\mathit{all}}\sigma}_{\mathcal{R},\mathcal{Q}}(t_i)$. However, as $\mathcal{S}_{all}\sigma \subseteq NF(\mathcal{Q})$ and $t_i \xrightarrow{\mathcal{Q}}{}^{+}_{\mathcal{R}} s_i$ it suffices to apply Lemma 3.8. Hence, we have constructed the reduction $\mathcal{I}(t) \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{N}\cup\mathcal{C}_\varepsilon} s' = \mathcal{I}(t'')$ but in order to apply the induction hypothesis on t'' we have to guarantee $Inv(\mathcal{I}(t''))$. For this aim we use $Inv(\mathcal{I}(t))$ to conclude $Inv(\mathcal{I}(t_j))$ for every $j \neq i$, and $Inv(\mathcal{I}(s_i))$ follows from $s_i \in NF(\mathcal{Q})$ and (i). Thus, $Inv(\mathcal{I}(t''))$ is proven and by the induction hypothesis we conclude $\mathcal{I}(t) \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{N}\cup\mathcal{C}_\varepsilon} \mathcal{I}(t'') \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{N}\cup\mathcal{C}_\varepsilon} \mathcal{I}(s)$.

(vii) To prove that termination is preserved requires some more additional notions and definitions which are given in Definition A.1. The main problem is that in an infinite derivation of $\mathcal{I}(t)$ w.r.t. $\mathcal{M} \cup \mathcal{C}_\varepsilon$ one can produce arbitrary terms u which are not of the form $\mathcal{I}(s)$ for some s. Then it is unclear how to mimic the reductions of such a term u.

To this end, in Lemma A.2 we first prove that every infinite reduction of $\mathcal{I}(t)$ one can reorder the reductions in such a way that only certain rewrite steps are performed (Lemma A.2 (iv) and Lemma A.2 (viii)). Then for these rewrite steps in Lemma A.3 it is shown that a certain structure is kept (all terms are of the form $t[\mathcal{I}(t|_{p_1})]_{p_1} \dots [\mathcal{I}(t|_{p_n})]_{p_n}$ for pairwise independent positions p_i) and that it is possible to map reduction of such a term w.r.t. $\mathcal{M}\cup\mathcal{C}_\varepsilon$ back to a reduction of t w.r.t. $\mathcal{R}$. Then finally in Lemma A.3 (iv) we conclude the desired result that $\mathcal{I}(t)$ is terminating w.r.t. $\mathcal{M} \cup \mathcal{C}_\varepsilon$ if t is terminating w.r.t. $\mathcal{R}$.

(viii) As $u_1 \to v_1, u_2 \to v_2, \dots$ is a minimal chain for every i we know that $v_i\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} u_{i+1}\sigma$ and $u_i\sigma \in NF(\mathcal{Q})$. Using the previous results we can now exchange $\mathcal{R}$ by $\mathcal{N} \cup \mathcal{C}_\varepsilon$ and σ by $\mathcal{I}(\sigma)$ as illustrated in Figure 4.8.

By construction of $\mathcal{N}$ we know that $\mathcal{N}^{\{u_i\}}_{\mathcal{R},\mathcal{Q}}(v_i) \subseteq \mathcal{N}$ and $u_i \in \mathcal{S}_{all}$ for every i. Hence, using *(ii)* we obtain $\mathcal{I}(v_i\sigma) = v_i\mathcal{I}(\sigma)$. Remember that σ instantiates every term by a $\mathcal{Q}$-normal form. Thus, in the same way as in the proof of (v) we obtain $Inv(\mathcal{I}(v_i\sigma))$ by applying (i). From (iii) we conclude $\mathcal{I}(u_i\sigma) \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{C}_\varepsilon} u_i\mathcal{I}(\sigma)$ and $u_i\mathcal{I}(\sigma) \in NF(\mathcal{Q})$. By (vi) we glue everything together and obtain

$$u_1\mathcal{I}(\sigma) \to_{\mathcal{P}} v_1\mathcal{I}(\sigma) = \mathcal{I}(v_1\sigma) \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{N}\cup\mathcal{C}_\varepsilon} \mathcal{I}(u_2\sigma) \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{C}_\varepsilon} u_2\mathcal{I}(\sigma) \to_{\mathcal{P}} v_2\mathcal{I}(\sigma) \dots$$

As $\mathcal{M} \supseteq \mathcal{N}$ we know by Lemma 2.4 that $u_1 \to v_1, u_2 \to v_2, \dots$ also is a $(\mathcal{P}, \mathcal{Q}, \mathcal{M} \cup \mathcal{C}_\varepsilon)$-chain.

Moreover, if $\mathcal{M} \subseteq \mathcal{R}$, $\mathcal{Q} = \varnothing$, and if $\mathcal{M}$ is left-linear then by (vii) we know that every $\mathcal{I}(v_i\sigma)$ is terminating w.r.t. $\mathcal{M} \cup \mathcal{C}_\varepsilon$ and we obtain a minimal chain. ☐

Definition A.1 (c-$\mathcal{D}, \overset{>}{\to}, \overset{<}{\to}, \overset{=}{\to}, \overset{\leq}{\to}, \overset{=!}{\to}$). *Let Σ be a signature, let $\mathsf{c} \notin \Sigma$ be a fresh binary symbol. We define the* c-depth *of a term $t \in \mathcal{T}(\Sigma \cup \{\mathsf{c}\}, \mathcal{V})$ as c-$\mathcal{D}(t)$ where*

- $\mathsf{c}\text{-}\mathcal{D}(x) = 0$
- $\mathsf{c}\text{-}\mathcal{D}(f(t_1, \dots, t_n)) = max\{\mathsf{c}\text{-}\mathcal{D}(t_1), \dots, \mathsf{c}\text{-}\mathcal{D}(t_n)\}$ *for* $f \in \Sigma$
- $\mathsf{c}\text{-}\mathcal{D}(\mathsf{c}(t_1, t_2)) = 1 + max\{\mathsf{c}\text{-}\mathcal{D}(t_1), \mathsf{c}\text{-}\mathcal{D}(t_2)\}$

Here, the maximum of the empty set is 0.

For any rewrite relation $\to$ we define $\overset{>}{\to}, \overset{<}{\to}, \overset{=}{\to}$ as follows:

- $t \overset{>}{\to} u$ *iff* $t \to_{piq} u$ *and* $t|_p = \mathsf{c}(t_1, t_2)$. *(Reduction strictly below a* c*)*

- *$t \overset{<}{\to} u$ iff $t \to_p u$ and $root(t|_q) \in \Sigma$ for all $q \leq p$. (Reduction above all c's)*
- *$t \overset{=}{\to} u$ iff $t \to_p u$ and $root(t|_p) = \mathsf{c}$ and $root(t|_q) \in \Sigma$ for all $q < p$. (Reduction at top-most c)*

We define $\overset{\leq}{\to}$ as $\overset{<}{\to} \cup \overset{=}{\to}$, and $t \overset{=!}{\to} u$ iff $t \overset{=}{\to}_p^+ u$ and $root(u|_p) \in \Sigma$.

Clearly, $\overset{>}{\to}$, $\overset{<}{\to}$, and $\overset{=}{\to}$ build a partition of $\to$. This allows us to perform case-distinctions on $\to$.

We now show that if $\mathcal{R}$ is left-linear then any infinite reduction of t w.r.t. $\mathcal{R} \cup \mathcal{C}_\varepsilon$ implies that there also is an infinite reduction of t only using $\overset{\leq}{\to}$ steps (i-iv). We further show that we also have an infinite reduction only using $\overset{=!}{\to}$ and $\overset{<}{\to}$ reductions (v-viii). And reductions of this kind will allow us later on to construct an infinite reduction from t from an infinite $\mathcal{I}(t)$ reduction.

Lemma A.2. *Let $\mathcal{R}$ be a left-linear TRS over Σ. Let $\mathsf{c} \notin \Sigma$ and let $\mathcal{C}_\varepsilon = \{\mathsf{c}(x, y) \to x, \mathsf{c}(x, y) \to y\}$. All terms are from $\mathcal{T}(\Sigma \cup \{\mathsf{c}\}, \mathcal{V})$ and all rewrite steps are w.r.t. $\mathcal{R} \cup \mathcal{C}_\varepsilon$.*

(i) If $t \to^* u$ then $\mathsf{c}\text{-}\mathcal{D}(t) \geq \mathsf{c}\text{-}\mathcal{D}(u)$.

(ii) If $t \overset{>}{\to}^* \overset{<}{\to} u$ then $t \overset{<}{\to} \overset{>}{\to}^* u$.

(iii) If $t \overset{>}{\to}^* \overset{=}{\to} u$ then $t \overset{=}{\to} \overset{\leq}{\to}^* \overset{>}{\to}^* u$.

(iv) If t is not terminating w.r.t. $\to$ then t is not terminating w.r.t. $\overset{\leq}{\to}$.

(v) If $t \overset{=}{\to}_p u \overset{<}{\to} s$ then $root(u|_p) \in \Sigma$ or $t \overset{<}{\to} \overset{=}{\to}^* s$.

(vi) If $t \overset{=}{\to}^n \overset{=!}{\to}^* u$ then $t \overset{=!}{\to}^* \overset{=}{\to}^k u$ where $k \leq n$.

(vii) If $t \overset{=}{\to}^* \overset{<}{\to} u$ then $t \overset{=!}{\to}^* \overset{<}{\to} \overset{=}{\to}^* u$.

(viii) If t is not terminating w.r.t. $\to$ then t is not terminating w.r.t. $\overset{<}{\to} \cup \overset{=!}{\to}$.

Proof. (i) It obviously suffices to consider one-step reductions, so let $t \to u$. We perform structural induction on t where the only interesting case is a reduction at the root where $t = \ell\sigma \to r\sigma = u$. But then $\mathsf{c}\text{-}\mathcal{D}(\ell\sigma) \geq max\{\mathsf{c}\text{-}\mathcal{D}(x\sigma) \mid x \in \mathcal{V}(\ell)\} \geq max\{\mathsf{c}\text{-}\mathcal{D}(x\sigma) \mid x \in \mathcal{V}(r)\} = \mathsf{c}\text{-}\mathcal{D}(r\sigma)$ since $\mathcal{V}(\ell) \subseteq \mathcal{V}(r)$ and since r does not contain the symbol c.

(ii) Let $t \overset{>}{\to}^* s \overset{<}{\to}_p u$. We perform induction on the position p. If $p = \varepsilon$ then $t = t[\mathsf{c}(v_1, w_1)]_{q_1} \dots [\mathsf{c}(v_n, w_n)]_{q_n}$ where the $t|_{q_i}$ are the maximal subterms of t which are rooted by c. Hence, $s = t[\mathsf{c}(v'_1, w'_1)]_{q_1} \dots [\mathsf{c}(v'_n, w'_n)]_{q_n}$ where $\mathsf{c}(v_i, w_i) \overset{>}{\to}^* \mathsf{c}(v'_i, w'_i)$. Let $s = \ell\sigma \overset{<}{\to} r\sigma = u$. As $\ell \to r$ must be from $\mathcal{R}$ we know that ℓ does not contain any c and that ℓ is linear. Hence, all terms $\mathsf{c}(v'_i, w'_i)$ must be part of the substitution. Due to linearity we can define δ to be like σ but all terms $\mathsf{c}(v'_i, w'_i)$ are replaced by $\mathsf{c}(v_i, w_i)$. Thus, $t = \ell\delta \overset{<}{\to} r\delta$. From our construction we obtain $x\delta \overset{>}{\to}^* x\sigma$ and hence $t \overset{<}{\to} r\delta \overset{>}{\to}^* r\sigma = u$.

If $p = ip'$ then $t = f(t_1, \dots, t_n)$ where $f \in \Sigma$. Thus, in the reduction from t to u there are only non-root steps involved. Then of course we can exchange all reduction on positions below i with reductions below $j \neq i$ such that the reductions below i will come first. Hence, we obtain $t = f(t_1, \dots, t_i, \dots, t_n) \xrightarrow{>}{}^*_{>i} \xrightarrow{<}_{ip'} f(t_1, \dots, u_i, \dots, t_n) \xrightarrow{>}{}^*_{>j} u$, where $t_i \xrightarrow{>}{}^* \xrightarrow{<}_{p'} u_i$. By induction we obtain $t_i \xrightarrow{<}\xrightarrow{>}{}^* u_i$ and hence $t \xrightarrow{<}\xrightarrow{>}{}^* f(t_1, \dots, u_i, \dots, t_n) \xrightarrow{>}{}^* u$.

(iii) Let $t \xrightarrow{>}{}^* \xrightarrow{=}_p u$. We perform induction on $\mathsf{c}\text{-}\mathcal{D}(t)$. First, we can shift all steps at independent positions of p to the back, i.e., $t \xrightarrow{>}{}^*_{>p} t[v]_p \xrightarrow{=}_p t[w]_p \xrightarrow{>}{}^* u$. Hence, $t|_p = \mathsf{c}(t_1, t_2) \xrightarrow{>}{}^* \mathsf{c}(t'_1, t'_2) = v \xrightarrow{=} t'_i = w$ for some $i \in \{1, 2\}$ with $t_i \to^* w$. Thus, $t \xrightarrow{=} t[t_i]_p \to^* t[w]_p \xrightarrow{>}{}^* u$. We now show by an inner induction on the reduction length that for every w' with $t_i \to^* w'$ there also is the reduction $t_i \xrightarrow{\leq}{}^* \xrightarrow{>}{}^* w'$ $(\star)$. This directly proves (iii) as then we have $t|_p \xrightarrow{=} t_i \xrightarrow{\leq}{}^* \xrightarrow{>}{}^* w$ and hence, $t \xrightarrow{=}\xrightarrow{\leq}{}^* \xrightarrow{>}{}^* t[w]_p \xrightarrow{>}{}^* u$.

Property $(\star)$ is obviously satisfied if $t_i = w'$, so let $t_i \to^* r' \to w'$. By the inner induction we obtain $t_i \xrightarrow{\leq}{}^* r \xrightarrow{>}{}^* r' \to w'$. We distinguish three cases: First, if $r' \xrightarrow{>} w'$ we are done. Second, if $r' \xrightarrow{<} w'$ then we use (ii) to obtain $r \xrightarrow{<}\xrightarrow{>}{}^* w'$ and are done. Finally, if $r' \xrightarrow{=} w'$ then note that $\mathsf{c}\text{-}\mathcal{D}(t) > \mathsf{c}\text{-}\mathcal{D}(t_i) \geq \mathsf{c}\text{-}\mathcal{D}(r)$ due to (i). Hence, we can use the outer induction hypothesis to obtain $r \xrightarrow{=}\xrightarrow{\leq}{}^* \xrightarrow{>}{}^* w'$ which finishes this case.

(iv) Let s be a minimal terminating subterm of t. As the only c-rules are the two rules of $\mathcal{C}_\varepsilon$ we know that $\mathit{root}(s) \in \Sigma$. Hence, all subterms of s with $\mathit{root}(s) = \mathsf{c}$ are terminating. As no rule introduces a c-symbol we know that in the infinite reduction of s there must be infinitely many $\xrightarrow{\leq}$-steps. Thus, $s \xrightarrow{>}{}^* \xrightarrow{\leq} s_1 \xrightarrow{>}{}^* \xrightarrow{\leq} s_2 \xrightarrow{>}{}^* \xrightarrow{\leq} \dots$. We define $v_0 = s$ and obtain from (ii) and (iii) some v_1 with $v_0 \xrightarrow{\leq}{}^+ v_1 \xrightarrow{>}{}^* s_1$. Hence, $v_1 \xrightarrow{>}{}^* s_1 \xrightarrow{>}{}^* \xrightarrow{\leq} s_2$. Again using (ii) and (iii) we obtain some v_2 with $v_1 \xrightarrow{\leq}{}^+ v_2 \xrightarrow{>}{}^* s_2$ and $v_2 \xrightarrow{>}{}^* s_2 \xrightarrow{>}{}^* \xrightarrow{\leq} s_3$. Continuing in this way we obtain the infinite reduction $s = v_0 \xrightarrow{\leq}{}^+ v_1 \xrightarrow{\leq}{}^+ v_2 \dots$ which proves that s is not terminating w.r.t. $\xrightarrow{\leq}$.

Now, if in t there is no c above s we can do the same reduction and obtain an infinite reduction of t w.r.t. $\xrightarrow{\leq}$. In the other case all reductions of s are $\xrightarrow{>}$ reductions. However, using some initial $\xrightarrow{=}$ reductions one can eliminate all c-symbols of t that are above s and then start the infinite $\xrightarrow{\leq}$ reduction of s.

(v) We perform induction on t. If $t = \mathsf{c}(t_1, t_2) \xrightarrow{=}_p u \xrightarrow{<} s$ then p must be ε and from $u \xrightarrow{<} s$ we know that $\mathit{root}(u) = \mathit{root}(u|_p) \in \Sigma$.

Otherwise we have $t = f(t_1, \dots, t_n) \xrightarrow{=}_{ip'} f(t_1, \dots, t'_i, \dots, t_n) = u \xrightarrow{<}_q s$ where $t_i \xrightarrow{=}_{p'} t'_i$. If ip' and q are independent we can easily exchange the two reductions and are done. If $q = iq'$ we know $t_i \xrightarrow{=}_{p'} t'_i \xrightarrow{<}_{q'} u_i$ and $u = f(t_1, \dots, u_i, \dots, t_n)$. Using induction yields $\mathit{root}(t'_i|_{p'}) \in \Sigma$ or $t_i \xrightarrow{<}\xrightarrow{=}{}^* u_i$. In the latter case we also have $t = f(t_1, \dots, t_i, \dots, t_n) \xrightarrow{<}\xrightarrow{=}{}^* f(t_1, \dots, u_i, \dots, t_n) = u$ and in the former case $\mathit{root}(t|_{ip'}) = \mathit{root}(t_i|_{p'}) \in \Sigma$. Hence, in both case we obtain our required result.

It remains the case $q = \varepsilon$ where $t \xrightarrow{=}_{>\varepsilon} u = \ell\sigma \xrightarrow{<} r\sigma = s$. Of course, as we have a $\xrightarrow{<}$-reduction the rule $\ell \to r$ must be from $\mathcal{R}$. If $\mathit{root}(u|_p) \in \Sigma$ then there is nothing to show. So, let $\mathit{root}(u|_p) \notin \Sigma$, i.e., $u|_p \in \mathcal{V}$ or $u|_p = \mathsf{c}(\dots)$. As c does not occur in $\mathcal{R}$ the term $u|_p$ must be part of σ, i.e., there is a variable x such that $u|_p$ is a subterm of $\sigma(x)$. We define the substitution δ to be like σ except that $\delta(x)$ differs from $\sigma(x)$ where the subterm $u|_p$ has been replaced by $t|_p$. As x occurs at most once in ℓ we know that $t = \ell\delta$. Finally, from the reduction $\delta(x) \overset{=}{\to} \sigma(x)$ we obtain $t = \ell\delta \overset{<}{\to} r\delta \overset{=}{\to}^* r\sigma = s$.

(vi) Let $t \overset{=}{\to}^n v \overset{=!}{\to}^m u$. We perform an outer induction on n and an inner reduction on m. If $n = 0$ or $m = 0$ then there is nothing to show.

Otherwise, let $t \overset{=}{\to}^{n-1} s \overset{=}{\to}_p \ell \overset{=!}{\to}_q w \overset{=!}{\to}^{m-1} u$. We consider four cases to proof $s \overset{=!}{\to}^* \overset{=}{\to}^? w$. If $p = q$ then by the definition of $\overset{=!}{\to}$ we also have $s \overset{=!}{\to} w$. If p and q are independent then we can exchange the two reductions between s and w and are done. If p is below q, i.e., $p = qiq'$, then $\mathit{root}(\ell|_q) = \mathit{root}(s|_q) = \mathsf{c}$. Hence, the reduction from s to ℓ was below a c which contradicts $s \overset{=}{\to}_p \ell$. Finally, if q is below p, i.e., $q = pip'$, then from $\ell \overset{=!}{\to}_q w$ we know that $\mathit{root}(\ell|_p) \in \Sigma$ and hence $s \overset{=!}{\to} \ell \overset{=!}{\to} w$.

At this point we have shown that there is some term v with $s \overset{=!}{\to}^* v \overset{=}{\to}^? w$. Now using the inner induction hypothesis yields $v \overset{=!}{\to}^* r \overset{=}{\to}^? u$. Finally, the outer induction hypothesis results in $t \overset{=!}{\to}^* \overset{=}{\to}^{k'} r$ with $k' \leq n - 1$. Hence, $t \overset{=!}{\to}^* \overset{=}{\to}^k u$ with $k' \leq k' + 1 = k$.

(vii) We perform an induction on the number of $\overset{=}{\to}$-reductions. If there is no $\overset{=}{\to}$-reduction we are done. Otherwise, let $t \overset{=}{\to}^n s \overset{=}{\to}_p v \overset{<}{\to} u$. We use (v) to obtain $s \overset{<}{\to} w \overset{=}{\to}^* u$ or $\mathit{root}(v|_p) \in \Sigma$. In the former case we just have to use induction to obtain $t \overset{=!}{\to}^* \overset{<}{\to} \overset{=}{\to}^* w \overset{=}{\to}^* u$ and are done. In the latter case we know by the definition of $\overset{=!}{\to}$ that $s \overset{=!}{\to} v$ and hence $t \overset{=}{\to}^n s \overset{=!}{\to} v \overset{<}{\to} u$. Using (vi) we obtain $t \overset{=!}{\to}^* w \overset{=}{\to}^k v \overset{<}{\to} u$ where $k \leq n$. Thus, by induction we conclude $t \overset{=!}{\to}^* w \overset{=!}{\to}^* \overset{<}{\to} \overset{=}{\to}^* u$.

(viii) The proof is analogous to the proof of (iv). Suppose, t is not terminating w.r.t. $\to$. Then by *(iv)* the term t is not terminating w.r.t. $\overset{\leq}{\to}$, too.

As $\overset{=}{\to} \subseteq \to_{\mathcal{C}_\varepsilon}$ we know that in an infinite reduction w.r.t. $\overset{\leq}{\to}$ there must be infinitely many $\overset{<}{\to}$-steps. Hence, $t \overset{=}{\to}^* \overset{<}{\to} s_1 \overset{=}{\to}^* \overset{<}{\to} s_2 \overset{=}{\to}^* \overset{<}{\to} \dots$. We define $v_0 = t$ and obtain from (vii) some v_1 with $v_0 \overset{=!}{\to}^* \overset{<}{\to} v_1 \overset{=}{\to}^* s_1$. Thus, $v_1 \overset{=}{\to}^* s_1 \overset{=}{\to}^* \overset{<}{\to} s_2$. Again using (vii) we obtain some v_2 with $v_1 \overset{=!}{\to}^* \overset{<}{\to} v_2 \overset{=}{\to}^* s_2$ and $v_2 \overset{=}{\to}^* s_2 \overset{=}{\to}^* \overset{<}{\to} s_3$. Continuing in this way we obtain the infinite reduction $t = v_0 \overset{=!}{\to}^* \overset{<}{\to} v_1 \overset{=!}{\to}^* \overset{<}{\to} v_2 \dots$ which proves that t is not terminating w.r.t. $\overset{<}{\to} \cup \overset{=!}{\to}$. □

Lemma A.3. *Let $\mathcal{M} \subseteq \mathcal{R}$ be some set of left-linear rules. We use t, s for terms of $\mathcal{T}(\Sigma, \mathcal{V})$. The reductions $\overset{<}{\to}, \overset{=!}{\to}$ are used w.r.t. $\to_{\mathcal{M} \cup \mathcal{C}_\varepsilon}$. The notation $t[\cdot]_{p_1} \dots [\cdot]_{p_n}$ always induces that the p_i are independent positions.*

(i) For every $u = t[\mathcal{I}(t|_{p_1})]_{p_1} \dots [\mathcal{I}(t|_{p_n})]_{p_n}$ there are positions q_i such that $u = t[\mathcal{I}(t|_{q_1})]_{q_1} \dots [\mathcal{I}(t|_{q_m})]_{q_m}]$, $\mathit{root}(\mathcal{I}(t|_{q_i})) = \mathsf{c}$, and for each q_i there is some p_j which is (non-strictly) above q_i.

(ii) *If* $t[\mathcal{I}(t|_{p_1})]_{p_1} \dots [\mathcal{I}(t|_{p_n})]_{p_n} \overset{\leq}{\to} u$ *then* $t \to_{\mathcal{R}} s$ *and* u *is a term of the form* $s[\mathcal{I}(s|_{q_1})]_{q_1} \dots [\mathcal{I}(s|_{q_m})]_{q_m}$.

(iii) *If* $t[\mathcal{I}(t|_{p_1})]_{p_1} \dots [\mathcal{I}(t|_{p_n})]_{p_n} \overset{=!}{\to} u$ *then* $t \to_{\mathcal{R}}^{*} s$ *and* u *is a term of the form* $s[\mathcal{I}(s|_{q_1})]_{q_1} \dots [\mathcal{I}(s|_{q_m})]_{q_m}$.

(iv) *If* t *is terminating w.r.t.* $\mathcal{R}$ *then* $\mathcal{I}(t)$ *is terminating w.r.t.* $\mathcal{M} \cup \mathcal{C}_{\varepsilon}$.

Proof. (i) We perform induction on t. If $n = 0$ there is nothing to show. If some $p_i = \varepsilon$ then $n = 1$ and we have $u = \mathcal{I}(t)$. If $\mathit{root}(u) = \mathsf{c}$ we are done.

If all p_j are below the root then t must be of the form $f(t_1, \dots, t_k)$, $u = f(u_1, \dots, u_k)$, and $p_1, \dots, p_n = p_{1,1}, \dots, p_{1,n_1}, \dots, p_{k,1}, \dots, p_{k,n_k}$ where each $p_{i,j} = ip'_{i,j}$. Hence, each $u_i = t_i[\mathcal{I}(t|_{p_{i,1}})]_{p'_{i,1}} \dots [\mathcal{I}(t|_{p_{i,n_i}})]_{p'_{i,n_i}} = t_i[\mathcal{I}(t_i|_{p'_{i,1}})]_{p'_{i,1}} \dots [\mathcal{I}(t_i|_{p'_{i,n_i}})]_{p'_{i,n_i}}$. Thus, we can apply the induction hypothesis to obtain $u_i = t_i[\mathcal{I}(t_i|_{q_{i,1}})]_{q_{i,1}} \dots [\mathcal{I}(t_i|_{q_{i,m_i}})]_{q_{i,m_i}}$ where all the roots of all $\mathcal{I}(t_i)|_{q_{i,j}}$ are c. Hence,

$$\begin{aligned} u &= f(\dots, u_i, \dots) \\ &= f(\dots, t_i[\mathcal{I}(t_i|_{q_{i,1}})]_{q_{i,1}} \dots [\mathcal{I}(t_i|_{q_{i,m_i}})]_{q_{i,m_i}}, \dots) \\ &= f(\dots, t_i[\mathcal{I}(t|_{iq_{i,1}})]_{q_{i,1}} \dots [\mathcal{I}(t|_{iq_{i,m_i}})]_{q_{i,m_i}}, \dots) \\ &= t \dots [\mathcal{I}(t|_{iq_{i,1}})]_{iq_{i,1}} \dots [\mathcal{I}(t|_{iq_{i,m_i}})]_{iq_{i,m_i}} \dots \end{aligned}$$

shows the required result.

In the other case, there is only one position $p = \varepsilon$ and thus, $u = \mathcal{I}(t)$. If $\mathit{root}(u) = \mathsf{c}$ or if $t = u$ is a variable, then we are directly done. Otherwise, $t = f(t_1, \dots, t_k)$ and $u = f(\mathcal{I}(t_1), \dots, \mathcal{I}(t_k)) = t[\mathcal{I}(t_1)]_1 \dots [\mathcal{I}(t_k)]_k$. Hence, it suffices to apply the previous case where all positions $1, \dots, k$ are below the root.

(ii) As we have a $\overset{\leq}{\to}$-reduction the rule used must be from $\mathcal{M}$ and hence, left-linear. We perform induction on the position of the reduction. For a reduction at the root we have $t[\mathcal{I}(t|_{p_1})]_{p_1} \dots [\mathcal{I}(t|_{p_n})]_{p_n} = \ell\sigma \overset{\leq}{\to} r\sigma = u$. Using (i) we can assume that the roots of all $\mathcal{I}(t|_{p_i})$ are c. Hence, all these parts must be matched from variables in ℓ. As ℓ is linear we can define δ to be like σ where all $\mathcal{I}(t|_{p_i})$ are replaced by $t|_{p_i}$. Thus $t = t[t|_{p_1}]_{p_1} \dots [t|_{p_n}]_{p_n} = \ell\delta \to_{\mathcal{M}} r\delta = s$. Moreover, by replacing in s those subterm $s|_{q_j}$ which correspond to a previously term $t|_{p_i}$ back to $\mathcal{I}(t|_{p_i}) = \mathcal{I}(s|_{q_j})$ we obtain u.

For a non-root reduction all p_j must be of the form ip'_j for some $1 \leq i \leq k$ and the left-hand side $w = t[\mathcal{I}(t|_{p_1})]_{p_1} \dots [\mathcal{I}(t|_{p_n})]_{p_n}$ is a term of the form $f(t_1 \dots, \dots, t_i[\mathcal{I}(t_i|_{p'_j})]_{p'_j} \dots, \dots, t_k \dots)$ with $t_i[\mathcal{I}(t_i|_{p'_j})]_{p'_j} \dots \overset{\leq}{\to} u_i$ and $u = w[u_i]_i$. Hence, by induction we obtain some s_i such that $t_i \to_{\mathcal{R}} s_i$ with $u_i = s_i[\mathcal{I}(s_i|_{q_1})]_{q_1} \dots [\mathcal{I}(s_i|_{q_m})]_{q_m}$. We define $s = f(t_1, \dots, s_i, \dots, t_k)$. Then $t \to_{\mathcal{R}} s$ and

$$\begin{aligned} u &= w[u_i]_i \\ &= f(t_1[\mathcal{I}(t_1|_{p'_1})]_{p'_1} \dots, \dots, u_i, \dots, t_k[\mathcal{I}(t_k|_{p'_j})]_{p'_j} \dots) \\ &= s[\mathcal{I}(t_1|_{p'_1})]_{1p'_1} \dots [\mathcal{I}(s_i|_{q_1})]_{iq_1} \dots [\mathcal{I}(s_i|_{q_m})]_{iq_m} \dots [\mathcal{I}(t_k|_{p'_j})]_{kp'_j} \dots \\ &= s[\mathcal{I}(s|_{1p'_1})]_{1p'_1} \dots [\mathcal{I}(s|_{iq_1})]_{iq_1} \dots [\mathcal{I}(s|_{iq_m})]_{iq_m} \dots [\mathcal{I}(s|_{kp'_j})]_{kp'_j} \dots \end{aligned}$$

(iii) We perform induction over first the number of $\overset{=}{\rightarrow}$-steps in the $\overset{=!}{\rightarrow}$-reduction and then on the term t. We first consider the case of a root reduction. Then the root of $w = t[\mathcal{I}(t|_{p_1})]_{p_1} \dots$ must be c. Thus, $w = \mathcal{I}(t)$, $t = f(t_1, \dots, t_k)$ and $\mathcal{I}(t) = \mathsf{c}(x_t, \mathit{Comp}(\{f(\mathcal{I}(t_1), \dots, \mathcal{I}(t_k))\} \cup \mathit{Red}(t)))$. If $\mathcal{I}(t) \overset{=!}{\rightarrow} f(\mathcal{I}(t_1), \dots, \mathcal{I}(t_k)) = u$ then we choose $t = s$ and are done as $u = t[\mathcal{I}(t|_1)]_1 \dots [\mathcal{I}(t|_k)]_k$. Otherwise, if $\mathcal{I}(t) \overset{=!}{\rightarrow} \mathcal{I}(s)$ for some $s \in \mathit{Red}(t)$ then we know that $t \rightarrow_{\mathcal{R}}{}^{+} s$ and are done as $\mathcal{I}(s) = s[\mathcal{I}(s|_\varepsilon)]_\varepsilon$. The only missing case is $\mathcal{I}(t) \overset{=}{\rightarrow}{}_{\varepsilon}^{+} \mathcal{I}(r) \overset{=!}{\rightarrow} u$ where $r \in \mathit{Red}(t)$. Again we know that $t \rightarrow_{\mathcal{R}}{}^{+} r$ and can use the induction to obtain some s with $r \rightarrow_{\mathcal{R}}{}^{*} s$ and $u = s[\mathcal{I}(s|_{q_1})]_{q_1} \dots [\mathcal{I}(s|_{q_m})]_{q_m}$.

We can deal with non-root reductions in completely the same way as in (ii).

(iv) Suppose, $\mathcal{I}(t)$ is not terminating w.r.t. $\mathcal{M} \cup \mathcal{C}_\varepsilon$. Then by Lemma A.2 (iv,viii) we know that $\mathcal{I}(t)$ is not terminating w.r.t. $\overset{<}{\rightarrow} \cup \overset{=!}{\rightarrow}$. Of course, as $\overset{=!}{\rightarrow}$ is terminating we must have infinitely many $\overset{<}{\rightarrow}$-steps. But as $\mathcal{I}(t) = t[\mathcal{I}(t|_\varepsilon)]_\varepsilon$ we can use (ii, iii) to obtain an infinite $\mathcal{R}$-reduction of t. □

Proof of Theorem 4.12. Just use Lemma 4.11 (viii). □

Proof of Theorem 4.18. Completeness is a consequence of Lemma 2.17. For the soundness we just have to adapt the proof of Theorem 4.2 and integrate Lemma 4.11. Let $s_1 \rightarrow t_1, s_2 \rightarrow t_2, \dots$ be an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain. We distinguish two cases. If there is some $n \in \mathbb{N}$ such that for every $i \geq n$ the pair $s_i \rightarrow t_i$ is contained in $\mathcal{P} \setminus \succ_\pi$ then $s_n \rightarrow t_n, s_{n+1} \rightarrow t_{n+1}, \dots$ clearly is an infinite minimal $(\mathcal{P} \setminus \succ_\pi, \mathcal{Q}, \mathcal{R}, f)$ chain. Then we are immediately done.

Otherwise, there is some $s_i \rightarrow t_i \in \succ_\pi$ which occurs infinitely often in the chain. By Lemma 4.11 (viii) $s_1 \rightarrow t_1, s_2 \rightarrow t_2, \dots$ is also an infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{N}(\mathcal{P}, \mathcal{Q}, \mathcal{R}) \cup \mathcal{C}_\varepsilon)$-chain. We extend π by defining $\pi(\mathsf{c}) = [1, 2]$. Then in the same way as in the proof of Theorem 4.2 the constraints of the theorem yield an infinite decrease w.r.t. $\succ_\pi$. This is in contradiction to the well-foundedness of $\succ_\pi$. □

Proof of Theorem 4.20. Completeness is a consequence of Lemma 2.17. For soundness we consider an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain $s_1 \rightarrow t_1, s_2 \rightarrow t_2, \dots$ where σ is the substitution with $t_i\sigma \overset{\mathcal{Q}}{\rightarrow}{}_{\mathcal{R}}^{*} s_{i+1}\sigma$. We distinguish two cases. If there is some $n \in \mathbb{N}$ such that for every $i \geq n$ in the reductions $t_i\sigma \overset{\mathcal{Q}}{\rightarrow}{}_{\mathcal{R}}^{*} s_{i+1}\sigma$ only rules of $\mathcal{N}$ are used then $s_n \rightarrow t_n, s_{n+1} \rightarrow t_{n+1}, \dots$ is an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{N}, f)$ chain by Lemma 2.4.

Otherwise, there is some unneeded rule $\ell \rightarrow r \in \mathcal{R} \setminus \mathcal{N}$ which is used infinitely often in reductions of the chain. By Lemma 4.11 (viii) $s_1 \rightarrow t_1, s_2 \rightarrow t_2, \dots$ is also an infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{N} \cup \mathcal{C}_\varepsilon)$-chain where $\mathcal{I}(\sigma)$ is the corresponding substitution. A closer inspection of Lemma 4.11 (iv)-(vi) reveals that any application of an unneeded rule in the original chain with σ is transformed to at least one application of a rule of $\mathcal{C}_\varepsilon$ in the transformed chain with $\mathcal{I}(\sigma)$ as substitution. Due to the monotonicity of $\succ$ we can show as in Theorem 4.2 that there is an infinite decrease w.r.t. $\succ$ as the unneeded rule $\ell \rightarrow r$ is used infinitely often. This is in contradiction to the well-foundedness of $\succ$. □

Proof of Theorem 4.22. Completeness follows by Lemma 2.17. For soundness the proof is completely analogous to the proof of Theorem 4.2. □

Proof of Lemma 4.26. By a straightforward induction on the position p one can show that if $t \rightarrow_{\{\ell \rightarrow r\}, p} u$ and $p \in \mathit{RegPos}_\pi(t)$ then $\pi(t) \rightarrow_{\{\pi(\ell \rightarrow r)\}} \pi(u)$. $(\star)$

We now prove the following property which directly implies the lemma by ($\star$): if $s\sigma \in \mathit{NF}(\mathcal{Q})$ and $t\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} u$ then $\pi(t\sigma) \rightarrow^{*}_{\pi(\mathcal{U})} \pi(u)$ where $\mathcal{U} = \mathcal{EU}^{\{s\},\pi}_{\mathcal{R},\mathcal{Q}}(t)$.

To prove the property we perform induction on the reduction length. If there are no reductions then $\pi(t\sigma) = \pi(u)$ and we are done. Otherwise, $t\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} v \xrightarrow{\mathcal{Q}}{}_{\mathcal{R},p} u$. By the induction hypothesis we already obtain $\pi(t\sigma) \rightarrow^{*}_{\pi(\mathcal{U})} \pi(v)$. If $p \in \mathit{RegPos}_{\pi}(v)$ then by the definition of usable rules w.r.t. π the rule $\ell \rightarrow r$ is contained in $\mathcal{U}^{\{s\},\pi}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{U}$. Thus, we can apply ($\star$) to obtain $\pi(v) \rightarrow_{\pi(\mathcal{U})} \pi(u)$. Otherwise, if $p \notin \mathit{RegPos}_{\pi}(v)$ then $\pi(v) = \pi(u)$ which also implies $\pi(v) \rightarrow^{*}_{\pi(\mathcal{U})} \pi(u)$. □

Proof of Theorem 4.27. The proof is similar to the proof of Theorem 4.18. One just has to replace the application of Lemma 4.11 (viii) by Lemma 4.26. □

Proof of Theorem 4.30. If a rule $\ell \rightarrow r \in \mathcal{U}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$ then there is a substitution σ such that $\mathcal{S}\sigma \subseteq \mathit{NF}(\mathcal{Q})$ and $t\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} v \xrightarrow{\mathcal{Q}}{}_{\{\ell\rightarrow r\},p} u$ and $p \in \mathit{RegPos}_{\pi}(v)$. We show that $\ell \rightarrow r \in \mathcal{IU}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t)$ by induction on the lexicographic combination of first the reduction length and then the structure of t.

If t is a variable and $t\sigma$ can be reduced then $t\sigma \notin \mathit{NF}(\mathcal{R})$. If t does not occur as a subterm of a term in $\mathcal{S}$ or $\mathit{NF}(\mathcal{Q}) \not\subseteq \mathit{NF}(\mathcal{R})$ then $\mathcal{IU}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t) = \mathcal{R}$ and we are done. Otherwise, as $\mathcal{S}\sigma \subseteq \mathit{NF}(\mathcal{Q})$ we obtain $t\sigma \in \mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$ which is a contradiction to $t\sigma \notin \mathit{NF}(\mathcal{R})$.

So, in the remaining proof we let $t = f(t_1, \ldots, t_n)$. If any reduction step in $t\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} v$ is performed at a position $q \leq p$, i.e., $p = qq'$ then we choose q to be the highest such position. Note that $q \in \mathit{RegPos}_{\pi}(v)$ and $q' \in \mathit{RegPos}_{\pi}(v|_q)$. Hence, $t\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} w = w[\ell'\tau]_q \xrightarrow{\mathcal{Q}}{}_{\mathcal{R}} w[r'\tau]_q \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} v \xrightarrow{\mathcal{Q}}{}_{\{\ell\rightarrow r\},qq'} u$ and moreover, $r'\tau \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} v|_q \xrightarrow{\mathcal{Q}}{}_{\{\ell\rightarrow r,q'\}} u_q$ for some rule $\ell' \rightarrow r'$ with $\ell' = g(\ell'_1, \ldots, \ell'_m)$. Thus, all terms $\ell'_i\tau$ are in $\mathcal{Q}$-normal form. Therefore, we can use the induction hypothesis and obtain $\ell' \rightarrow r' \in \mathcal{IU}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$ and $\ell \rightarrow r \in \mathcal{IU}^{\{\ell'_1,\ldots,\ell'_m\}}_{\mathcal{R},\mathcal{Q}}(r')$. By Definition 4.29 the rule $\ell' \rightarrow r'$ is an element of $\mathcal{IU}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$, too.

Otherwise, there is no reduction above p in the reduction of $t\sigma$ to v. If $p = ip'$ then obviously $\pi(f) = i$ or $\pi(f) = [\ldots, i, \ldots]$, $p' \in \mathit{RegPos}_{\pi}(t_i)$, and $t_i\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} v|_i \xrightarrow{\mathcal{Q}}{}_{\{\ell\rightarrow r\},p'} u|_i$. As t_i is smaller than t and as the reduction of $t_i\sigma$ to $v|_i$ is not longer than the reduction of $t\sigma$ to v we know by induction that $\ell \rightarrow r \in \mathcal{IU}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t_i) \subseteq \mathcal{IU}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$. Finally, if $p = \varepsilon$ then the final reduction is the only reduction at root level. Hence, $t\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} f(v_1, \ldots, v_n) = v = \ell\tau \xrightarrow{\mathcal{Q}}{}_{\mathcal{R}} r\tau = u$ where $t_i\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} v_i$ for every i. Using Lemma 3.8 we know that $v = f(\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1)\mu, \ldots, \mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n)\mu) = f(\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1), \ldots, \mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n))\mu$, where μ differs from σ only on the free variables that are introduced by *ECap*. W.l.o.g. we may assume that μ is equal to τ on the variables of ℓ. Thus, $f(\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1), \ldots, \mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n))\mu = \ell\mu$ where all direct subterms of $\ell\mu$ are in $\mathcal{Q}$-normal form. Moreover, as σ differs from μ only on the fresh variables, all terms in $\mathcal{S}\mu$ are $\mathcal{Q}$-normal, too. Hence, ℓ is unifiable with $f(\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1), \ldots, \mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n))$ and the mgu δ also instantiates all direct subterms of ℓ and all terms in $\mathcal{S}$ to $\mathcal{Q}$-normal forms. A final look at Definition 4.29 reveals $\ell \rightarrow r \in \mathcal{IU}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t)$. □

Proof of Theorem 4.32. Completeness is proven by Lemma 2.17. The soundness proof is similar to the one of Theorem 4.18. Let $\mathcal{N} = \mathcal{N}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \pi)$. Given a minimal chain $s_1 \rightarrow t_1, s_2 \rightarrow t_2, \ldots$ either from some point all pairs are in $\mathcal{P} \setminus \succ_{\pi}$ or there is a strictly decreasing pair which occurs infinitely often. In the former case there is nothing to show and in the latter we use Lemma 4.35 (vi) to obtain

$$\pi(s_1)\sigma \rightarrow_{\pi(\mathcal{P})} \pi(t_1)\sigma \rightarrow^{*}_{\pi(\mathcal{N})\cup\mathcal{C}_{\varepsilon}} \pi(s_2)\sigma \rightarrow_{\pi(\mathcal{P})} \pi(t_2)\sigma \ldots$$

If we extend π by defining $\pi(\mathsf{c}) = [1,2]$ then $\pi(\mathcal{C}_\varepsilon) = \mathcal{C}_\varepsilon$. Hence, by the constraints of Theorem 4.32 and stability and monotonicity we obtain

$$\pi(s_1)\sigma \mathrel{{}_{(}{\succsim}_{)}} \pi(t_1)\sigma \succsim \pi(s_2)\sigma \mathrel{{}_{(}{\succsim}_{)}} \pi(t_2)\sigma \dots$$

with an infinite number of strict decreases. This is a contradiction to the well-foundedness of $\succ$. $\square$

Proof of Lemma 4.35. Note that by construction of $\mathcal{S}_{\mathit{all}}$ and σ all terms in $\mathcal{S}_{\mathit{all}}\sigma$ are $\mathcal{Q}$-normal.

(i) We perform induction on t. If t is a variable x then by definition $\mathcal{I}_\pi(t\sigma) = \mathcal{I}_\pi(x\sigma) = x\mathcal{I}_\pi(\sigma) = t\mathcal{I}_\pi(\sigma)$. Otherwise, let $t = f(t_1,\dots,t_n)$. Then we obtain $\mathcal{N}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t_i) \subseteq \mathcal{N}^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{N}$ for all $i \in \mathit{RegPos}_\pi(t)$. Hence, we may apply the induction hypothesis and know $\mathcal{I}_\pi(t_i\sigma) = t_i\mathcal{I}_\pi(\sigma)$ for all $i \in \mathit{RegPos}_\pi(t)$. If $\mathcal{I}_\pi(t\sigma)$ is built by the second case we immediately obtain $\mathcal{I}_\pi(t\sigma) = \pi(t)\mathcal{I}_\pi(\sigma)$ by the definition of $\mathcal{I}_\pi$. To show that $\mathcal{I}_\pi(t\sigma)$ cannot be built by the third case can be shown in the same way as in the proof of Lemma 4.11 (ii).

(ii) We prove the result by induction on t. If t is a variable then we use (i) with $\mathcal{S} = \varnothing$ to conclude $\mathcal{I}_\pi(t\sigma) = \pi(t)\mathcal{I}_\pi(\sigma)$.

Otherwise, if $t = f(t_1,\dots,t_n)$ we obtain by the induction hypothesis $\mathcal{I}_\pi(t_i\sigma) \to^*_{\mathcal{C}_\varepsilon} \pi(t_i)\mathcal{I}_\pi(\sigma)$ and hence, $\overline{t\sigma} \to^*_{\mathcal{C}_\varepsilon} \pi(t)\mathcal{I}_\pi(\sigma)$. Finally, if $\mathcal{I}_\pi(t\sigma)$ is built by the second case we are done and otherwise we perform one additional reduction step to obtain $\mathcal{I}_\pi(t\sigma) = \mathsf{c}(\overline{t\sigma}, \mathit{Comp}(\dots)) \to_{\mathcal{C}_\varepsilon} \overline{t\sigma} \to^*_{\mathcal{C}_\varepsilon} \pi(t)\mathcal{I}_\pi(\sigma)$.

(iii) The proof is a direct consequence of the definitions of $\mathit{Red}_\pi(t)$ and of Lemma 4.7. As $\mathcal{I}_\pi(t) = \mathsf{c}(\dots, \mathit{Comp}(\mathit{Red}_\pi(t)))$, $\mathcal{I}_\pi(s) \in \mathit{Red}_\pi(t)$, and as every term is in normal form w.r.t. the empty set we conclude $\mathcal{I}_\pi(t) \to^*_{\mathcal{C}_\varepsilon} \mathsf{c}(\dots, \mathcal{I}_\pi(s)) \to_{\mathcal{C}_\varepsilon} \mathcal{I}_\pi(s)$ by Lemma 4.7.

(iv) Let $t = \ell\sigma \xrightarrow{\mathcal{Q}}_{\mathcal{R}} r\sigma = s$ for some rule $\ell = f(\ell_1,\dots,\ell_n) \to r \in \mathcal{R}$. As in the proof of Lemma 4.11 (v) one can show that $\ell \to r \in \mathcal{N}$. Now by the closure properties of $\mathcal{N}$ we know that $\mathcal{N}^{\{\ell_1,\dots,\ell_n\},\pi}_{\mathcal{R},\mathcal{Q}}(r) \subseteq \mathcal{N}$. Moreover, by the construction of $\mathcal{S}_{\mathit{all}}$ we obtain $\{\ell_1,\dots,\ell_n\} \subseteq \mathcal{S}_{\mathit{all}}$. Hence, (ii) and (i) yield $\mathcal{I}_\pi(t) = \mathcal{I}_\pi(\ell\sigma) \to^*_{\mathcal{C}_\varepsilon} \pi(\ell)\mathcal{I}_\pi(\sigma) \to_{\pi(\mathcal{N})} \pi(r)\mathcal{I}_\pi(\sigma) = \mathcal{I}_\pi(r\sigma)$.

(v) We perform induction on t. Obviously t cannot be a variable and if t is built by the third case we use (iii). So let $t = f(t_1,\dots,t_n)$ and as $\mathcal{I}_\pi(t)$ is built by the second case we obtain $\mathcal{I}_\pi(t) = \overline{t}$. If the reduction is at the root position we use (iv). Otherwise, there is some t_i such that $t_i \xrightarrow{\mathcal{Q}}_{\mathcal{R}} s_i$ and $s = f(t_1,\dots,s_i,\dots,t_n)$. By induction we obtain $\mathcal{I}_\pi(t_i) \to^*_{\pi(\mathcal{N})\cup\mathcal{C}_\varepsilon} \mathcal{I}_\pi(s_i)$. Hence, $\mathcal{I}_\pi(t) = \overline{t} \to^*_{\pi(\mathcal{N})\cup\mathcal{C}_\varepsilon} \overline{s}$. Note that if $i \notin \mathit{RegPos}_\pi(t)$ then the i-th argument is dropped and we perform zero reduction steps from $\mathcal{I}_\pi(t)$ to $\overline{s}$. To prove that $\mathcal{I}_\pi(s) = \overline{s}$ is done in the same way as in the proof of Lemma 4.11 (vi).

(vi) As $u_1 \to v_1, u_2 \to v_2, \dots$ is a minimal chain for every i we know that $v_i\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} u_{i+1}\sigma$. Using the previous results we can now exchange $\mathcal{P}$ by $\pi(\mathcal{P})$, $\mathcal{R}$ by $\pi(\mathcal{N}) \cup \mathcal{C}_\varepsilon$, and σ by $\mathcal{I}_\pi(\sigma)$.

By construction of $\mathcal{N}$ we know that $\mathcal{N}^{\{u_i\},\pi}_{\mathcal{R},\mathcal{Q}}(v_i) \subseteq \mathcal{N}$ and $u_i \in \mathcal{S}_{all}$ for every i. Hence, using (i) we obtain $\mathcal{I}_\pi(v_i\sigma) = \pi(v_i)\mathcal{I}_\pi(\sigma)$. From (ii) we conclude $\mathcal{I}_\pi(u_i\sigma) \to^*_{\mathcal{C}_\varepsilon} \pi(u_i)\mathcal{I}_\pi(\sigma)$. By (v) we glue everything together and construct the new chain.

$$\begin{aligned} \pi(u_1)\mathcal{I}_\pi(\sigma) \to_{\pi(\mathcal{P})} \pi(v_1)\mathcal{I}_\pi(\sigma) = \mathcal{I}_\pi(v_1\sigma) \to^*_{\pi(\mathcal{N})\cup\mathcal{C}_\varepsilon} \\ \mathcal{I}_\pi(u_2\sigma) \to^*_{\mathcal{C}_\varepsilon} \pi(u_2)\mathcal{I}_\pi(\sigma) \to_{\pi(\mathcal{P})} \pi(v_2)\mathcal{I}_\pi(\sigma) = \mathcal{I}_\pi(v_2\sigma)\dots \end{aligned} \qquad \square$$

Proof of Lemma 4.37. We first define the two measures *rank* and *Rank*. Intuitively, *rank* counts the maximal nesting of head symbols in a term. Similarly, *Rank* also counts the maximal nesting of head symbols, but disregards the root-symbol of the given term. More formally, we define *rank* and *Rank* as follows.

$$rank(t) = \begin{cases} 0, & \text{if } t \text{ is a variable} \\ max\{rank(t_1),\dots,rank(t_n)\}+1, & \text{if } t = f(t_1,\dots,t_n),\ f \in \mathcal{H} \\ max\{rank(t_1),\dots,rank(t_n)\}, & \text{if } t = f(t_1,\dots,t_n),\ f \notin \mathcal{H} \end{cases}$$

$$Rank(t) = \begin{cases} 0, & \text{if } t \text{ is a variable} \\ max\{rank(t_1),\dots,rank(t_n)\}, & \text{if } t = f(t_1,\dots,t_n) \end{cases}$$

Here, we define the maximum of the empty set to be 0. We use the following properties of *rank* and *Rank*.

(i) $rank(t) \geq Rank(t)$ for all terms t.

(ii) If $t \in \mathcal{T}(\mathcal{F}\setminus\mathcal{H},\mathcal{V})$ then $rank(t\sigma) = max\{rank(x\sigma) \mid x \in \mathcal{V}(t)\}$.

(iii) If $t \to_\mathcal{R} s$ then $rank(t) \geq rank(s)$

(iv) If $t \to_\mathcal{R} s$ then $Rank(t) \geq Rank(s)$

(v) If $s \to t \in \mathcal{P}$ then $Rank(s\sigma) \geq Rank(t\sigma)$.

(vi) If $t \to_\mathcal{R}^* s$, $root(t) \notin \mathcal{H}$ and $root(s) \in \mathcal{H}$ then $Rank(t) > Rank(s)$.

(vii) If $s \to t \in \mathcal{P}$, $t \in \mathcal{V}$ and $root(t\sigma) \in \mathcal{H}$ then $Rank(s\sigma) > Rank(t\sigma)$.

Using these properties it is now easy to prove the lemma. If we have an infinite chain $s_1 \to t_1, s_2 \to t_2, \dots$ then we obtain $t_i\sigma \to_\mathcal{R}^* s_{i+1}\sigma$ for every i. Obviously whenever $root(t_i) \in \mathcal{H}$ then $root(s_{i+1}) \in \mathcal{H}$, because there is no rule in $\mathcal{R}$ to rewrite a head symbol. For the converse direction we first conclude by (iv), (v), and the fact that $>$ is well-founded on the natural numbers that there must be numbers n and c such that for all $i \geq n$ we have $c = Rank(s_i\sigma) = Rank(t_i\sigma)$. Let us assume $root(s_{i+1}) = root(s_{i+1}\sigma) \in \mathcal{H}$. Due to (vi) we know that $root(t_i\sigma) \in \mathcal{H}$. If $root(t_i) \in \mathcal{H}$ we are done. Otherwise, t_i must be a variable but this is a contradiction to (vii).

It remains to show that the listed properties of the measures are correct.

(i) Obvious.

(ii) We perform induction on t. If t is a variable then the result clearly holds. Otherwise, let $t = f(t_1,\dots,t_n)$. By the precondition $f \notin \mathcal{H}$ and thus we get $rank(t\sigma) = max\{rank(t_i\sigma) \mid 1 \leq i \leq n)\}$. Induction yields $rank(t\sigma) = max\{max\{rank(x\sigma) \mid x \in \mathcal{V}(t_i)\} \mid 1 \leq i \leq n\} = max\{rank(x\sigma) \mid x \in \mathcal{V}(t)\}$.

(iii) We perform induction on the position of the reduction. If $t = \ell\sigma \to_{\mathcal{R}} r\sigma = s$ then by the definition of $\mathcal{H}$ both ℓ and r are terms from $\mathcal{T}(\mathcal{F} \setminus \mathcal{H}, \mathcal{V})$. Thus, $\mathit{rank}(\ell\sigma) = \mathit{max}\{\mathit{rank}(x\sigma) \mid x \in \mathcal{V}(\ell)\} \geq \mathit{max}\{\mathit{rank}(x\sigma) \mid x \in \mathcal{V}(r)\} = \mathit{rank}(r\sigma)$. Otherwise, $t = f(t_1, \ldots, t_n) \to_{\mathcal{R}} f(t_1, \ldots, s_i, \ldots, t_n)$ where $t_i \to_{\mathcal{R}} s_i$. Induction yields $\mathit{rank}(t_i) \geq \mathit{rank}(s_i)$ and hence, $\mathit{rank}(t) - \mathit{rank}(s) = \mathit{max}\{\mathit{rank}(t_1), \ldots, \mathit{rank}(t_i), \ldots, \mathit{rank}(t_n)\} - \mathit{max}\{\mathit{rank}(t_1), \ldots, \mathit{rank}(s_i), \ldots, \mathit{rank}(t_n)\} \geq \mathit{rank}(t_i) - \mathit{rank}(s_i) \geq 0$.

(iv) If the reduction is below the root then the result is easily obtained from (iii). Otherwise, let $\ell\sigma \to_{\mathcal{R}} r\sigma$. By the definition of $\mathcal{H}$ we know that $\mathit{root}(\ell) \notin \mathcal{H}$ and hence $\mathit{Rank}(\ell\sigma) = \mathit{rank}(\ell\sigma) \geq \mathit{rank}(r\sigma) \geq \mathit{Rank}(r\sigma)$ by (iii) and (i).

(v) By the definition of $\mathcal{H}$ all proper subterms of s and t are of $\mathcal{T}(\mathcal{F} \setminus \mathcal{H}, \mathcal{V})$. Thus, by (ii) $\mathit{Rank}(s\sigma) = \mathit{max}\{\mathit{rank}(x\sigma) \mid x \in \mathcal{V}(s)\} \geq \mathit{max}\{\mathit{rank}(x\sigma) \mid x \in \mathcal{V}(t)\} =: m$. We show that $m \geq \mathit{Rank}(t\sigma)$. If $t \in \mathcal{T}(\mathcal{F} \setminus \mathcal{H}, \mathcal{V})$ then by (ii) and (i) $m = \mathit{rank}(t\sigma) \geq \mathit{Rank}(t\sigma)$. In the other case t must be of the form $f(t_1, \ldots, t_n)$ where by the construction of $\mathcal{H}$ the function symbol f is contained in $\mathcal{H}$ but all $t_i \in \mathcal{T}(\mathcal{F} \setminus \mathcal{H}, \mathcal{V})$. Thus, $\mathit{Rank}(t\sigma) = \mathit{max}\{\mathit{rank}(t_i\sigma) \mid 1 \leq i \leq n\} = \mathit{max}\{\mathit{rank}(x\sigma) \mid x \in \mathcal{V}(t_i), 1 \leq i \leq n\} = m$ by (ii).

(vi) As $\mathcal{R}$ does not contain any $\mathcal{H}$ symbols and as we get from t with $\mathit{root}(t) \notin \mathcal{H}$ to some term s with $\mathit{root}(s) \in \mathcal{H}$ we must have $t \to_{\mathcal{R}}{}^* t' = \ell\sigma \to_{\mathcal{R}} x\sigma = s' \to_{\mathcal{R}}{}^* s$ where t' has no root of $\mathcal{H}$, s' has a root of $\mathcal{H}$ and $\ell \to x$ is the collapsing rule that brings some head symbol to the root position. Thus, $\mathit{Rank}(t') = \mathit{Rank}(\ell\sigma) \geq \mathit{rank}(x\sigma) = 1 + \mathit{Rank}(x\sigma) > \mathit{Rank}(x\sigma) = \mathit{Rank}(s')$. Using (iv) we also have $\mathit{Rank}(t) > \mathit{Rank}(s)$.

(vii) Let $t = x$ and $\mathit{root}(x\sigma) = f(t_1, \ldots, t_n)$ where $f \in \mathcal{H}$. Thus, $\mathit{rank}(x\sigma) = \mathit{Rank}(x\sigma) + 1 > \mathit{Rank}(x\sigma) = \mathit{Rank}(t\sigma)$. On the other hand s is of the form $g(s_1, \ldots, s_n)$ where x must occur in some s_i. Hence, $\mathit{Rank}(s\sigma) \geq \mathit{rank}(s_i\sigma) = \mathit{max}\{\mathit{rank}(x\sigma) \mid x \in \mathcal{V}(s_i)\} \geq \mathit{rank}(x\sigma)$ by (ii). Thus, $\mathit{Rank}(s\sigma) > \mathit{Rank}(t\sigma)$. ☐

Proof of Theorem 4.39. Completeness follows from Lemma 2.17 and soundness is an immediate consequence of Lemma 4.37. ☐

Proof of Theorem 4.38. First note that the requirement of $\pi(\mathcal{P})$ being a TRS is only to ensure that $(\pi(\mathcal{P}), \mathcal{Q}, \mathcal{R}, \mathbf{m})$ really is a DP problem.

To prove soundness, let $s_1 \to t_1, s_2 \to t_2, \ldots$ be an infinite (minimal) $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain. Then there is a substitution σ such that for every i there is the reduction $t_i\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} s_{i+1}\sigma$, $s_i\sigma \in \mathit{NF}(\mathcal{Q})$, and if the chain is minimal then every $t_i\sigma$ is terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$.

We prove that $\pi(s_1) \to \pi(t_1), \pi(s_2) \to \pi(t_2), \ldots$ is an infinite chain using the same substitution σ. Therefore, we first show that each $\pi(s_i)\sigma$ is in $\mathcal{Q}$-normal form, and that $\pi(t_i)\sigma$ is terminating whenever $t_i\sigma$ is terminating. We consider two cases. If $s_i \trianglerighteq \pi(s_i)$ then obviously $\pi(s_i)\sigma \in \mathit{NF}(\mathcal{Q})$. Otherwise, $\mathit{root}(s_i) \in \mathcal{H}$, $s_i = f(u_1, \ldots, u_m)$ and $\pi(s_i) = f(u_{j_1}, \ldots, u_{j_k})$. Since $s_i\sigma \in \mathit{NF}(\mathcal{Q})$, the only possible $\mathcal{Q}$-redex of $\pi(s_i)\sigma$ can be at the root. But since the filtered, k-ary symbol f is a new symbol, it does not occur in $\mathcal{Q}$ and hence, $\pi(s_i)\sigma \in \mathit{NF}(\mathcal{Q})$. In the same way one can prove that $\pi(t_i)\sigma$ is terminating whenever $t_i\sigma$ is terminating.

It remains to prove $\pi(t_i)\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} \pi(s_{i+1})\sigma$. Due to Lemma 4.37 we can assume that for every i the equivalence $\mathit{root}(s_i) \in \mathcal{H} \Leftrightarrow \mathit{root}(t_{i+1}) \in \mathcal{H}$ is valid. If $\mathit{root}(t_i) \notin \mathcal{H}$ then $\pi(t_i) = t_i$ and due to the above equivalence, $\pi(s_{i+1}) = s_{i+1}$ is satisfied. In that case the result is immediately obtained from $t_i\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} s_{i+1}\sigma$. Otherwise, $\mathit{root}(t_i) \in \mathcal{H}$,

$t_i = f(u_1, \ldots, u_m)$, and since f does not occur in $\mathcal{R}$, we know that $s_{i+1} = f(v_1, \ldots, v_m)$ and $u_j\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} v_j\sigma$ for all j. As both $\pi(t_i) = f(u_{j_1}, \ldots, u_{j_k})$ and $\pi(s_{i+1}) = f(v_{j_1}, \ldots, v_{j_k})$ or both $\pi(t_i) = u_j$ and $\pi(s_{i+1}) = v_j$, this directly implies $\pi(t_i)\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} \pi(s_{i+1})\sigma$. $\square$

Proof of Theorem 4.41. Completeness follows from Lemma 2.17. For soundness consider an infinite chain $s_1 \to t_1, s_2 \to t_2, \ldots$. Let $\mathcal{H} = \mathcal{H}(\mathcal{P}, \mathcal{R})$. Using Theorem 4.39 we can w.l.o.g. assume that $\mathit{root}(t_i) \in \mathcal{H} \Leftrightarrow \mathit{root}(s_{i+1}) \in \mathcal{H}$. We consider two cases.

If $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$ then it suffices to apply Theorem 4.27 with π and as reduction pair $(\succsim, \succ)$ we use the embedding order and its reflexive closure for $\succ$ and $\succsim$, respectively. Then obviously the constraints for the pairs in $\mathcal{P}$ in Theorem 4.27 are satisfied and allow to delete the pairs in $\rhd_\pi \subseteq \succ_\pi$. Moreover, we will show that $\mathcal{IU}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \pi)$ is empty where we use *ICap* as estimated *Cap*-function. Thus, there are no constraints for $\mathcal{R}$ which finishes this case. To conclude that there are no usable rules w.r.t. π we show that $\mathcal{IU}^{\{s\},\pi}_{\mathcal{R},\mathcal{Q}}(t) = \varnothing$ for every $s \to t \in \mathcal{P}$. By a straight-forward induction on p one can show that $\mathit{ICap}^{\{s\}}_{\mathcal{R},\mathcal{Q}}(s|_p) = s|_p$ and $\mathcal{IU}^{\{s\}}_{\mathcal{R},\mathcal{Q}}(s|_p) = \varnothing$ for every subterm $s|_p$ of s. If $\mathit{root}(t) \notin \mathcal{H}$ this suffices to prove $\mathit{ICap}^{\{s\}}_{\mathcal{R},\mathcal{Q}}(t) = \varnothing$ as $s \unrhd \pi(s) \unrhd \pi(t) = t$. In the other case $t = g(t_1, \ldots, t_n)$ for a head symbol g. As g does not occur in $\mathcal{R}$ obviously the term $g(\mathit{ICap}^{\{s\}}_{\mathcal{R},\mathcal{Q}}(t_1), \ldots, \mathit{ICap}^{\{s\}}_{\mathcal{R},\mathcal{Q}}(t_n))$ does not unify with a left-hand side of $\mathcal{R}$. Thus, as $\pi(g) = i$ for some i the only usable rules are those of $\mathcal{IU}^{\{s\},\pi}_{\mathcal{R},\mathcal{Q}}(t_i)$. But as $s \unrhd \pi(s) \unrhd \pi(t) = t_i$ again there are no usable rules.

If we are not in the innermost case then by the requirement of the theorem we know that $f = \mathbf{m}$ and hence, we can assume that the terms $t_1\sigma$ is terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$. As π only filters head symbols this directly implies that the term $\pi(t_1)\sigma$ is also terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$. The reason is that for every term t that occurs in $\mathcal{P}$ the term $\pi(t)\sigma$ is either $t\sigma$ itself (in the case that $\mathit{root}(t) \notin \mathcal{H}$) or $\pi(t)\sigma$ is a subterm of $t\sigma$ (in the case that $\mathit{root}(t) \in \mathcal{H}$). Additionally $\pi(t_i)\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} \pi(s_{i+1})\sigma$ must be valid, too: If the roots of t_i and s_{i+1} are not in $\mathcal{H}$ then $\pi(t_i)\sigma = t_i\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} s_{i+1}\sigma = \pi(s_{i+1})\sigma$. Otherwise, both terms t_i and s_{i+1} must be rooted with the same head symbol g. Let $\pi(g) = j$. Then from $t_i\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} s_{i+1}\sigma$ we can conclude $t_i|_j\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} s_{i+1}|_j\sigma$ as the head symbol at the root position cannot be reduced.

As $\unrhd$ is stable and the conditions of the processor are satisfied we finally obtain $\pi(t_1)\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} \pi(s_2)\sigma \unrhd \pi(t_2)\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} \pi(s_3)\sigma \unrhd \ldots$. If a pair $s \to t$ with $\pi(s) \rhd \pi(t)$ occurs infinitely often in this chain we obtain an infinite $(\xrightarrow{\mathcal{Q}}_{\mathcal{R}} \cup \rhd)$-reduction starting in $\pi(s_1)\sigma$. This is a contradiction to the termination of $\pi(t_1)\sigma$ w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ as every term is terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ iff it is terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}} \cup \rhd$. $\square$

A.5. Proofs of Chapter 5

Proof of Theorem 5.3. Note that due to our *semantic Cap*-function this proof is even simpler than the original proof in [GA01, Theorem 20].

If $\ldots u_1 \to v_1$, $s \to t$, $u_2 \to v_2, \ldots$ is a $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain, then there exists a substitution σ such that

(i) $v_1\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} s\sigma$, $t\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} u_2\sigma$

(ii) $\{u_1\sigma, s\sigma, u_2\sigma\} \subseteq \mathit{NF}(\mathcal{Q})$

(iii) $(u_1 \to v_1, s \to t) \in E$ and $(s \to t, u_2 \to v_2) \in E$

First we use Lemma 3.8 to see that $s\sigma = \mathit{ECap}^{\{u_1,s\}}_{\mathcal{R},\mathcal{Q}}(v_1)\mu$ for some substitution μ that differs from σ at most on the variables that are introduced by *ECap*. W.l.o.g. we can assume that σ is equal to μ on all these fresh variables. Hence, $s\sigma = \mathit{ECap}^{\{u_1,s\}}_{\mathcal{R},\mathcal{Q}}(v_1)\sigma$ shows there is an mgu δ of s and $\mathit{ECap}^{\{u_1,s\}}_{\mathcal{R},\mathcal{Q}}(v_1)$ with $\sigma = \delta\tau$ for some substitution τ. Moreover, the property $\{s\sigma, u_1\sigma\} \subseteq \mathit{NF}(\mathcal{Q})$ must remain true when replacing σ by the more general substitution δ, i.e., $\{s\delta, u_1\delta\} \subseteq \mathit{NF}(\mathcal{Q})$. Hence, $s\delta \to t\delta \in N'$ and we obtain

(i) $v_1\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} s\sigma = (s\delta)\tau$, $(t\delta)\tau = t\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} u_2\sigma$

(ii) $\{u_1\sigma, s\sigma = (s\delta)\tau, u_2\sigma\} \subseteq \mathit{NF}(\mathcal{Q})$

(iii) $(u_1 \to v_1, s\delta \to t\delta) \in E'$ and $(s\delta \to t\delta, u_2 \to v_2) \in E'$

Thus, by renaming the variables in $s\delta \to t\delta$ to fresh ones we can extend σ to behave like τ on these fresh variables. In this way we can use σ to obtain the chain $\ldots u_1 \to v_1, s\delta \to t\delta, u_2 \to v_2, \ldots$. Moreover, when the original chain with $s \to t$ is minimal then by construction the new chain with $s\delta \to t\delta$ is minimal, too. The reason is that $t\sigma$ is terminating iff $t\delta\sigma$ is terminating as $t\sigma = t\delta\tau = t\delta\sigma$.

In this way, one can replace all occurrences of $s \to t$ in (minimal) chains except for the very first pair in the chain. However, if $s \to t, v_1 \to w_1, v_2 \to w_2, \ldots$ is an infinite minimal chain, then $v_1 \to w_1, v_2 \to w_2, \ldots$ is an infinite minimal chain as well.

For completeness, let $\ldots, s\delta \to t\delta, \ldots$ be a chain. Then by the construction of the graph $\mathcal{P}[s \to t/N']$ the sequence $\ldots, s \to t, \ldots$ is also a path in $\mathcal{P}$. And as different occurrences of pairs in $\mathcal{P}$ may be assumed variable disjoint, we can extend every substitution σ to behave like $\delta\sigma$ on the variables of s. Hence, this direction of the theorem is immediately proved. ☐

Proof of Theorem 5.5. If $\ldots, u_1 \to v_1, s \to t, u_2 \to v_2, \ldots$ is a $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain, then there exists a substitution σ such that

(i) $v_1\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} s\sigma$ and $t\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} u_2\sigma$

(ii) $\{u_1\sigma, s\sigma, u_2\sigma\} \subseteq \mathit{NF}(\mathcal{Q})$

(iii) $(u_1 \to v_1, s \to t)$ and $(s \to t, u_2 \to v_2) \in E$

Let $\mathcal{R}'' = \mathcal{EU}^{\{s,u_2\}}_{\mathcal{R},\mathcal{Q}}(t)$ and let $\mathcal{R}' = \mathcal{R}''^{-1}$. First we use Definition 3.24 to see that $t\sigma \to^*_{\mathcal{R}''} u\sigma$ and hence, $u_2\sigma \to^*_{\mathcal{R}'} t\sigma$. By Lemma 3.8 one obtains $t\sigma = \mathit{ECap}^{\varnothing}_{\mathcal{R}',\varnothing}(u_2)\mu$ for some substitution μ that differs on σ at most on the variables that are introduced by *ECap*. W.l.o.g. we can assume that σ is equal to μ on all these fresh variables. Hence, $t\sigma = \mathit{ECap}^{\varnothing}_{\mathcal{R}',\varnothing}(u_2)\sigma$ shows there is an mgu δ of t and $\mathit{ECap}^{\varnothing}_{\mathcal{R}',\varnothing}(u_2)$ with $\sigma = \delta\tau$ for some substitution τ.

The remainder of the soundness proof and the completeness proof is now completely analogous to the proof of Theorem 5.3. ☐

Proof of Lemma 5.7. Let $t \overset{\mathcal{Q}}{\to}_{\mathcal{R},p_1} t_1$ and $t \overset{\mathcal{Q}}{\to}_{\mathcal{R},p_2} t_2$ by rules $\ell_i \to r_i \in \mathcal{R}$ for $1 \leq i \leq 2$ where by renaming of the variables we can assume that both rules are instantiated by the same substitution σ. Since $t|_{p_1}$ and $t|_{p_2}$ are no $\mathcal{R}$-normal forms and thus no $\mathcal{Q}$-normal forms (by $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$), p_2 cannot be strictly above p_1 and p_1 cannot be strictly above p_2. If $p_1 = p_2$, then for $1 \leq i \leq 2$ we know $t = C[\ell_i\sigma]_{p_1}$ and $t_i = C[r_i\sigma]_{p_1}$. Hence, by the requirements of the lemma we directly conclude $r_1\sigma = r_2\sigma$ and thus, $t_1 = t_2$. Otherwise, p_1 and p_2 are independent and thus, t_1 and t_2 can obviously be joined in one step. ☐

Proof of Theorem 5.10. Let $t = t[\ell\mu]_p \to_{\mathcal{R}} t[r\mu]_p = t'$ for some $\ell \to r \in \mathcal{R}$ and substitution μ.

We first prove the soundness and only consider the case where $f = \mathbf{m}$. The case $f = \mathsf{a}$ is analogous. Let $s \to t, u \to v$ be a minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain. Thus, $t\sigma \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} u\sigma$, both $s\sigma$ and $u\sigma$ are in $\mathcal{Q}$-normal form, and $t\sigma$ is terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$. We want to show $t'\sigma \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} u\sigma$ and that $t'\sigma$ is also terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$. Then we can exchange all occurrences of $s \to t$ in chains by $s \to t'$.

We consider the reduction $t\sigma = t\sigma[\ell\mu\sigma]_p \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} u\sigma$. As $u\sigma$ is in $\mathcal{Q}$-normal form and as $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$-reductions cannot take place strictly above $\mathcal{Q}$-redexes, w.l.o.g. we first reduce $\ell\mu\sigma$ to some $\mathcal{Q}$-normal form w. Thus, $t\sigma = t\sigma[\ell\mu\sigma]_p \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} t\sigma[w]_p \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} u\sigma$ where $\ell\mu\sigma \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} w$. The only rules applicable to $t|_p\sigma = \ell\mu\sigma$ or its reducts are from $\mathcal{U}^{\{s\}}_{\mathcal{R},\mathcal{Q}}(t|_p) \subseteq \mathcal{U}$. Hence, $\ell\mu\sigma \xrightarrow{\mathcal{Q}}^*_{\mathcal{U}} w$. As w is a $\mathcal{Q}$-normal form, w.l.o.g. one first reduces all terms $x\mu\sigma$ with $x \in \mathcal{V}(\ell)$ to $\mathcal{Q}$-normal forms. As $\xrightarrow{\mathcal{Q}}_{\mathcal{U}}$ is confluent these normal forms are unique. Thus, $\ell\mu\sigma \xrightarrow{\mathcal{Q}}^*_{\mathcal{U}} \ell\delta \xrightarrow{\mathcal{Q}}^*_{\mathcal{U}} w$ for some $\mathcal{Q}$-normal substitution δ and for all $x \in \mathcal{V}(\ell)$ we have $x\mu\sigma \xrightarrow{\mathcal{Q}}^*_{\mathcal{U}} x\delta$. Note that $\ell\delta$ is not yet a $\mathcal{Q}$-normal form as $\ell \to r \in \mathcal{R}$ and $NF(\mathcal{Q}) \subseteq NF(\mathcal{R})$. Thus, we need at least one more step to get from $\ell\delta$ to w. As δ is a $\mathcal{Q}$-normal substitution, the reduction is above δ and as all critical pairs between $\ell \to r$ and $\mathcal{U}$ are trivial joinable, the only possible reduction is $\ell\delta \xrightarrow{\mathcal{Q}}_{\mathcal{U}} r\delta \xrightarrow{\mathcal{Q}}^*_{\mathcal{U}} w$. This finally proves $t'\sigma = t\sigma[r\mu\sigma]_p \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} t\sigma[r\delta]_p \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} t\sigma[w]_p \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} u\sigma$.

Now minimality (i.e., termination of $t'\sigma$ w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$), can be proved in an analogous way. As before, w.l.o.g. any infinite $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$-reduction of $t'\sigma = t\sigma[r\mu\sigma]_p$ first reduces all redexes in $x\mu\sigma$ for $x \in \mathcal{V}(r)$. These reductions either lead to non-termination or they end in some $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$-normal forms. Since $x \in \mathcal{V}(r) \subseteq \mathcal{V}(\ell)$, all $x\mu$ are contained in $t|_p$. Thus, for every $x \in \mathcal{V}(r)$ the term $x\mu\sigma$ is terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ and the possible rules to rewrite are all contained in $\mathcal{U}$. Hence, by Lemma 2.4 all these steps can also be done by $\xrightarrow{\mathcal{Q}}_{\mathcal{U}}$ and as $\xrightarrow{\mathcal{Q}}_{\mathcal{U}}$ is confluent, the reduction must begin with $r\mu\sigma \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} r\delta$. Thus, whenever $t'\sigma$ is non-terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ then so is $t\sigma[r\delta]_p$. But this would contradict the termination of $t\sigma$ w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ as we know that $t\sigma \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} t\sigma[r\delta]_p$.

The completeness of the rewriting processor is obvious if $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ is not terminating. Otherwise, we show that if there is a reduction $t'\sigma \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} u\sigma$ such that $s\sigma$ and $u\sigma$ are in $\mathcal{Q}$-normal form, then $t\sigma \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} u\sigma$ also holds. We use the same way of reasoning as for the soundness proof. So if $t'\sigma = t\sigma[r\mu\sigma]_p \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} u\sigma$, we may assume that we first reduce $r\mu\sigma$ to some $\mathcal{Q}$-normal form w which is again started with reducing every subterm $x\mu\sigma$ with $x \in \mathcal{V}(r)$ to a $\mathcal{Q}$-normal form by $\xrightarrow{\mathcal{Q}}_{\mathcal{U}}$ reductions. By the confluence of $\xrightarrow{\mathcal{Q}}_{\mathcal{U}}$, we have $r\mu\sigma \xrightarrow{\mathcal{Q}}^*_{\mathcal{U}} r\delta \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} w$ for some $\mathcal{Q}$-normal substitution δ. In the same way as before we obtain $t'\sigma = t\sigma[r\mu\sigma]_p \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} t\sigma[r\delta]_p \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} t\sigma[w]_p \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} u\sigma$. It remains to show that $\ell\mu\sigma \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} r\delta$, as this implies $t\sigma = t\sigma[\ell\mu\sigma]_p \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} t\sigma[r\delta]_p \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} t\sigma[w]_p \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} u\sigma$.

We know that $x\mu\sigma \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} x\delta$ for all $x \in \mathcal{V}(r)$, where $x\delta$ is in $\mathcal{Q}$-normal form. By $NF(\mathcal{Q}) \subseteq NF(\mathcal{R})$, $x\delta$ is in normal form w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$, too. As $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ is terminating, we can extend δ such that $x\delta$ is a normal form of $x\mu\sigma$ w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ for every variable $x \in \mathcal{V}(\ell) \setminus \mathcal{V}(r)$. Then we have $\ell\mu\sigma \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} \ell\delta$ for the $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$-normal substitution δ. To prove the desired result $\ell\delta \xrightarrow{\mathcal{Q}}_{\mathcal{R}} r\delta$, we prove that for every position p' of ℓ there is the reduction $\ell\delta \xrightarrow{\mathcal{Q}}_{\mathcal{R}} r\delta$ at a position below p' or the term $\ell\delta|_{p'}$ is in $\mathcal{Q}$-normal form. This suffices to prove $\ell\delta \xrightarrow{\mathcal{Q}}_{\mathcal{R}} r\delta$ as for $p' = \varepsilon$ we know that $\ell\delta|_{p'} = \ell\delta \notin NF(\mathcal{R}) \supseteq NF(\mathcal{Q})$.

We perform induction on $\ell|_{p'}$. If $\ell|_{p'}$ is a variable then by the requirements of the processor we know $nfc^{\{s\}}_{\mathcal{R},\mathcal{Q}}(\ell|_{p'}\mu)$. Thus, $\ell|_{p'}\mu\sigma \xrightarrow{\mathcal{Q}}^*_{\mathcal{R}} \ell|_{p'}\delta$ is a reduction to a normal form w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ and by Definition 5.8 we can infer $\ell|_{p'}\delta = \ell\delta|_{p'} \in NF(\mathcal{Q})$. Otherwise, $\ell|_{p'} = f(\ell_1, \ldots, \ell_n)$. If by induction there is an $1 \leq i \leq n$ such that $\ell\delta \xrightarrow{\mathcal{Q}}_{\mathcal{R}} r\delta$ at a position below $p'i$ then we are done immediately. In the other case we know that for every $1 \leq i \leq n$ the term $\ell\delta|_{p'i}$ is in $\mathcal{Q}$-normal form and hence, every reduction of $\ell\delta$ at position p' respects the strategy. If there is a reduction of $\ell\delta \xrightarrow{\mathcal{Q}}_{\mathcal{R},p'} v$ then this gives rise to a critical pair and by the requirements of the processor we know $v = r\delta$. In the final case, there is no reduction at position p' possible. Hence, $\ell\delta|_{p'}$ is in normal form w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$. This case obviously cannot occur for $p' = \varepsilon$. Thus, by the requirements of the processor we know $\mathit{nfc}^{\{s\}}_{\mathcal{R},\mathcal{Q}}(\ell|_{p'}\mu)$ and conclude $\ell\delta|_{p'} \in \mathit{NF}(\mathcal{Q})$ by Definition 5.8 as in the variable case. □

Proof of Theorem 5.17. If $\mathit{nfc}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) = \mathit{false}$ then there is a substitution σ such that $\mathcal{S}\sigma \subseteq \mathit{NF}(\mathcal{Q})$ and $t\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} u$ for some $u \notin \mathit{NF}(\mathcal{Q})$ which is in normal form w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$. We perform induction on first the length of this reduction and then the structure of t to show that $\mathit{enfc}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) = \mathit{false}$.

First we consider the case that t is a variable. Hence, due to $\mathcal{V}(t) \subseteq \mathcal{V}(s)$ and $\mathcal{S}\sigma \subseteq \mathit{NF}(\mathcal{Q})$ the term $t\sigma$ is in $\mathcal{Q}$-normal form. Moreover, $t\sigma$ is also in normal form w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ as $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$. Thus, $u = t\sigma$ which contradicts $u \notin \mathit{NF}(\mathcal{Q})$. So, in the remaining proof we may assume $t = f(t_1, \ldots, t_n)$.

In the second case there is at least one reduction at the root position. Hence, $t\sigma = f(t_1\sigma, \ldots, t_n\sigma) \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} f(u_1, \ldots, u_n) = \ell\sigma \xrightarrow{\mathcal{Q}}_{\mathcal{R}} r\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} u$ for some rule $f(\ell_1, \ldots, \ell_n) = \ell \to r \in \mathcal{R}$. We define $\mathcal{S}' = \{\ell_1, \ldots, \ell_n\}$ as the direct subterms of ℓ and conclude $\mathcal{S}'\sigma \subseteq \mathit{NF}(\mathcal{Q})$ as there is a root reduction. Thus, $\mathit{nfc}^{\mathcal{S}'}_{\mathcal{R},\mathcal{Q}}(r) = \mathit{false}$ and by induction $\mathit{enfc}^{\mathcal{S}'}_{\mathcal{R},\mathcal{Q}}(r) = \mathit{false}$. Moreover, using Lemma 3.8 we know that $f(u_1, \ldots, u_n) = f(\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1)\delta, \ldots, \mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n))$ for some δ that differs from σ at most on the variables that are introduced by ECap. Therefore, we may assume $\sigma = \delta$ which shows that $f(\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1), \ldots, \mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n))$ and ℓ are unifiable by some mgu μ with $\sigma = \mu\tau$ for some substitution τ. Hence, $\mathcal{S}\mu \cup \mathcal{S}'\mu \subseteq \mathit{NF}(\mathcal{Q})$ and by Definition 5.16 the required fact that $\mathit{enfc}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) = \mathit{false}$ follows from $\mathit{enfc}^{\mathcal{S}'}_{\mathcal{R},\mathcal{Q}}(r) = \mathit{false}$.

Finally, it may be the case that there are no root reductions at all in the rewrite sequence $t\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} u$. As in the second case we obtain $t\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} f(u_1, \ldots, u_n)$ where $t_i\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} u_i$ for every i. Here, every u_i is in normal form w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ as u itself is a normal form w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$. If some u_i is not in $\mathcal{Q}$-normal form then obviously $\mathit{nfc}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_i)$ does not hold and by induction we obtain $\mathit{enfc}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_i) = \mathit{false}$ which shows $\mathit{enfc}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) = \mathit{false}$ by Definition 5.16. It remains the case that every u_i is in $\mathcal{Q}$-normal form. As $u \notin \mathit{NF}(\mathcal{Q})$ the only possible redex can be at the root position, i.e., $u = q\sigma$ for some $q \in \mathcal{Q}$. Similar to the second case we obtain $f(\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1)\sigma, \ldots, \mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n)\sigma) = f(u_1, \ldots, u_n) = u = q\sigma$ and hence q and $f(\mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_1), \ldots, \mathit{ECap}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_n))$ are unifiable by some mgu μ with $\sigma = \mu\tau$ for some substitution τ. Then clearly $\mathcal{S}\mu \subseteq \mathit{NF}(\mathcal{Q})$ but we also have $q\mu \in \mathit{NF}(\mathcal{R})$. The reason is that otherwise $u = q\sigma = q\mu\tau$ would be reducible by $\mathcal{R}$ and this contradicts the fact that u is in $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$-normal form and hence in $\mathcal{R}$-normal form as all subterms of u are in $\mathcal{Q}$-normal form. Thus, by Definition 5.16 we obtain $\mathit{enfc}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) = \mathit{false}$ and are done. □

Proof of Theorem 5.19. The proof is an extension of the corresponding proofs of [AG00, Theorems 27 and 42]. For soundness, we prove that if there is an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain, then there is also an infinite minimal $(\mathcal{P}[s \to t/N'], \mathcal{Q}, \mathcal{R})$-chain. To this end, it suffices to show that if

$$\ldots\; u_1 \to v_1,\; s \to t, u_2 \to v_2\; \ldots$$

is a minimal chain, then there is a pair $s' \to t' \in N'$ such that

$$\ldots\; u_1 \to v_1,\; s' \to t', u_2 \to v_2\; \ldots$$

is also a minimal chain. Here, $s \to t$ resp. $s' \to t'$ may also be the first pair in the chain (i.e., $u_1 \to v_1$ may be missing). If this has been proved then all occurrences of $s \to t$ in an infinite minimal chain may be replaced by its narrowings. The result for arbitrary chains can be achieved in the same way.

There must be a substitution σ such that

(i) $v_1\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} s\sigma$, $t\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} u_2\sigma$

(ii) $u_1\sigma$, $s\sigma$, and $u_2\sigma$ are in $\mathcal{Q}$-normal form

(iii) $v_1\sigma$, $t\sigma$, and $v_2\sigma$ are terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$.

Let σ be such a substitution where the length of the reduction $t\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} u_2\sigma$ is *minimal.* Note that we have to perform at least one step below position p which is w.l.o.g. the first step in the reduction. To see this, we consider two cases.

In the one case $t|_p \notin \mathit{NF}(\mathcal{Q})$. Hence, if one only reduces $t\sigma$ at positions independent of p then the resulting term will still contain $t|_p\sigma$ which is not in $\mathcal{Q}$-normal form. As $u_2\sigma \in \mathit{NF}(\mathcal{Q})$ there must be some first reduction at a position which is not independent of p. And since $t|_p\sigma \notin \mathit{NF}(\mathcal{Q})$ this reduction must be below p. By reordering the reduction $t\sigma \xrightarrow{\mathcal{Q}}{}^{+}_{\mathcal{R}} u_2\sigma$ w.l.o.g. one can assume that the reduction below p is the first step.

In the other case remark that $p \in \mathit{Pos}(\mathit{ECap}^{\{s\}}_{\mathcal{R},\mathcal{Q}}(t))$ implies that there can be no reductions strictly above p as otherwise *ECap* would have replaced a position strictly above p with a fresh variable. Second, if there is no reduction step below p then σ would be a unifier of $t|_p$ and $u_2|_p$, i.e., there would be an mgu μ of $t|_p$ and $u_2|_p$ such that $\sigma = \mu\tau$. But since $s\mu$ or $u_2\mu$ is no $\mathcal{Q}$-normal form, then $s\sigma$ or $u_2\sigma$ would not be a $\mathcal{Q}$-normal form either, which contradicts (ii). As there are no reductions above p w.l.o.g. we assume that the first step in the reduction is at a position below p.

Hence, we have $t\sigma \xrightarrow{\mathcal{Q}}_{\mathcal{R}} q \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} u_2\sigma$ for some term q and the first reduction is below the position p. There are two possibilities for the reduction $t\sigma \xrightarrow{\mathcal{Q}}_{\mathcal{R}} q$. Let us first assume that this reduction takes place 'in σ'. Hence, there is a variable x in t (i.e., $t|_{p'} = x$ for some position p') such that $\sigma(x) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} r$ and $w = t[r]_{p'}$. Clearly, this cannot happen if $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$. Otherwise, we have $s\sigma \in \mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$ due to (ii) and as $\mathcal{V}(t) \subseteq \mathcal{V}(s)$ we obtain $x\sigma \in \mathit{NF}(\mathcal{Q})$ being a contradiction. Hence, $\mathit{NF}(\mathcal{Q}) \not\subseteq \mathit{NF}(\mathcal{R})$.

Thus, the variable x only occurs *once* in t (as t is linear) and therefore, we have $q = t\sigma'$, where σ' is the substitution with $\sigma'(x) = r$ and $\sigma'(y) = \sigma(y)$ for all $y \neq x$. As all (occurrences of) dependency pairs are variable disjoint, σ' behaves like σ for all pairs except $s \to t$. Thus, we have

(i) $v_1\sigma' = v_1\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} s\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} s\sigma'$, $t\sigma' = q \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} u_2\sigma = u_2\sigma'$

(ii) $u_i\sigma'$ and $s\sigma'$ are in $\mathcal{Q}$-normal form (since $\mathcal{Q} = \varnothing$). Here is the reason for requiring $\mathcal{Q} = \varnothing$ if $\mathit{NF}(\mathcal{Q}) \not\subseteq \mathit{NF}(\mathcal{R})$. Otherwise, we could not conclude that $s\sigma'$ is in $\mathcal{Q}$-normal form.

(iii) $v_i\sigma' = v_i\sigma$ are terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$. Moreover, since $t\sigma$ terminates and $t\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} t\sigma'$, $t\sigma'$ is also terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$.

But as the reduction from $t\sigma'$ to $u_2\sigma'$ is shorter than the reduction from $t\sigma$ to $u_2\sigma$, this is a contradiction to the minimality of σ.

So the reduction $t\sigma \xrightarrow{\mathcal{Q}}_{\mathcal{R}} q$ cannot take place 'in σ'. Hence, t contains some non-variable subterm w at a position p' below p such that a rule $\ell \to r$ has been applied to $w\sigma$. In other words, ℓ matches $w\sigma$ (i.e., $\ell\rho = w\sigma$). Hence, the reduction has the following form:

$$t\sigma = t\sigma[w\sigma]_{p'} = t\sigma[\ell\rho]_{p'} \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t\sigma[r\rho]_{p'} = q.$$

We again assume that the variables of $\ell \to r$ have been renamed to fresh ones. Therefore we can extend σ to 'behave' like ρ on the variables of ℓ and r (but it still remains the same on the variables of all pairs in the chain). Now σ is a unifier of ℓ and w and hence, there also exists a most general unifier μ. By the definition of most general unifiers, then there must be a substitution τ such that $\sigma = \mu\tau$. Note that since $\ell\sigma = \ell\rho \xrightarrow{\mathcal{Q}}_{\mathcal{R}} r\rho = r\sigma$, all proper subterms of $\ell\sigma$ must be in $\mathcal{Q}$-normal form. But then all proper subterms of $\ell\mu$ are in $\mathcal{Q}$-normal form as well. Similarly, $s\sigma$ and thus, $s\mu$ are in $\mathcal{Q}$-normal form as well.

Let t' be the term $t\mu[r\mu]_{p'}$ and let s' be $s\mu$. Then $s \to t$ narrows to $s' \to t'$ at position p' (since $s\mu$ and the proper subterms of $\ell\mu$ are in $\mathcal{Q}$-normal form) and hence, $s' \to t' \in N'$. As we may assume s' and t' to be variable disjoint from all other pairs, we may extend σ to behave like τ on the variables of s' and t'. Then we have

(i) $v_1\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} s\sigma = s\mu\tau = s'\tau = s'\sigma$ and $t'\sigma = t'\tau = t\mu\tau[r\mu\tau]_{p'} = t\sigma[r\sigma]_{p'} = t\sigma[r\rho]_{p'} = q \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} u_2\sigma$.

(ii) $u_i\sigma$ and $s'\sigma = s\sigma$ are in $\mathcal{Q}$-normal form

(iii) $v_i\sigma$ are terminating. Moreover, since $t\sigma$ terminates and $t\sigma \xrightarrow{\mathcal{Q}}_{\mathcal{R}} q = t'\sigma$, $t'\sigma$ terminates as well.

The completeness proof for $\mathcal{Q} = \varnothing$ is completely analogous to the one of [AG00, Theorem 27] since we do not have to consider minimal, but ordinary chains. The reason is that if $\mathcal{R}$ is not terminating then completeness is immediately satisfied.

Now, let us consider the case $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$. Here we mainly refer to the completeness proofs of the instantiation and the rewriting processors. The important fact is that narrowing is just a sequence of an instantiation and a rewrite step. To be more precise, $s' \to t' \in N'$ implies that $s' = s\mu$ and $t\mu \xrightarrow{\mathcal{Q}}_{\mathcal{R},p'} t'$. Hence, we can first argue as in Theorem 5.3 that replacing $s \to t$ by $s\mu \to t\mu$ is complete. The next step from $s\mu \to t\mu$ to $s\mu = s' \to t'$ is nothing but a rewrite step at position p' as analyzed in Theorem 5.10, and thus, is also complete. □

A.6. Proofs of Chapter 6

Proof of Lemma 6.4. (i) The claim is proved by straightforward structural inductions. For $t = x$ the result is obtained by the definition of $\mathcal{A}(\sigma)$. In the case $t = f^{?_1}t_1{}^{?_2}\ldots{}^{?_n}t_n$ we conclude

$$\begin{array}{rcl}
\mathcal{A}(t\sigma) & = & \mathcal{A}(f^{?_1}t_1\sigma^{?_2}\ldots{}^{?_n}t_n\sigma) \\
 & = & f_{?_1\ldots?_n}(\mathcal{A}(t_1\sigma),\ldots,\mathcal{A}(t_n\sigma)) \\
\text{(by ind.)} & = & f_{?_1\ldots?_n}(\mathcal{A}(t_1)\mathcal{A}(\sigma),\ldots,\mathcal{A}(t_n)\mathcal{A}(\sigma)) \\
 & = & f_{?_1\ldots?_n}(\mathcal{A}(t_1),\ldots,\mathcal{A}(t_n))\mathcal{A}(\sigma) \\
 & = & \mathcal{A}(t)\mathcal{A}(\sigma)
\end{array}$$

For $u = x$ the result is obtained by the definition of $\mathcal{A}^{-1}(\delta)$. In the case $u = f_{?_1\ldots?_n}(u_1, \ldots, u_n)$ we conclude

$$\begin{array}{rcl}
\mathcal{A}^{-1}(u\delta) & = & \mathcal{A}^{-1}(f_{?_1\ldots?_n}(u_1\delta, \ldots, u_n\delta)) \\
 & = & f^{?_1}\mathcal{A}^{-1}(u_1\delta)^{?_2}\ldots{}^{?_n}\mathcal{A}^{-1}(u_n\delta) \\
\text{(by ind.)} & = & f^{?_1}\mathcal{A}^{-1}(u_1)\mathcal{A}^{-1}(\delta)^{?_2}\ldots{}^{?_n}\mathcal{A}^{-1}(u_n)\mathcal{A}^{-1}(\delta) \\
 & = & (f^{?_1}\mathcal{A}^{-1}(u_1)^{?_2}\ldots{}^{?_n}\mathcal{A}^{-1}(u_n))\mathcal{A}^{-1}(\delta) \\
 & = & \mathcal{A}^{-1}(u)\mathcal{A}^{-1}(\delta)
\end{array}$$

(ii) We perform induction on t. As $\mathcal{R}$ is proper in the rule $\ell \to r$ that is used for the reduction both terms ℓ and r are proper. If we have a root reduction then $t = \ell\sigma \to_{\mathcal{R}} r\sigma = s$. Hence, by Definition 6.2 all terms $x\sigma$ with $x \in \mathcal{V}(\ell)$ are proper. As $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$ we conclude that $r\sigma = s$ is proper, too.

Otherwise, $t = f^{?_1}t_1{}^{?_2}\ldots{}^{?_n}t_n$ and the reduction is below the root. As $\mathcal{R}$ is proper the reduction must be inside some t_i. Thus, $t_i \to_{\mathcal{R}} s_i$ and $s = f^{?_1}t_1{}^{?_2}\ldots{}^{?_i}s_i{}^{?_{i+1}}\ldots{}^{?_n}t_n$. By induction we conclude that s_i is proper and therefore, s is proper, too.

(iii) We perform induction on t. If t is a variable then there is nothing to show. Otherwise, $t = f^{?_1}t_1{}^{?_2}\ldots{}^{?_n}t_n$ and as $\mathcal{Q}$ is proper the only possible way that t is not in $\mathcal{Q}$-normal form is that $t = q\sigma$ for some $q \in \mathcal{Q}$ or that some t_i is not in $\mathcal{Q}$-normal form. Thus, we obtain

$$\begin{array}{rl}
 & t \notin \mathit{NF}(\mathcal{Q}) \\
 & \Leftrightarrow \text{some } t_i \notin \mathit{NF}(\mathcal{Q}) \text{ or } t = q\sigma \text{ for some } q \in \mathcal{Q} \\
\text{(ind.)} & \Leftrightarrow \text{some } \mathcal{A}(t_i) \notin \mathit{NF}(\mathcal{A}(\mathcal{Q})) \text{ or } t = q\sigma \text{ for some } q \in \mathcal{Q} \\
(i) & \Leftrightarrow \text{some } \mathcal{A}(t_i) \notin \mathit{NF}(\mathcal{A}(\mathcal{Q})) \text{ or } \mathcal{A}(t) = \mathcal{A}(q)\mathcal{A}(\sigma) \text{ for some } q \in \mathcal{Q} \\
 & \Leftrightarrow \text{some } \mathcal{A}(t_i) \notin \mathit{NF}(\mathcal{A}(\mathcal{Q})) \text{ or } \mathcal{A}(t) = q'\sigma' \text{ for some } q' \in \mathcal{A}(\mathcal{Q}) \\
 & \Leftrightarrow \mathcal{A}(t) = f_{?_1\ldots?_n}(\mathcal{A}(t_1), \ldots, \mathcal{A}(t_n)) \notin \mathit{NF}(\mathcal{A}(\mathcal{Q}))
\end{array}$$

(iv) It suffices to show that $t \overset{\mathcal{Q}}{\to}_{\mathcal{R}} s$ iff $\mathcal{A}(t) \overset{\mathcal{A}(\mathcal{Q})}{\to}_{\mathcal{A}(\mathcal{R})} \mathcal{A}(s)$ by structural induction on t. Then the claim for $m > 1$ follows by induction and (ii). This is sufficient, since $\mathcal{A}$ is surjective.

We first prove the "only if" direction by induction on t. As t is no variable we consider $t = f^{?_1}t_1{}^{?_2}\ldots{}^{?_n}t_n$. If some t_i is reduced to s_i then the induction hypothesis yields $\mathcal{A}(t_i) \overset{\mathcal{A}(\mathcal{Q})}{\to}_{\mathcal{A}(\mathcal{R})} \mathcal{A}(s_i)$. Thus, $\mathcal{A}(t) = f_{?_1\ldots?_n}(\mathcal{A}(t_1), \ldots, \mathcal{A}(t_i), \ldots, \mathcal{A}(t_n)) \overset{\mathcal{A}(\mathcal{Q})}{\to}_{\mathcal{A}(\mathcal{R})} f_{?_1\ldots?_n}(\mathcal{A}(t_1), \ldots, \mathcal{A}(s_i), \ldots, \mathcal{A}(t_n)) = \mathcal{A}(s)$. Otherwise, we have $t = \ell\sigma \overset{\mathcal{Q}}{\to}_{\mathcal{R}} r\sigma = s$ as $\mathcal{R}$ is proper. Then $\mathcal{A}(t) = \mathcal{A}(\ell\sigma) = \mathcal{A}(\ell)\mathcal{A}(\sigma) \to_{\mathcal{A}(\mathcal{R})} \mathcal{A}(r)\mathcal{A}(\sigma) = \mathcal{A}(r\sigma) = \mathcal{A}(s)$ by (i). As each t_i is a subterm of t and therefore in $\mathcal{Q}$-normal form by (iii) we know that every $\mathcal{A}(t_i)$ is in $\mathcal{A}(\mathcal{Q})$-normal form. Thus, every direct subterm of $\mathcal{A}(t)$ is in $\mathcal{A}(\mathcal{Q})$-normal form which proves $\mathcal{A}(t) \overset{\mathcal{A}(\mathcal{Q})}{\to}_{\mathcal{A}(\mathcal{R})} \mathcal{A}(s)$.

For the other direction we assume $\mathcal{A}(t) \overset{\mathcal{A}(\mathcal{Q})}{\to}_{\mathcal{A}(\mathcal{R})} \mathcal{A}(s)$. Again, we only consider $t = f^{?_1}t_1{}^{?_2}\ldots{}^{?_n}t_n$, since t can be no variable. Hence, $\mathcal{A}(t) = f_{?_1\ldots?_n}(\mathcal{A}(t_1), \ldots, \mathcal{A}(t_n))$. If $\mathcal{A}(s)$ is obtained by reducing $\mathcal{A}(t_i) \overset{\mathcal{A}(\mathcal{Q})}{\to}_{\mathcal{A}(\mathcal{R})} \mathcal{A}(s_i)$ then we obtain $t_i \overset{\mathcal{Q}}{\to}_{\mathcal{R}} s_i$ and hence, $f^{?_1}t_1{}^{?_2}\ldots{}^{?_i}t_i{}^{?_{i+1}}\ldots{}^{?_n}t_n \overset{\mathcal{Q}}{\to}_{\mathcal{R}} f^{?_1}t_1{}^{?_2}\ldots{}^{?_i}s_i{}^{?_{i+1}}\ldots{}^{?_n}t_n = s$. If we have a root reduction then $\mathcal{A}(t) = \mathcal{A}(\ell)\delta \to_{\mathcal{R}} \mathcal{A}(r)\delta = \mathcal{A}(s)$. By (i) one obtains the desired reduction.

$$\begin{array}{rcl} t & = & \mathcal{A}^{-1}(\mathcal{A}(t)) \\ & = & \mathcal{A}^{-1}(\mathcal{A}(\ell)\delta) \\ & = & \mathcal{A}^{-1}(\mathcal{A}(\ell))\mathcal{A}^{-1}(\delta) \\ & = & \ell\mathcal{A}^{-1}(\delta) \\ & \overset{\mathcal{Q}}{\rightarrow}_{\mathcal{R}} & r\mathcal{A}^{-1}(\delta) \\ & = & \mathcal{A}^{-1}(\mathcal{A}(r))\mathcal{A}^{-1}(\delta) \\ & = & \mathcal{A}^{-1}(\mathcal{A}(r)\delta) \\ & = & \mathcal{A}^{-1}(\mathcal{A}(s)) \\ & = & s \end{array}$$

That the reduction really respects the evaluation strategy is due to the fact that $\mathcal{Q}$ is proper and that every $t_i \in \mathit{NF}(\mathcal{Q})$ by (iii). ☐

Proof of Lemma 6.6. We define ι as the substitution which inverses the effect of $\mathcal{Y}$. The domain of ι are those variables $\bot_t$ of $\mathcal{V}'$ that are in the image of $\mathcal{Y}$ and it is defined by $\iota(\bot_t) = t$.

(i) The claim can be proven by straightforward structural induction on t.

(ii) We perform structural induction on t. If $t = x$ then $\mathcal{Y}(t\sigma) = \mathcal{Y}(\sigma(x)) = \mathcal{Y}(\sigma)(x) = t\mathcal{Y}(\sigma)$. Otherwise, $t = f^{?_1}t_1{}^{?_2}\dots{}^{?_n}t_n$ and we conclude

$$\begin{array}{rcl} \mathcal{Y}(t\sigma) & = & \mathcal{Y}(f^{?_1}t_1\sigma^{?_2}\dots{}^{?_n}t_n\sigma) \\ & = & f^{?_1}\mathcal{Y}(t_1\sigma)^{?_2}\dots{}^{?_n}\mathcal{Y}(t_n\sigma) \\ \text{(by ind.)} & = & f^{?_1}t_1\mathcal{Y}(\sigma)^{?_2}\dots{}^{?_n}t_n\mathcal{Y}(\sigma) \\ & = & (f^{?_1}t_1{}^{?_2}\dots{}^{?_n}t_n)\mathcal{Y}(\sigma) \\ & = & t\mathcal{Y}(\sigma) \end{array}$$

(iii) Suppose $\mathcal{Y}(t)$ is not in $\mathcal{Q}$-normal form. Then obviously no instance of $\mathcal{Y}(t)$ is in $\mathcal{Q}$-normal form. Thus, $t = \mathcal{Y}(t)\iota$ is not in $\mathcal{Q}$-normal form which proves (iii).

(iv) We only show that $t\sigma \overset{\mathcal{Q}}{\rightarrow}_{\mathcal{R}} s$ implies $\mathcal{Y}(t\sigma) \overset{\mathcal{Q}}{\rightarrow}_{\mathcal{R}} \mathcal{Y}(s)$ and $s = u\delta$ for a proper term u and a $\mathcal{Q}$-normal substitution δ. Then the required result follows by induction on the number of reduction steps.

As σ is $\mathcal{Q}$-normal and $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$, t is no variable. Thus, $t = f^{?_1}t_1{}^{?_2}\dots{}^{?_n}t_n$. If s is obtained by reducing $t_i\sigma \overset{\mathcal{Q}}{\rightarrow}_{\mathcal{R}} s_i$, i.e., $s = f^{?_1}t_1{}^{?_2}\dots{}^{?_i}s_i{}^{?_{i+1}}\dots{}^{?_n}t_n$, then by the induction hypothesis we conclude $\mathcal{Y}(t_i\sigma) \overset{\mathcal{Q}}{\rightarrow}_{\mathcal{R}} \mathcal{Y}(s_i)$ and $s_i = u_i\delta$ for some $\mathcal{Q}$-normal substitution δ and proper term u_i. We may assume that u_i is variable disjoint from t_j for all $j \neq i$. Then we can extend δ to behave like σ on the variables of t_j for all $j \neq i$. Hence, for $u = f^{?_1}t_1{}^{?_2}\dots{}^{?_i}u_i{}^{?_{i+1}}\dots{}^{?_n}t_n$ we obtain $u\delta = s$. Moreover,

$$\begin{array}{rcl} \mathcal{Y}(t\sigma) & = & f^{?_1}\mathcal{Y}(t_1\sigma)^{?_2}\dots{}^{?_i}\mathcal{Y}(t_i\sigma)^{?_{i+1}}\dots{}^{?_n}\mathcal{Y}(t_n\sigma) \\ & \overset{\mathcal{Q}}{\rightarrow}_{\mathcal{R}} & f^{?_1}\mathcal{Y}(t_1\sigma)^{?_2}\dots{}^{?_i}\mathcal{Y}(u_i\delta)^{?_{i+1}}\dots{}^{?_n}\mathcal{Y}(t_n\sigma) \\ & = & f^{?_1}\mathcal{Y}(t_1\delta)^{?_2}\dots{}^{?_i}\mathcal{Y}(u_i\delta)^{?_{i+1}}\dots{}^{?_n}\mathcal{Y}(t_n\delta) \\ & = & \mathcal{Y}(f^{?_1}t_1\delta^{?_2}\dots{}^{?_i}u_i\delta^{?_{i+1}}\dots{}^{?_n}t_n\delta) \\ & = & \mathcal{Y}(u\delta) \\ & = & \mathcal{Y}(s). \end{array}$$

Otherwise, the reduction is on the root position, i.e., $t\sigma = \ell\tau \overset{\mathcal{Q}}{\rightarrow}_{\mathcal{R}} r\tau = s$ where $\ell = f^{?_1}\ell_1{}^{?_2}\dots{}^{?_n}\ell_n$. We choose $u = r$ and $\delta = \tau$ to obtain $s = u\delta$. Then $\mathcal{Y}(t\sigma) =$

$\mathcal{Y}(\ell\tau) = \ell\mathcal{Y}(\tau) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} r\mathcal{Y}(\tau) = \mathcal{Y}(r\tau) = \mathcal{Y}(s)$ by (ii). This reduction indeed respects the evaluation strategy given by $\mathcal{Q}$ since all $\ell_i\tau$ are normal and by (iii) all $\mathcal{Y}(\ell_i\tau)$ are normal, too.

(v) Suppose $t\sigma$ would not be in $\mathcal{Q}$-normal form. Then $t\sigma$ can perform a $\mathcal{Q}$-restricted rewrite step w.r.t. the TRS $\mathcal{R}' = \{q \to \mathsf{a} \mid q \in \mathcal{Q}\}$ where a is some fresh constant. Hence, by (iv) we know that $\mathcal{Y}(t\sigma)$ can also perform this rewrite step with $\mathcal{R}'$. Thus, $\mathcal{Y}(t\sigma)$ is not in $\mathcal{Q}$-normal form in contradiction to the assumption.

(vi) It suffices to prove the one-step property that if σ is $\mathcal{Q}$-normal and t is proper then $\mathcal{Y}(t\sigma) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} u$ implies $u = \mathcal{Y}(s\delta)$ for some proper s and some $\mathcal{Q}$-normal substitution δ such that $t\sigma \xrightarrow{\mathcal{Q}}_{\mathcal{R}} s\delta$. Then any infinite reduction of $\mathcal{Y}(t\sigma)$ can easily be transformed into an infinite reduction of $t\sigma$: If $\mathcal{Y}(t\sigma)$ is not terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ then $\mathcal{Y}(t\sigma) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} u_1 \xrightarrow{\mathcal{Q}}_{\mathcal{R}} u_2 \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \ldots$. Using the one-step property we obtain proper terms s_i and $\mathcal{Q}$-normal substitutions δ_i such that $u_i = \mathcal{Y}(s_i\delta_i)$ and $t\sigma \xrightarrow{\mathcal{Q}}_{\mathcal{R}} s_1\delta_1 \xrightarrow{\mathcal{Q}}_{\mathcal{R}} s_2\delta_2 \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \ldots$ proving that $t\sigma$ is not terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$.

To prove the one-step property we perform induction on t. By (ii) we know that $\mathcal{Y}(t\sigma) = t\mathcal{Y}(\sigma)$. Moreover, by (iii) $\mathcal{Y}(\sigma)$ is a $\mathcal{Q}$-normal substitution. Thus, $t \notin \mathcal{V}$, but $t = f^{?_1}t_1{}^{?_2}\ldots{}^{?_n}t_n$. We have $\mathcal{Y}(t\sigma) = f^{?_1}\mathcal{Y}(t_1\sigma)^{?_2}\ldots{}^{?_n}\mathcal{Y}(t_n\sigma)$. If the reduction was in the i-th argument (i.e., $\mathcal{Y}(t_i\sigma) \xrightarrow{\mathcal{Q}}_{\mathcal{R}} u_i$), then by the induction hypothesis we know that there are s_i and δ such that $t_i\sigma \xrightarrow{\mathcal{Q}}_{\mathcal{R}} s_i\delta$ and $u_i = \mathcal{Y}(s_i\delta)$. W.l.o.g. we may assume that the variables of s_i do not occur in t. Thus, we can extend δ to behave like σ on the variables of t. By choosing $s = f^{?_1}t_1{}^{?\ldots?_i}s_i{}^{?_{i+1}}\ldots{}^{?_n}t_n$ we obtain

$$\begin{array}{rcl} t\sigma & = & f^{?_1}t_1\sigma^{?_2}\ldots{}^{?_i}t_i\sigma^{?_{i+1}}\ldots{}^{?_n}t_n\sigma \\ & = & f^{?_1}t_1\delta^{?_2}\ldots{}^{?_i}t_i\sigma^{?_{i+1}}\ldots{}^{?_n}t_n\delta \\ & \xrightarrow{\mathcal{Q}}_{\mathcal{R}} & f^{?_1}t_1\delta^{?_2}\ldots{}^{?_i}s_i\delta^{?_{i+1}}\ldots{}^{?_n}t_n\delta \\ & = & s\delta \end{array}$$

Moreover, the equality $u = \mathcal{Y}(s\delta)$ is also satisfied.

$$\begin{array}{rcl} u & = & f^{?_1}\mathcal{Y}(t_1\sigma)^{?_2}\ldots{}^{?_i}u_i{}^{?_{i+1}}\ldots{}^{?_n}\mathcal{Y}(t_n\sigma) \\ & = & f^{?_1}\mathcal{Y}(t_1\delta)^{?_2}\ldots{}^{?_i}\mathcal{Y}(s_i\delta)^{?_{i+1}}\ldots{}^{?_n}\mathcal{Y}(t_n\delta) \\ & = & \mathcal{Y}(s\delta) \end{array}$$

Otherwise, $\mathcal{Y}(t\sigma) = \ell\tau \xrightarrow{\mathcal{Q}}_{\mathcal{R}} r\tau = u$. Thus, $\ell = f^{?_1}\ell_1{}^{?_2}\ldots{}^{?_n}\ell_n$ and $\mathcal{Y}(t_i\sigma) = \ell_i\tau$. As we had an $\mathcal{Q}$-restricted rewriting step all $\mathcal{Y}(t_i\sigma)$ are $\mathcal{Q}$-normal. By (v) we know that all terms $t_i\sigma$ are $\mathcal{Q}$-normal as well. Hence, as $\mathcal{Q}$ is proper every reduction of $t\sigma$ respects the evaluation strategy given by $\mathcal{Q}$. As $\mathcal{Y}(t\sigma) = \ell\tau$ we obtain $t\sigma = \iota(\mathcal{Y}(t\sigma)) = \iota(\ell\tau) = \ell\tau\iota$. By choosing $s = r$ and $\delta = \tau\iota$, we obtain $t\sigma = \ell\delta \xrightarrow{\mathcal{Q}}_{\mathcal{R}} r\delta = s\delta$. As the subterms of $\ell\delta$ are in $\mathcal{Q}$-normal form we also know that δ is a $\mathcal{Q}$-normal substitution. Moreover, we know that $\mathcal{Y}(s\delta) = \mathcal{Y}(r\tau\iota) = \mathcal{Y}(u\iota)$.

Hence, it remains to show that $\mathcal{Y}(u\iota) = u$. To this end, we first prove that u is proper. By (i) $\mathcal{Y}(t\sigma)$ is proper and as ℓ is proper and $\mathcal{Y}(t\sigma) = \ell\tau$ we know that $\tau(x)$ is proper for all $x \in \mathcal{V}(\ell) \supseteq \mathcal{V}(r)$. Hence, as r is proper the term $u = r\tau$ must be proper, too.

Now to finish the proof one can show $\mathcal{Y}(u\iota) = u$ for every proper term u by induction on u. First, if u is a variable then there are two cases. If u is not in the domain of ι then $\mathcal{Y}(u\iota) = \mathcal{Y}(u) = u$. Otherwise, $u = \bot_v$ for some v with $\mathcal{Y}(v) = u$. Hence, $\mathcal{Y}(u\iota) = \mathcal{Y}(v) = u$. Finally, if u is no variable then $u = f^{?_1}u_1{}^{?_2}\dots{}^{?_n}u_n$ and we conclude $\mathcal{Y}(u\iota) = \mathcal{Y}(f^{?_1}u_1\iota^{?_2}\dots{}^{?_n}u_n\iota) = f^{?_1}\mathcal{Y}(u_1\iota)^{?_2}\dots{}^{?_n}\mathcal{Y}(u_n\iota) = f^{?_1}u_1{}^{?_2}\dots{}^{?_n}u_n = u$ by induction. □

Proof of Theorem 6.8. We first prove soundness of the processor. Let $s_1 \to t_1, s_2 \to t_2, \dots$ be an infinite (minimal) $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain using the substitution σ. Hence, all $s_i\sigma$ are $\mathcal{Q}$-normal and $t_i\sigma \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} s_{i+1}\sigma$. Using Lemma 6.6 (ii) and (iv) we conclude

$$t_i\mathcal{Y}(\sigma) = \mathcal{Y}(t_i\sigma) \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} \mathcal{Y}(s_{i+1}\sigma) = s_{i+1}\mathcal{Y}(\sigma)$$

where t_i, s_{i+1}, and $\mathcal{Y}(\sigma)$ are proper by the requirements on the DP problem and by Lemma 6.6 (i). Using Lemma 6.6 (iii) we know that all $s_i\mathcal{Y}(\sigma)$ are in $\mathcal{Q}$-normal form and whenever $t_i\sigma$ is terminating then so is $t_i\mathcal{Y}(\sigma)$ by Lemma 6.6 (vi). Now we can apply Lemma 6.4 to obtain

$$\mathcal{A}(t_i)\mathcal{A}(\mathcal{Y}(\sigma)) = \mathcal{A}(t_i\mathcal{Y}(\sigma)) \overset{\mathcal{A}(\mathcal{Q})}{\to}{}^{*}_{\mathcal{A}(\mathcal{R})} \mathcal{A}(s_{i+1}\mathcal{Y}(\sigma)) = \mathcal{A}(s_{i+1})\mathcal{A}(\mathcal{Y}(\sigma))$$

Moreover, using Lemma 6.4 (iii) and (iv) we know that every $\mathcal{A}(s_i)\mathcal{A}(\mathcal{Y}(\sigma))$ is in $\mathcal{A}(\mathcal{Q})$-normal form and whenever $t_i\mathcal{Y}(\sigma)$ is terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ then $\mathcal{A}(t_i)\mathcal{A}(\mathcal{Y}(\sigma))$ is terminating w.r.t. $\overset{\mathcal{A}(\mathcal{Q})}{\to}_{\mathcal{A}(\mathcal{R})}$. Thus, $\mathcal{A}(s_1) \to \mathcal{A}(t_1), \mathcal{A}(s_2) \to \mathcal{A}(t_2), \dots$ is an infinite (minimal) $(\mathcal{A}(\mathcal{P}), \mathcal{A}(\mathcal{Q}), \mathcal{A}(\mathcal{R}))$-chain as desired.

For completeness we have to show two results. First, if $\overset{\mathcal{A}(\mathcal{Q})}{\to}_{\mathcal{A}(\mathcal{R})}$ is non-terminating then $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ must be non-terminating. To prove this it suffices to apply Lemma 6.4 (iv): if $t_1 \overset{\mathcal{A}(\mathcal{Q})}{\to}_{\mathcal{A}(\mathcal{R})} t_2 \overset{\mathcal{A}(\mathcal{Q})}{\to}_{\mathcal{A}(\mathcal{R})} \dots$ is an infinite reduction then $\mathcal{A}^{-1}(t_1) \overset{\mathcal{Q}}{\to}_{\mathcal{R}} \mathcal{A}^{-1}(t_2) \overset{\mathcal{Q}}{\to}_{\mathcal{R}} \dots$ is an infinite reduction.

Second, if there is an infinite $(\mathcal{A}(\mathcal{P}), \mathcal{A}(\mathcal{Q}), \mathcal{A}(\mathcal{R}))$-chain, i.e., if there is a substitution σ with $\mathcal{A}(t_i)\sigma \overset{\mathcal{A}(\mathcal{Q})}{\to}{}^{*}_{\mathcal{A}(\mathcal{R})} \mathcal{A}(s_{i+1})\sigma$ for pairs $s_i \to t_i \in \mathcal{P}$ with $\mathcal{A}(s_i)\sigma \in \mathit{NF}(\overset{\mathcal{A}(\mathcal{Q})}{\to}_{\mathcal{A}(\mathcal{R})})$ then again using Lemma 6.4 shows that $s_1 \to t_1, s_2 \to t_2, \dots$ is an infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain using the substitution $\mathcal{A}^{-1}(\sigma)$. □

Proof of Lemma 6.11. (i) We prove the claim by induction on t using the embedding order as induction relation. If $\mathcal{Z}(t)$ is a variable then it is proper. Otherwise, $t = f^{?_1}t_1{}^{?_2}\dots{}^{?_k}t_k$ with $n = \mathit{a}\text{-}\mathit{ar}(f)$ and $k \geq n$. By the induction we know that $\mathcal{Z}(t_i{}^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k)$ is proper for every $1 \leq i \leq n$. Thus, we also know that

$$\mathcal{Z}(t) = f^{?_1}\mathcal{Z}(t_1{}^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k)^{?_2}\dots{}^{?_n}\mathcal{Z}(t_n{}^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k)$$

is proper.

(ii) We perform induction on t. If t is a variable x, then $\mathcal{Z}(\sigma(x)^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k) = \mathcal{Z}(\overline{\sigma}(x)) = x\mathcal{Z}(\overline{\sigma})$ by definition of $\overline{\sigma}$.

Otherwise, $t = f^{?_1}t_1{}^{?_2}\dots{}^{?_n}t_n$, and we conclude

$$\begin{aligned}
& \mathcal{Z}(t\sigma^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k) \\
&= \mathcal{Z}(f^{?_1}t_1\sigma^{?_2}\dots{}^{?_n}t_n\sigma^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k) \\
&= f^{?_1}\mathcal{Z}(t_1\sigma^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k)^{?_2}\dots{}^{?_n}\mathcal{Z}(t_n\sigma^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k) \\
\text{(ind.)} &= f^{?_1}t_1\mathcal{Z}(\overline{\sigma})^{?_2}\dots{}^{?_n}t_n\mathcal{Z}(\overline{\sigma}) \\
&= t\mathcal{Z}(\overline{\sigma})
\end{aligned}$$

(iii) It suffices to show that $t \to_{\mathcal{R}} s$ implies $\mathcal{Z}(t) \to_{\mathcal{R}}^* \mathcal{Z}(s)$. We use induction on t with the embedding order as induction relation. Obviously, $t \notin \mathcal{V}$. First assume that $t = x^{?_1}t_1{}^{?_2}\dots{}^{?_k}t_k$ or $t = f^{?_1}t_1{}^{?_2}\dots{}^{?_k}t_k$ where $k < a\text{-}ar(f)$. Here, s is obtained by reducing $t_i \to_{\mathcal{R}} s_i$ for some i, since $\mathcal{R}$ is proper. Thus, $\mathcal{Z}(t) = \bot = \mathcal{Z}(s)$.

In the remainder we can assume $t = f^{?_1}t_1{}^{?_2}\dots{}^{?_k}t_k$, where $a\text{-}ar(f) = n$ and $k \geq n$. First we consider the case that s is obtained by reducing t_i to s_i. If $i \leq n$, then $\mathcal{Z}(t_i{}^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k) \to_{\mathcal{R}}^* \mathcal{Z}(s_i{}^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k)$ by induction. Hence, $\mathcal{Z}(t) \to_{\mathcal{R}}^* \mathcal{Z}(s)$ by the definition of $\mathcal{Z}$:

$$\begin{array}{rl}
 & \mathcal{Z}(t)\\
= & f^{?_1}\mathcal{Z}(t_1{}^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k)^{?_2}\dots{}^{?_i}\mathcal{Z}(t_i{}^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k)^{?_{i+1}}\\
 & \dots{}^{?_n}\mathcal{Z}(t_n{}^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k)\\
\to_{\mathcal{R}}^* & f^{?_1}\mathcal{Z}(t_1{}^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k)^{?_2}\dots{}^{?_i}\mathcal{Z}(s_i{}^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k)^{?_{i+1}}\\
 & \dots{}^{?_n}\mathcal{Z}(t_n{}^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k)\\
= & \mathcal{Z}(s)
\end{array}$$

Otherwise, $i > n$ and by the induction hypothesis we conclude that there is the reduction $\mathcal{Z}(t_j{}^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_i}t_i{}^{?_{i+1}}\dots{}^{?_k}t_k) \to_{\mathcal{R}}^* \mathcal{Z}(t_j{}^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_i}s_i{}^{?_{i+1}}\dots{}^{?_k}t_k)$ for all $1 \leq j \leq n$. By the definition of $\mathcal{Z}$ we again get $\mathcal{Z}(t) \to_{\mathcal{R}}^* \mathcal{Z}(s)$:

$$\begin{array}{rcl}
\mathcal{Z}(t) & = & f^{?_1}\mathcal{Z}(t_1{}^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_i}t_i{}^{?_{i+1}}\dots{}^{?_k}t_k)^{?_2}\dots\\
 & & {}^{?_n}\mathcal{Z}(t_n{}^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_i}t_i{}^{?_{i+1}}\dots{}^{?_k}t_k)\\
 & \to_{\mathcal{R}}^* & f^{?_1}\mathcal{Z}(t_1{}^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_i}s_i{}^{?_{i+1}}\dots{}^{?_k}t_k)^{?_2}\dots\\
 & & {}^{?_n}\mathcal{Z}(t_n{}^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_i}s_i{}^{?_{i+1}}\dots{}^{?_k}t_k)\\
 & = & \mathcal{Z}(s)
\end{array}$$

Finally we have to consider the case that the reduction is not in one of terms t_i. Then, $t = \ell\sigma^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k \to_{\mathcal{R}} r\sigma^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k = s$. Let $\overline{\sigma}$ be the substitution with $\overline{\sigma}(x) = \sigma(x)^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k$. Hence,

$$\begin{array}{rcl}
\mathcal{Z}(t) & = & \mathcal{Z}(\ell\sigma^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k)\\
\text{(by } (ii)\text{)} & = & \ell\mathcal{Z}(\overline{\sigma})\\
 & \to_{\mathcal{R}} & r\mathcal{Z}(\overline{\sigma})\\
\text{(by } (ii)\text{)} & = & \mathcal{Z}(r\sigma^{?_{n+1}}t_{n+1}{}^{?_{n+2}}\dots{}^{?_k}t_k)\\
 & = & \mathcal{Z}(s)
\end{array}$$

□

Proof of Theorem 6.12. We first prove soundness of the processor. If $s_1 \to t_1, s_2 \to t_2, \dots$ is an infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain then there is a substitution σ with $t_i\sigma \overset{\mathcal{Q}}{\to}{}_{\mathcal{R}}^* s_{i+1}\sigma$ for all i. Hence, by Lemma 2.4 we also have $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$. Thus, using Lemma 6.11 we know that $t_i\mathcal{Z}(\sigma) = \mathcal{Z}(t_i\sigma) \to_{\mathcal{R}}^* \mathcal{Z}(s_{i+1}\sigma) = s_{i+1}\mathcal{Z}(\sigma)$ where $\mathcal{Z}(\sigma)$ instantiates all variables with proper terms. Now we can use Lemma 6.4 to conclude

$$\mathcal{A}(t_i)\mathcal{A}(\mathcal{Z}(\sigma)) = \mathcal{A}(t_i\mathcal{Z}(\sigma)) \to_{\mathcal{A}(\mathcal{R})}^* \mathcal{A}(s_{i+1}\mathcal{Z}(\sigma)) = \mathcal{A}(s_{i+1})\mathcal{A}(\mathcal{Z}(\sigma))$$

Thus, $\mathcal{A}(s_1) \to \mathcal{A}(t_1), \mathcal{A}(s_2) \to \mathcal{A}(t_2), \dots$ is an infinite $(\mathcal{A}(\mathcal{P}), \varnothing, \mathcal{A}(\mathcal{R}))$-chain as desired.

For completeness we have to show two results. First, if $\mathcal{A}(\mathcal{R})$ is non-terminating then $\mathcal{R}$ must be non-terminating. This was already proven in [KKSV96] and can also be shown by using Lemma 6.4 (iv): if $t_1 \to_{\mathcal{A}(\mathcal{R})} t_2 \to_{\mathcal{A}(\mathcal{R})} \dots$ is an infinite $\mathcal{A}(\mathcal{R})$-reduction then $\mathcal{A}^{-1}(t_1) \to_{\mathcal{R}} \mathcal{A}^{-1}(t_2) \to_{\mathcal{R}} \dots$ is an infinite reduction w.r.t. $\mathcal{A}^{-1}(\mathcal{A}(\mathcal{R})) = \mathcal{R}$.

Second, if there is an infinite $(\mathcal{A}(\mathcal{P}), \varnothing, \mathcal{A}(\mathcal{R}))$-chain, i.e., if there is a substitution σ with $\mathcal{A}(t_i)\sigma \to^*_{\mathcal{A}(\mathcal{R})} \mathcal{A}(s_{i+1})\sigma$ for pairs $s_i \to t_i \in \mathcal{P}$ then again using Lemma 6.4 shows that $s_1 \to t_1, s_2 \to t_2, \dots$ is an infinite $(\mathcal{P}, \varnothing, \mathcal{R})$-chain using the substitution $\mathcal{A}^{-1}\sigma$. $\square$

Proof of Lemma 6.16. Note that by construction of $\mathcal{S}_{\mathit{all}}$ and σ all terms in $\mathcal{S}_{\mathit{all}}\sigma$ are $\mathcal{Q}$-normal. The proof is very similar to the proof of Lemma 4.11. Therefore it is often just referred to that proof. The main new difficulty is to show that if a term $f^{?_1}t_1{}^{?_2}\dots{}^{?_n}t_n$ is transformed by the second case that then every reduction is either on the root level or inside some t_i.

(i) We perform induction on t. If t is a variable x then by definition $\mathcal{I}'(t\sigma) = \mathcal{I}'(x\sigma) = x\mathcal{I}'(\sigma) = \mathcal{A}(t)\mathcal{I}'(\sigma)$. Otherwise, let $t = f^{?_1}t_1{}^{?_2}\dots{}^{?_n}t_n$. Then we obtain $\mathcal{N}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t_i) \subseteq \mathcal{N}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{N}$ and all t_i are proper. Hence, by induction we conclude $\mathcal{I}'(t_i\sigma) = \mathcal{A}(t_i)\mathcal{I}'(\sigma)$. If $\mathcal{I}'(t\sigma)$ is built by the second case we immediately get $\mathcal{I}'(t\sigma) = f_{?_1\dots?_n}(\mathcal{A}(t_1)\mathcal{I}'(\sigma), \dots, \mathcal{A}(t_n)\mathcal{I}'(\sigma)) = \mathcal{A}(t)\mathcal{I}'(\sigma)$. Otherwise, there must be a rule $\ell \to r \notin \mathcal{N}$ and an i such that $\mathit{Cap}^{\mathcal{S}_{\mathit{all}}\sigma}_{\mathcal{R},\mathcal{Q}}(f^{?_1}t_1\sigma^{?_2}\dots{}^{?_{i-1}}t_{i-1})^{?_i}\,\mathit{Cap}^{\mathcal{S}_{\mathit{all}}\sigma}_{\mathcal{R},\mathcal{Q}}(t_i\sigma)$ unifies with ℓ by some mgu μ with $(\mathcal{S}_{\mathit{all}}\sigma \cup \{\ell_1, \dots, \ell_k\})\mu \subseteq \mathit{NF}(\mathcal{Q})$. In the same way as in the proof of Lemma 4.11 one can show that $\ell \to r \in \mathcal{N}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(f^{?_1}t_1{}^{?_2}\dots{}^{?_i}t_i)$. Hence, by Definition 4.5 the rule $\ell \to r$ is also contained in $\mathcal{N}^{\mathcal{S}}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{N}$ as $f^{?_1}t_1{}^{?_2}\dots{}^{?_i}t_i$ is a subterm of t. This is a contradiction to $\ell \to r \notin \mathcal{N}$.

(ii) The claim is proven in the same way as in Lemma 4.11.

(iii) As in Lemma 4.11 the proof is a direct consequence of Lemma 4.7.

(iv) Let $t = \ell\sigma \xrightarrow{\mathcal{Q}}_{\mathcal{R}} r\sigma = s$ for some rule $\ell \to r \in \mathcal{R}$. In the same way as in Lemma 4.11 one can show that $\ell \to r \in \mathcal{N}$. Hence, $\ell = f^{?_1}\ell_1{}^{?_2}\dots{}^{?_n}\ell_n$ and r are proper because $\mathcal{N}$ is proper.

Now by the closure properties of $\mathcal{N}$ we obtain $\mathcal{N}^{\{f^{?_1}\ell_1{}^{?_2}\dots{}^{?_{n-1}}\ell_{n-1}, \ell_n\}}_{\mathcal{R},\mathcal{Q}}(r) \subseteq \mathcal{N}$. And by the construction of $\mathcal{S}_{\mathit{all}}$ we get $\{f^{?_1}\ell_1{}^{?_2}\dots{}^{?_{n-1}}\ell_{n-1}, \ell_n\} \subseteq \mathcal{S}_{\mathit{all}}$. Hence, (ii) and (i) yield $\mathcal{I}'(t) = \mathcal{I}'(\ell\sigma) \to^*_{\mathcal{C}_\varepsilon} \mathcal{A}(\ell)\mathcal{I}'(\sigma) \to_{\mathcal{A}(\mathcal{N})} \mathcal{A}(r)\mathcal{I}'(\sigma) = \mathcal{I}'(r\sigma)$.

(v) We perform induction on the structure of t. Obviously t cannot be a variable. If $\mathcal{I}'(t)$ is built by the third or forth case we use (iii). So, let $t = f^{?_1}t_1{}^{?_2}\dots{}^{?_n}t_n$ where $\mathcal{I}'(t) = f_{?_1\dots?_n}(\mathcal{I}'(t_1), \dots, \mathcal{I}'(t_n))$. If we have a root reduction we use (iv) to obtain $\mathcal{I}'(t) \to^+_{\mathcal{N}\cup\mathcal{C}_\varepsilon} \mathcal{I}'(s)$. If the reduction is below the root then we consider two cases.

First, the reduction can be inside some t_j. Then $t_j \xrightarrow{\mathcal{Q}}_{\mathcal{R}} s_j$ and if we define $s_i = t_i$ for all $i \neq j$ then $s = f^{?_1}s_1{}^{?_2}\dots{}^{?_j}s_j{}^{?_{j+1}}\dots{}^{?_n}s_n$. Induction yields $\mathcal{I}'(t_j) \to^+_{\mathcal{A}(\mathcal{N})\cup\mathcal{C}_\varepsilon} \mathcal{I}'(s_j)$ and hence, $\mathcal{I}'(t) \to^+_{\mathcal{A}(\mathcal{N})\cup\mathcal{C}_\varepsilon} f_{?_1\dots?_n}(\mathcal{I}'(s_1), \dots, \mathcal{I}'(s_j), \dots, \mathcal{I}'(s_n)) =: w$. It remains to show $\mathcal{I}'(s) = w$, i.e., we have to prove that $\mathcal{I}'(s)$ is built by the second case. To this end we use $t_i \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} s_i$ and $f^{?_1}t_1{}^{?_2}\dots{}^{?_{i-1}}t_{i-1} \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} f^{?_1}s_1{}^{?_2}\dots{}^{?_{i-1}}s_{i-1}$ for all i to conclude by Lemma 3.8 that $\mathit{Cap}^{\mathcal{S}_{\mathit{all}}\sigma}_{\mathcal{R},\mathcal{Q}}(f^{?_1}s_1{}^{?_2}\dots{}^{?_{i-1}}s_{i-1})^{?_i}\,\mathit{Cap}^{\mathcal{S}_{\mathit{all}}\sigma}_{\mathcal{R},\mathcal{Q}}(s_i)$ is an instance of $\mathit{Cap}^{\mathcal{S}_{\mathit{all}}\sigma}_{\mathcal{R},\mathcal{Q}}(f^{?_1}t_1{}^{?_2}\dots{}^{?_{i-1}}t_{i-1})^{?_i}\,\mathit{Cap}^{\mathcal{S}_{\mathit{all}}\sigma}_{\mathcal{R},\mathcal{Q}}(t_i)$ where only variables that are introduced by *Cap* are instantiated. Thus, as t is transformed by the second case this must also be the case for s.

Finally, if the reduction is not in some t_j and also not at the root level then there is some $i < n$ such that $f^{?_1}t_1{}^{?_2}\dots{}^{?_i}t_i$ is the redex $\ell\sigma$ and $s = r\sigma^{?_{i+1}}t_{i+1}{}^{?_{i+2}}\dots{}^{?_n}t_n$

for some rule $\ell \to r$. Note that $\ell \to r$ cannot be contained in the proper TRS $\mathcal{N}$ as ℓ is not proper because $i < n$. This results in a contradiction as one can prove similarly as in (iv) that $\ell \to r \in \mathcal{N}$. The reason is that $\mathcal{I}'(t)$ is built by the second case, that $Cap^{\mathcal{S}_{all}\sigma}_{\mathcal{R},\mathcal{Q}}(f^{?_1}t_1{}^{?_2}\dots{}^{?_{i-1}}t_{i-1})^{?_i}Cap^{\mathcal{S}_{all}\sigma}_{\mathcal{R},\mathcal{Q}}(t_i)$ unifies with ℓ, and that the mgu satisfies the required normal form conditions.

(vi) As $u_1 \to v_1, u_2 \to v_2, \dots$ is a minimal chain for every i we know that $v_i\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} u_{i+1}\sigma$ and $u_i\sigma \in NF(\mathcal{Q})$. Moreover, as $\mathcal{P}$ is proper every u_i and v_i are proper. Using the previous results we can now exchange $\mathcal{R}$ by $\mathcal{A}(\mathcal{N}) \cup \mathcal{C}_\varepsilon$ and σ by $\mathcal{I}'(\sigma)$ similar to Figure 4.8.

By construction of $\mathcal{N}$ we know that $\mathcal{N}^{\{u_i\}}_{\mathcal{R},\mathcal{Q}}(v_i) \subseteq \mathcal{N}$ and $u_i \in \mathcal{S}_{all}$ for every i. Hence, using (i) we obtain $\mathcal{I}'(v_i\sigma) = \mathcal{A}(v_i)\mathcal{I}'(\sigma)$. From (ii) we conclude $\mathcal{I}'(u_i\sigma) \to^*_{\mathcal{C}_\varepsilon} \mathcal{A}(u_i)\mathcal{I}'(\sigma)$. By (v) we glue everything together and obtain

$$\mathcal{A}(v_i)\mathcal{I}'(\sigma) = \mathcal{I}'(v_i\sigma) \to^*_{\mathcal{A}(\mathcal{N})\cup\mathcal{C}_\varepsilon} \mathcal{I}'(u_{i+1}\sigma) \to^*_{\mathcal{C}_\varepsilon} \mathcal{A}(u_{i+1})\mathcal{I}'(\sigma)$$

Thus, $\mathcal{A}(u_1) \to \mathcal{A}(v_1), \mathcal{A}(u_2) \to \mathcal{A}(v_2), \dots$ is an infinite $(\mathcal{A}(\mathcal{P}), \varnothing, \mathcal{A}(\mathcal{N}) \cup \mathcal{C}_\varepsilon)$-chain. □

Proof of Theorem 6.17. We first consider completeness. In the cases (B) and (D) we just have to apply Lemma 2.17. In case (C) an additional application of Lemma 6.4 is sufficient to prove completeness.

For soundness we consider an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain $s_1 \to t_1, s_2 \to t_2, \dots$ where σ is the substitution with $t_i\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} s_{i+1}\sigma$. As $\mathcal{P} \cup \mathcal{N}$ is proper we just have to apply Lemma 6.16 (vi) to show that $\mathcal{A}(s_1) \to \mathcal{A}(t_1), \mathcal{A}(s_2) \to \mathcal{A}(t_2), \dots$ is an infinite $(\mathcal{A}(\mathcal{P}), \varnothing, \mathcal{A}(\mathcal{N}) \cup \mathcal{C}_\varepsilon)$-chain. Hence, we have already proven soundness of case (A).

Next we consider case (D). Due to the constraints on the well-founded order there must be some n such that there is no pair $s_i \to t_i$ with $i \geq n$ and $\mathcal{A}(s_i) \succ \mathcal{A}(t_i)$. Hence, $s_n \to t_n, s_{n+1} \to t_{n+1}, \dots$ is an infinite $(\mathcal{P} \setminus \{s \to t \mid \mathcal{A}(s) \succ \mathcal{A}(t)\}, \mathcal{Q}, \mathcal{R}, f)$-chain.

For case (B) we just have to adapt and combine the proofs of Theorem 4.20 and 4.22. Due to the constraints there is some n such that in every reduction $\mathcal{A}(t_i)\mathcal{I}'(\sigma) = \mathcal{I}'(t_i\sigma) \to^*_{\mathcal{A}(\mathcal{N})\cup\mathcal{C}_\varepsilon} \mathcal{I}'(s_{i+1}\sigma) \to^*_{\mathcal{C}_\varepsilon} \mathcal{A}(s_{i+1})\mathcal{I}'(\sigma)$ only reductions with $\mathcal{A}(\mathcal{N}')$ are performed. As in Theorem 4.20 we look a bit more careful at Lemma 6.16 to conclude that even in the original chain $s_n \to t_n, s_{n+1} \to t_{n+1}, \dots$ only rules of $\mathcal{N}'$ have been used. Moreover, if a term $t_i\sigma$ is terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ then it is also terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{N}'}$ by Lemma 2.4. Thus, $s_n \to t_n, s_{n+1} \to t_{n+1}, \dots$ is a minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{N}', f)$-chain.

Finally, to prove soundness of case (C) we again argue over details of the proof of Lemma 6.16. It turns out that there are no reductions below a c-symbol. Thus, even in the original chain $s_n \to t_n, s_{n+1} \to t_{n+1}, \dots$ one never reduced a term which is improper at the root-position, i.e., which is of the form $x^{?_1}u_1{}^{?_2}\dots{}^{?_u}{}_n$ or $f^{?_1}u_1{}^{?_2}\dots{}^{?_n}u_n$ where in the second case $a\text{-}ar(f) \neq n$. Thus, these terms u which are improper at the root-position can be seen as normal forms and can be replaced by fresh variables $\bot_u$. This is exactly done by the $\mathcal{Y}$-transformation. We give two arguments to prove that $s_n \to t_n, s_{n+1} \to t_{n+1}, \dots$ is also a $(\mathcal{P}, \mathcal{Q}, \mathcal{N}')$-chain using the substitution $\mathcal{Y}(\sigma)$.

First, we show that all reductions with $\mathcal{R}$ are still possible: We already know that no subterm u that has been replaced by a fresh variable $\bot_u$ is reduced. Additionally these terms u can only be matched by variables, i.e., all reductions above are also possible after applying $\mathcal{Y}$. And second, $\mathcal{Y}$ does not introduce additional $\mathcal{Q}$-redexes and thus, the strategy is still respected after $\mathcal{Y}$-transformation, cf. Lemma 6.6 (iii).

A more detailed proof of the desired result $t_i\mathcal{Y}(\sigma) \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{N}'} s_{i+1}\mathcal{Y}(\sigma)$ can be obtained by an adaptation of Lemma 6.6 (iv). One has to replace the requirement $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{N}')$ by the requirement that there are no reductions inside improper terms.

Finally, by Lemma 6.6 (ii) and (vi) we know that all terms $t_i\mathcal{Y}(\sigma) = \mathcal{Y}(t_i\sigma)$ are terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{N}'}$. Hence, $s_n \to t_n, s_{n+1} \to t_{n+1}, \dots$ is an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{N}')$-chain using the substitution $\mathcal{Y}(\sigma)$. Since $\mathcal{Y}(\sigma)$ instantiates all variables with proper terms, we can apply Lemma 6.4 to obtain that $\mathcal{A}(s_n) \to \mathcal{A}(t_n), \mathcal{A}(s_{n+1}) \to \mathcal{A}(t_{n+1}), \dots$ is an infinite minimal $(\mathcal{A}(\mathcal{P}), \mathcal{A}(\mathcal{Q}), \mathcal{A}(\mathcal{N}'))$-chain using the substitution $\mathcal{A}(\mathcal{Y}(\sigma))$. □

Proof of Theorem 6.22. One can reuse the proof of Theorem 4.32 where one has to replace Lemma 4.35 (vi) by Lemma 6.24 (vi). □

Proof of Lemma 6.24. Note that by construction of $\mathcal{S}_{\mathit{all}}$ and σ all terms in $\mathcal{S}_{\mathit{all}}\sigma$ are $\mathcal{Q}$-normal. The proof is very similar to the proof of Lemma 6.16. Therefore we only show the proofs of properties (i) and (v) which need additional argumentation.

(i) One can handle the variable case as in Lemma 6.16 (i) and one can also show as before that $\mathcal{I}'_\pi(t)$ cannot be build by the third or forth case.

So, let $t = f^{?_1}t_1{}^{?_2}\dots{}^{?_n}t_n$ where $\mathcal{I}'_\pi(t)$ is build by the second case. Then we know that $\mathcal{N}'^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t_i) \subseteq \mathcal{N}'^{\mathcal{S},\pi}_{\mathcal{R},\mathcal{Q}}(t) \subseteq \mathcal{N}$ and that t_i is π-proper for every $i \in \mathit{RegPos}_\pi(f_{?_1\dots?_n})$. Hence, by the induction hypothesis we obtain $\mathcal{I}'_\pi(t_i\sigma) = \mathcal{A}_\pi(t_i)\mathcal{I}'_\pi(\sigma)$ for all $i \in \mathit{RegPos}_\pi(f_{?_1\dots?_n})$. Concluding $\mathcal{I}'_\pi(t\sigma) = \overline{f_{?_1\dots?_n}(\mathcal{A}_\pi(t_1)\mathcal{I}'_\pi(\sigma), \dots, \mathcal{A}_\pi(t_n)\mathcal{I}'_\pi(\sigma))} = \mathcal{A}_\pi(t)\mathcal{I}'_\pi(\sigma)$[46] finishes this case.

(v) We perform induction on t. As in Lemma 6.16 (v) the only interesting case is when $\mathcal{I}'_\pi(t)$ is build by the second case. So, let $t = f^{?_1}t_1{}^{?_2}\dots{}^{?_n}t_n$ where $\mathcal{I}'_\pi(t) = \overline{f_{?_1\dots?_n}(\mathcal{I}'_\pi(t_1), \dots, \mathcal{I}'_\pi(t_n))}$. If we have a root reduction then we apply (iv) to obtain $\mathcal{I}'_\pi(t) \to^{+}_{\mathcal{N}\cup\mathcal{C}_\varepsilon} \mathcal{I}'_\pi(s)$. If the reduction is below the root then as in Lemma 6.16 one can show that the reduction must be in some t_j as $\mathcal{N}$ is π-proper. Then $t_j \xrightarrow{\mathcal{Q}}_{\mathcal{R}} s_j$ and if we define $s_i = t_i$ for all $i \neq j$ then $s = f^{?_1}s_1{}^{?_2}\dots{}^{?_j}s_j{}^{?_{j+1}}\dots{}^{?_n}s_n$.

The induction hypothesis yields $\mathcal{I}'_\pi(t_j) \to^{*}_{\mathcal{A}_\pi(\mathcal{N})\cup\mathcal{C}_\varepsilon} \mathcal{I}'_\pi(s_j)$ and hence, $\mathcal{I}'_\pi(t) \to^{*}_{\mathcal{A}_\pi(\mathcal{N})\cup\mathcal{C}_\varepsilon} \overline{f_{?_1\dots?_n}(\mathcal{I}'_\pi(s_1), \dots, \mathcal{I}'_\pi(s_j), \dots, \mathcal{I}'_\pi(s_n))} =: w$. It remains to show that $\mathcal{I}'_\pi(s) = w$, i.e., we have to prove that $\mathcal{I}'_\pi(s)$ is built by the second case. This can be done as in Lemma 6.16. □

A.7. Proofs of Chapter 7

Proof of Lemma 7.3. We consider the TRS $\mathcal{R} = \{q \to q \mid q \in \mathcal{Q}\}$. Then clearly $\mathcal{M}$ is a model of $\mathcal{R}$. We conclude

$$\begin{array}{rcl} & & t \notin \mathit{NF}(\mathcal{Q}) \\ & \Leftrightarrow & t \to_{\mathcal{R}} t \\ \text{(Lemma 7.2)} & \Rightarrow & \mathit{Lab}(t,\beta) \to_{\overline{\mathcal{R}}} \mathit{Lab}(t,\beta) \\ & \Leftrightarrow & \mathit{Lab}(t,\beta) \notin \mathit{NF}(\overline{\mathcal{R}}) = \mathit{NF}(\overline{\mathcal{Q}}) \end{array}$$

---

[46] Note that here we apply $\mathcal{A}_\pi$ on possibly non π-proper terms t_i, if $i \notin \mathit{RegPos}_\pi(f_{?_1\dots?_n})$. However, as these terms are not occurring in the resulting term $\overline{f_{?_1\dots?_n}(\dots)}$, we can ignore these applications.

and moreover,

$$\begin{array}{rll} & & \mathit{Lab}(t,\beta) \notin \mathit{NF}(\overline{\mathcal{Q}}) \\ & \Leftrightarrow & \mathit{Lab}(t,\beta) \to_{\overline{\mathcal{R}}} \mathit{Lab}(t,\beta) \\ (\text{Lemma 7.2}) & \Rightarrow & t = \mathit{Unlab}(\mathit{Lab}(t,\beta)) \to_{\mathcal{R}} \mathit{Unlab}(\mathit{Lab}(t,\beta)) \\ & \Leftrightarrow & t \notin \mathit{NF}(\mathcal{R}) = \mathit{NF}(\mathcal{Q}) \end{array} \qquad \square$$

Proof of Theorem 7.5. Let $s_1 \to t_1, s_2 \to t_2, \ldots$ be an infinite $(\mathcal{P},\mathcal{Q},\mathcal{R})$-chain. Thus, there is a substitution σ such that $t_i\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} s_{i+1}\sigma$ for every $i \in \mathbb{N}$. Let β be an arbitrary but fixed variable assignment. By Lemma 7.4 we obtain $\mathit{Lab}(t_i\sigma,\beta) \xrightarrow{\overline{\mathcal{Q}}}{}^{*}_{\overline{\mathcal{R}}} \mathit{Lab}(s_i\sigma,\beta)$ and by Lemma 7.3 all terms $\mathit{Lab}(s_i\sigma,\beta)$ are in normal form w.r.t. $\overline{\mathcal{Q}}$. We define the substitution $\overline{\sigma}$ as as $\overline{\sigma}(x) = \mathit{Lab}(\sigma(x),\beta)$ and the variable assignment $\overline{\beta}$ as $\overline{\beta}(x) = [\beta](\sigma(x))$. Then by [Zan95, Lemma 2] all equations $\mathit{Lab}(s_i\sigma,\beta) = \mathit{Lab}(s_i,\overline{\beta})\overline{\sigma}$ and $\mathit{Lab}(t_i\sigma,\beta) = \mathit{Lab}(t_i,\overline{\beta})\overline{\sigma}$ are satisfied and hence, $\mathit{Lab}(s_1,\overline{\beta}) \to \mathit{Lab}(t_1,\overline{\beta}), \mathit{Lab}(s_2,\overline{\beta}) \to \mathit{Lab}(t_2,\overline{\beta}), \ldots$ is an infinite $(\overline{\mathcal{P}},\overline{\mathcal{Q}},\overline{\mathcal{R}})$-chain. □

Proof of Lemma 7.9. (i) We perform induction on t. Let $t = f(t_1,\ldots,t_n)$ and let t_i be a subterm which contains x. If $t_i = x$ then $\mathit{Lab}(t,\beta)\sigma = f_{(\ldots,\beta(x),\ldots)}(\ldots,x,\ldots)\sigma \neq f_{(\ldots,\beta'(x),\ldots)}(\ldots,x,\ldots)\sigma' = \mathit{Lab}(t,\beta')\sigma'$. Otherwise, t_i is not a variable. By induction we know that $\mathit{Lab}(t_i,\beta)\sigma \neq \mathit{Lab}(t_i,\beta')\sigma'$. Hence, we can finish this part as follows.

$$\begin{array}{rcl} \mathit{Lab}(t,\beta)\sigma & = & f_{(\ldots)}(\ldots,\mathit{Lab}(t_i,\beta)\sigma,\ldots) \\ & \neq & f_{(\ldots)}(\ldots,\mathit{Lab}(t_i,\beta')\sigma',\ldots) \\ & = & \mathit{Lab}(t,\beta')\sigma' \end{array}$$

(ii) Let $\mathit{Lab}(t,\beta) \to_{\overline{\mathcal{R}}} s$ at position p by some rule $\mathit{Lab}(\ell,\beta') \to \mathit{Lab}(r,\beta') \in \overline{\mathcal{R}}$. Then by Lemma 7.2 $\mathit{Unlab}(\mathit{Lab}(t,\beta)) = t \to_{\ell\to r} t'$ is a reduction at position p and hence, again by Lemma 7.2 we obtain the reduction $\mathit{Lab}(t,\beta) \to_{\mathit{Lab}(\ell,\beta'')\to \mathit{Lab}(r,\beta'')} \mathit{Lab}(t',\beta)$ at position p[47]. Thus, $\mathit{Lab}(t,\beta)|_p = \mathit{Lab}(\ell,\beta')\mu_1 = \mathit{Lab}(\ell,\beta'')\mu_2$. As ℓ is not a variable by (i) the variable assignments β' and β'' must be identical on $\mathcal{V}(\ell) \supseteq \mathcal{V}(r)$. This finally proves that the rules $\mathit{Lab}(\ell,\beta') \to \mathit{Lab}(r,\beta')$ and $\mathit{Lab}(\ell,\beta'') \to \mathit{Lab}(r,\beta'')$ are identical and hence, $s = \mathit{Lab}(t',\beta)$. □

Proof of Theorem 7.10. It suffices to extend the proof of Theorem 7.5 by showing for every term t that $\mathcal{Q}$-termination of t w.r.t. $\mathcal{R}$ implies $\overline{\mathcal{Q}}$-termination of $\mathit{Lab}(t,\beta)$ w.r.t. $\overline{\mathcal{R}}$. Therefore, suppose that $\mathit{Lab}(t,\beta) \xrightarrow{\overline{\mathcal{Q}}}_{\overline{\mathcal{R}}} s_1 \xrightarrow{\overline{\mathcal{Q}}}_{\overline{\mathcal{R}}} s_2 \xrightarrow{\overline{\mathcal{Q}}}_{\overline{\mathcal{R}}} \ldots$ is an infinite reduction. By Lemma 7.9 (ii) every s_i is of the form $\mathit{Lab}(t_i,\beta)$ and $t \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \ldots$. Then using Lemma 7.3 finally proves that t is not $\mathcal{Q}$-terminating w.r.t. $\mathcal{R}$ as $t \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t_1 \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t_2 \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \ldots$. □

Proof of Lemma 7.13. We consider both directions separately. If $\mathit{Lab}(t,\beta) \notin \mathit{NF}(\underline{\mathcal{Q}})$ then $\mathit{Lab}(t,\beta)|_p = q\mu$ for some $q \in \underline{\mathcal{Q}}$. By the definition of $\underline{\mathcal{Q}}$ the term $\mathit{Unlab}(q)$ is contained in $\mathcal{Q}$. As $t|_p = \mathit{Unlab}(\mathit{Lab}(t,\beta)|_p) = \mathit{Unlab}(q\mu) = \mathit{Unlab}(q)\mathit{Unlab}(\mu)$ we have proven $t \notin \mathit{NF}(\mathcal{Q})$.

For the other direction assume that $\mathit{Unlab}(t) \notin \mathit{NF}(\mathcal{Q})$, i.e., $\mathit{Unlab}(t|_p) = \mathit{Unlab}(t)|_p = q\mu$ for some $q \in \mathcal{Q}$. If we can show that $t|_p = q'\mu'$ for some $q' \in \underline{\mathcal{Q}}$ then we are done as this implies $t \notin \mathit{NF}(\underline{\mathcal{Q}})$.

---

[47] Note that in Lemma 7.2 the position of the reduction is the same for the reduction of the labeled terms and for the reduction of the unlabeled terms.

To this end we prove by induction that for every linear term $s \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and every term $r \in \mathcal{T}(\overline{\mathcal{F}}, \mathcal{V})$ that whenever $\mathit{Unlab}(r) = s\mu$ then there is a substitution μ' such that $r = s'\mu'$ for some s' with $\mathit{Unlab}(s') = s$. Then we use this result for $s = q$, $r = t|_p$, and $s' = q'$.

If s is a variable then we just choose $s' = s$ and $\mu' = \{s/r\}$. Otherwise, $s = f(s_1, \ldots, s_n)$ and hence, $\mathit{Unlab}(r) = f(\mathit{Unlab}(r_1), \ldots, \mathit{Unlab}(r_n))$ with $\mathit{Unlab}(r_i) = s_i\mu$ for every i. By induction we obtain terms s'_i and μ'_i with $\mathit{Unlab}(s'_i) = s_i$ and $r_i = s'_i\mu'_i$. As s is linear we can combine all substitutions μ'_i to one shared substitution μ' such that $r_i = s'_i\mu'$ for every i. It remains to construct s'. As r has the form $f_l(r_1, \ldots, r_n)$ we just have to choose $s' = f_l(s'_1, \ldots, s'_n)$. In this way obviously $s'\mu' = r$ and $\mathit{Unlab}(s') = f(\mathit{Unlab}(s'_1), \ldots, \mathit{Unlab}(s'_n)) = s$ are both valid. □

Proof of Theorem 7.15. For soundness one can prove in the same way as in the proof of Theorem 7.5 that if $s_1 \to t_1, s_2 \to t_2, \ldots$ is an infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain by using the substitution σ then $\mathit{Lab}(s_1 \to t_1, \overline{\beta})$ is an infinite $(\overline{\mathcal{P}}, \underline{\mathcal{Q}}, \overline{\mathcal{R}})$-chain by using the substitution $\overline{\sigma}$. To this end one just has to exchange Lemma 7.4 with Lemma 7.14 and Lemma 7.3 with Lemma 7.13, respectively.

To carry over minimality if $\mathcal{Q}$ is linear, suppose that some $\mathit{Lab}(t_i, \overline{\beta})\overline{\sigma} = \mathit{Lab}(t_i\sigma, \beta)$ is not $\underline{\mathcal{Q}}$-terminating w.r.t. $\overline{\mathcal{R}}$. Then by Lemma 7.14 the term $\mathit{Unlab}(\mathit{Lab}(t_i\sigma, \beta)) = t_i\sigma$ is not $\mathcal{Q}$-terminating w.r.t. $\mathcal{R}$.

In the same way one can prove completeness for a linear $\mathcal{Q}$. If $\overline{\mathcal{R}}$ is not $\underline{\mathcal{Q}}$-terminating, then there is some t which starts an infinite reduction. By Lemma 7.14 the term $\mathit{Unlab}(t)$ shows that then $\mathcal{R}$ is not $\mathcal{Q}$-terminating. Moreover, suppose $\mathit{Lab}(s_1 \to t_1, \beta_1), \mathit{Lab}(s_2 \to t_2, \beta_2), \ldots$ is an infinite $(\overline{\mathcal{P}}, \underline{\mathcal{Q}}, \overline{\mathcal{R}})$-chain, i.e., there is some σ such that $\mathit{Lab}(t_i, \beta_i)\sigma \xrightarrow{\underline{\mathcal{Q}}}{}^{*}_{\overline{\mathcal{R}}} \mathit{Lab}(s_{i+1}, \beta_{i+1})\sigma$ and $\mathit{Lab}(s_i, \beta_i)\sigma \in \mathit{NF}(\underline{\mathcal{Q}})$ for all i. Then using Lemma 7.14 we obtain the reductions $t_i\,\mathit{Unlab}(\sigma) = \mathit{Unlab}(\mathit{Lab}(t_i, \beta_i)\sigma) \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} \mathit{Unlab}(\mathit{Lab}(s_{i+1}, \beta_{i+1})\sigma) = s_{i+1}\,\mathit{Unlab}(\sigma)$ and by Lemma 7.13 all terms $s_i\,\mathit{Unlab}(\sigma) = \mathit{Unlab}(\mathit{Lab}(s_i, \beta_i)\sigma)$ are in $\mathcal{Q}$-normal form. Thus, $s_1 \to t_1, s_2 \to t_2, \ldots$ is an infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain. □

Proof of Lemma 7.18. We use $\hookrightarrow^*$ as a shortcut for $\xrightarrow{\underline{\mathcal{Q}}}{}^{*}_{\overline{\mathcal{R}} \cup \mathcal{D}ecr}$.

For the first claim we extend the proof of [Zan95, Lemma 7] and integrate Lemma 7.13. So, let $t \xrightarrow{\mathcal{Q}}{}^{n}_{\mathcal{R}} t'$ with $t' \in \mathit{NF}(\mathcal{Q})$. We perform an outer induction on the reduction length n and an inner induction on t. If $n = 0$ then there is nothing to show. Otherwise, there is at least one reduction step.

If all reductions are below the root, then $t = f(t_1, \ldots, t_n) \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} f(t'_1, \ldots, t'_n) = t'$ where $t_i \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} t'_i$. By the inner induction hypothesis we obtain $\mathit{Lab}(t_i, \beta) \hookrightarrow^* \mathit{Lab}(t'_i,)$. Let $l = \lambda_f([\beta](t_1), \ldots, [\beta](t_n))$. Then $\mathit{Lab}(t, \beta) = f_l(\mathit{Lab}(t_1, \beta), \ldots, \mathit{Lab}(t_n, \beta)) \hookrightarrow^* f_l(\mathit{Lab}(t'_1, \beta), \ldots, \mathit{Lab}(t'_n, \beta))$. As $\mathcal{M}$ is a quasi-model of $\mathcal{R}$ we can derive $[\beta](t_i) \geq [\beta](t'_i)$. By weak monotonicity we also obtain $l \geq \lambda_f([\beta](t'_1), \ldots, [\beta](t'_n)) =: l'$. If $l = l'$ then we are done. Otherwise, if $l > l'$ then there is a corresponding decreasing rule $f_l(x_1, \ldots, x_n) \to f_{l'}(x_1, \ldots, x_n)$. Thus, we obtain $\mathit{Lab}(t, \beta) \hookrightarrow^* \to_{\mathcal{D}ecr} \mathit{Lab}(t', \beta)$. As t' is in normal form w.r.t. $\mathcal{Q}$ by Lemma 7.13 every term $\mathit{Lab}(t'_i,)$ is in normal form w.r.t. $\underline{\mathcal{Q}}$. Hence, this last reduction step with the decreasing rule respects the evaluation strategy given by $\underline{\mathcal{Q}}$ and we are done with this case.

It remains to consider the case that there is at least one root reduction, i.e., we have $t \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} \ell\sigma \xrightarrow{\mathcal{Q}}_{\mathcal{R}} r\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} t'$ where the step from $\ell\sigma$ to $r\sigma$ with rule $\ell \to r \in \mathcal{R}$ is the first reduction at the root. Note that all direct subterms of $\ell\sigma$ are in $\mathcal{Q}$-normal form. Thus, by the inner induction hypothesis we can reason as in the previous case to obtain $\mathit{Lab}(t, \beta) \hookrightarrow^* \mathit{Lab}(\ell\sigma, \beta)$. Moreover, by the outer induction hypothesis we know that

$\mathit{Lab}(r\sigma, \beta) \hookrightarrow^* \mathit{Lab}(t', \beta,)$. Hence, it suffices to prove $\mathit{Lab}(\ell\sigma, \beta) \xrightarrow{\underline{\mathcal{Q}}}_{\overline{\mathcal{R}}} \mathit{Lab}(r\sigma, \beta)$. But this step directly follows from Lemma 7.13 and [Zan95, Lemma 2] as shown in the proof of Theorem 7.5. $\square$

Proof of Theorem 7.19. The proof is similar to the one in Theorem 7.15 where one replaces Lemma 7.14 by Lemma 7.18. However, to carry over minimality and to prove completeness one now has to construct infinite reductions w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ from infinite reductions w.r.t. $\xrightarrow{\underline{\mathcal{Q}}}_{\overline{\mathcal{R}} \cup \mathcal{D}ecr}$. Using Lemma 7.18 we only get an infinite $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$-reduction if there are infinitely many $\xrightarrow{\underline{\mathcal{Q}}}_{\overline{\mathcal{R}}}$ steps. But this must always be the case as $\mathcal{D}ecr$ is terminating due to the well-foundedness of $>$. $\square$

Proof of Theorem 7.23. Completeness is proven as in Theorem 7.19. For soundness consider a $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain $s_1 \to t_1, s_2 \to t_2, \dots$, i.e., there is a substitution σ such that $t_i\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} s_{i+1}\sigma$ and $s_i\sigma \in \mathit{NF}(\mathcal{Q})$ for every i. Moreover, if the chain is minimal, then every $t_i\sigma$ is terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$.

We define $\hookrightarrow^* := \xrightarrow{\underline{\mathcal{Q}}}{}^*_{\overline{\mathcal{R}} \cup \mathcal{D}ecr_{\neg\mathcal{H}}}$. We first prove $\mathit{Lab}(t_i\sigma, \beta) \hookrightarrow^* \mathit{Lab}(s_{i+1}\sigma, \beta)$. Unfortunately, by Lemma 7.18 it is only possible to conclude $\mathit{Lab}(t_i\sigma, \beta) \xrightarrow{\underline{\mathcal{Q}}}{}^*_{\overline{\mathcal{R}} \cup \mathcal{D}ecr} \mathit{Lab}(s_{i+1}\sigma, \beta)$, i.e., up to now the rules $\mathcal{D}ecr_{\mathcal{H}}$ cannot be dropped. Therefore we use an alternative labeling Lab' where the corresponding set $\mathcal{D}ecr'_{\mathcal{H}}$ is empty. We define the labels and the label maps for Lab' to be like those for Lab but for each $f \in \mathcal{H}$ we define $L'_f = \{\textvisiblespace\}$ and $\lambda'_f(\dots) = \textvisiblespace$. Then, for all terms $t \in \mathcal{T}(\mathcal{F} \setminus \mathcal{H}, \mathcal{V})$ we have $\mathit{Lab}(t, \beta) = \mathit{Lab}'(t, \beta)$. Hence, by the definition of head symbols we obtain the same sets $\overline{\mathcal{R}}$ and $\underline{\mathcal{Q}}$, regardless whether we use Lab or Lab' to produce the labeled systems. Moreover, the decreasing rules $\mathcal{D}ecr'$ for the labeling Lab' are exactly $\mathcal{D}ecr_{\neg\mathcal{H}}$. Thus, by Lemma 7.18 we conclude for all terms t with $t \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} s$ that there is the reduction $\mathit{Lab}'(t, \beta) \hookrightarrow^* \mathit{Lab}'(s, \beta)$.

Now we show that from some point onwards every reduction $s_i\sigma \to t_i\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} s_{i+1}\sigma$ can be simulated in the labeled system when using the substitution $\overline{\sigma}$ as in the proof of Theorem 7.5. By Theorem 4.37 we obtain some n such that for all $i \geq n$ the equivalence $\mathit{root}(s_{i+1}) \in \mathcal{H}$ iff $\mathit{root}(t_i) \in \mathcal{H}$ is satisfied. W.l.o.g. we may assume $n = 1$ and we fix some variable assignment β. Using [Zan95, Lemma 2] we obtain the substitution $\overline{\sigma}$ which is defined as $\overline{\sigma}(x) = \mathit{Lab}'(\sigma(x), \beta)$ and the variable assignment $\overline{\beta}$ with $\overline{\beta}(x) = [\beta](\sigma(x))$ such that for all terms $t \in \mathcal{T}(\mathcal{F} \setminus \mathcal{H}, \mathcal{V})$ the equation $\mathit{Lab}'(t\sigma, \beta) = \mathit{Lab}'(t, \overline{\beta})\overline{\sigma} = \mathit{Lab}(t, \overline{\beta})\overline{\sigma}$ is satisfied. We consider two cases.

If $\mathit{root}(t_i) \notin \mathcal{H}$ then $\mathit{root}(s_{i+1}) \notin \mathcal{H}$ and by the definition of head symbols we know that $\{t_i, s_{i+1}\} \subseteq \mathcal{T}(\mathcal{F} \setminus \mathcal{H}, \mathcal{V})$. As $s_{i+1}\sigma \in \mathit{NF}(\mathcal{Q})$ we can apply Lemma 7.18 and obtain

$$\begin{array}{rcl} \mathit{Lab}(s_i, \overline{\beta})\overline{\sigma} & \to & \mathit{Lab}(t_i, \overline{\beta})\overline{\sigma} \\ & = & \mathit{Lab}'(t_i\sigma, \beta) \\ & \hookrightarrow^* & \mathit{Lab}'(s_{i+1}\sigma, \beta) \\ & = & \mathit{Lab}(s_{i+1}, \overline{\beta})\overline{\sigma} \end{array}$$

where by construction $\mathit{Lab}(s_i, \overline{\beta}) \to \mathit{Lab}(t_i, \overline{\beta}) \in \underline{\mathcal{P}}$.

In the other case $\mathit{root}(t_i) \in \mathcal{H}$, i.e., t_i is of the form $f(\ell_1, \dots, \ell_n)$ and $s_{i+1} = f(r_1, \dots, r_n)$. As $t_i\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} s_{i+1}\sigma$ we know that $\ell_i\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} r_i\sigma$ for all i. Moreover, by the definition of head symbols all terms $\ell_1, \dots, \ell_n, r_1, \dots, r_n$ are from $\mathcal{T}(\mathcal{F} \setminus \mathcal{H}, \mathcal{V})$. Hence, as in the previous case we obtain $\mathit{Lab}(\ell_i, \overline{\beta})\overline{\sigma} \hookrightarrow^* \mathit{Lab}(r_i, \overline{\beta})\overline{\sigma}$. Additionally, as $\mathcal{M}$ is a quasi-model we obtain $[\beta](\ell_i\sigma) \geq [\beta](r_i\sigma)$. Using weak monotonicity of $\geq$ and the result of [Zan95, Lemma 1] that $[\beta](\ell_i\sigma) = [\overline{\beta}](\ell_i)$ and $[\beta](r_i\sigma) = [\overline{\beta}](r_i)$ we obtain $\lambda_f([\overline{\beta}](\ell_1), \dots, [\overline{\beta}](\ell_n)) \geq l'$ for

$l' = \lambda_f([\overline{\beta}](r_1), \dots, [\overline{\beta}](r_n))$. Thus, $\mathit{Lab}(s_i, \overline{\beta}) \to f_{l'}(\mathit{Lab}(\ell_1, \overline{\beta}), \dots, \mathit{Lab}(\ell_n, \overline{\beta})) \in \underline{\mathcal{P}}$. We conclude

$$\begin{array}{rcl} \mathit{Lab}(s_i, \overline{\beta})\overline{\sigma} & \to_{\underline{\mathcal{P}}} & f_{l'}(\mathit{Lab}(\ell_1, \overline{\beta}), \dots, \mathit{Lab}(\ell_n, \overline{\beta}))\overline{\sigma} \\ & = & f_{l'}(\mathit{Lab}(\ell_1, \overline{\beta})\overline{\sigma}, \dots, \mathit{Lab}(\ell_n, \overline{\beta})\overline{\sigma}) \\ & \hookrightarrow^* & f_{l'}(\mathit{Lab}(r_1, \overline{\beta})\overline{\sigma}, \dots, \mathit{Lab}(r_n, \overline{\beta})\overline{\sigma}) \\ & = & \mathit{Lab}(f(r_1, \dots, r_n), \overline{\beta})\overline{\sigma} \\ & = & \mathit{Lab}(s_{i+1}, \overline{\beta})\overline{\sigma} \end{array}$$

So, in both cases we get the reduction $\mathit{Lab}(s_i, \overline{\beta})\overline{\sigma} \to_{\underline{\mathcal{P}}} \hookrightarrow^* \mathit{Lab}(s_{i+1}, \overline{\beta})\overline{\sigma}$. Since every $s_i\sigma \in \mathit{NF}(\mathcal{Q})$, by Lemma 7.13 every $\mathit{Lab}(s_i\sigma, \beta) = \mathit{Lab}(s_i, \overline{\beta})\overline{\sigma} \in \mathit{NF}(\underline{\mathcal{Q}})$. This finally shows that we have constructed an infinite $(\underline{\mathcal{P}}, \underline{\mathcal{Q}}, \overline{\mathcal{R}} \cup \mathcal{D}\mathit{ecr}_{\neg\mathcal{H}})$-chain. A direct consequence of Lemma 7.18 is the fact that this chain is minimal if $\mathcal{Q}$ is linear and if all terms $t_i\sigma$ are terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$. □

Proof of Theorem 7.30. We only consider the case where we use a processor $\mathit{Proc}_{\mathit{quasi}}$ of Theorem 7.19 and where $\mathcal{Q}$ is linear. Then $(\mathcal{P}', \mathcal{Q}', \mathcal{R}', f') = (\overline{\mathcal{P}}, \underline{\mathcal{Q}}, \overline{\mathcal{R}} \cup \mathcal{D}\mathit{ecr}, f)$. The remaining cases are completely identical. Let $s_1 \to t_1, s_2 \to t_2, \dots$ be an infinite chain where the pairs are instantiated by the substitution σ and the where the intermediate steps are done with the rules $\ell_{i,j} \to r_{i,j}$. By Theorem 7.19 we know that $\mathit{Lab}(s_1, \overline{\beta}) \to \mathit{Lab}(t_1, \overline{\beta}), \mathit{Lab}(s_2, \overline{\beta}) \to \mathit{Lab}(t_2, \overline{\beta}), \dots$ is an infinite $(\overline{\mathcal{P}}, \underline{\mathcal{Q}}, \overline{\mathcal{R}} \cup \mathcal{D}\mathit{ecr}, f)$-chain. In more detail, for every reduction step with $\ell_{i,j} \to r_{i,j}$ in the original chain we used one reduction step with $\mathit{Lab}(\ell_{i,j}, \overline{\beta}) \to \mathit{Lab}(r_{i,j}, \overline{\beta})$ together with some additional $\mathcal{D}\mathit{ecr}$-steps in the labeled chain.

By Definition 7.29 there are numbers n and m such that $\mathit{Lab}(s_n, \overline{\beta}) \to \mathit{Lab}(t_n, \overline{\beta})$, $\mathit{Lab}(s_{n+1}, \overline{\beta}) \to \mathit{Lab}(t_{n+1}, \overline{\beta}), \dots$ is an infinite path in $\mathcal{P}_m$ where all rules $\mathit{Lab}(\ell_{i,j}, \overline{\beta}) \to \mathit{Lab}(r_{i,j}, \overline{\beta})$ with $i \geq n$ are from $\mathcal{R}_m \setminus \mathcal{D}\mathit{ecr}$.[48] Thus, even for the original chain we know that from point n onwards, only rules $\ell_{i,j} \to r_{i,j}$ have been used where the labeled version of the rule is in $\mathcal{R}_m \setminus \mathcal{D}\mathit{ecr}$, i.e., $\ell_{i,j} \to r_{i,j} \in \mathit{Unlab}(\mathcal{R}_m \setminus \mathcal{D}\mathit{ecr})$. And as the infinite path $\mathit{Lab}(s_n, \overline{\beta}) \to \mathit{Lab}(t_n, \overline{\beta}), \mathit{Lab}(s_{n+1}, \overline{\beta}) \to \mathit{Lab}(t_{n+1}, \overline{\beta}), \dots$ only uses edges from $\mathcal{P}_m$ we also know that in the original chain from point n onwards, only edges $s_i \to t_i, s_{i+1} \to t_{i+1}$ with $\mathit{Lab}(s_i, \overline{\beta}) \to \mathit{Lab}(t_i, \overline{\beta}), \mathit{Lab}(s_{i+1}, \overline{\beta}) \to \mathit{Lab}(t_{i+1}, \overline{\beta}) \in \mathcal{P}_m$ are used. Hence, all these edges of the original chain are present in $\mathit{Unlab}(\mathcal{P}_m)$.

Thus, we have proven that $s_n \to t_n, s_{n+1} \to t_{n+1}, \dots$ is an infinite path in $\mathit{Unlab}(\mathcal{P}_m)$ and that all rules $\ell_{i,j} \to r_{i,j}$ with $i \geq n$ are from $\mathit{Unlab}(\mathcal{R}_m \setminus \mathcal{D}\mathit{ecr})$, i.e., we have proven the characterizing property of a chain identifying processor. □

Proof of Lemma 7.31. The proof is trivial using the definition of a chain identifying processor. □

A.8. Proofs of Chapter 8

Proof of Theorem 8.5. If $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ is looping then there are the following reductions where we use the terms t_i^n to identify the intermediate results.

$$s\mu^n = t_1^n \xrightarrow{\mathcal{Q}}_{\ell_1 \to r_1, p_1} t_2^n \xrightarrow{\mathcal{Q}}_{\ell_2 \to r_2, p_2} t_3^n \dots \xrightarrow{\mathcal{Q}}_{\ell_m \to r_m, p_m} = t_{m+1}^n = s\mu^{n+1}$$

---

[48] Note that the TRSs $\overline{\mathcal{R}}$ and $\mathcal{D}\mathit{ecr}$ are disjoint even if some rule of the form $f(x_1, \dots, x_n) \to f(x_1, \dots, x_n)$ is contained in $\mathcal{R}$.

Let $I = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, m\}$ be the set of indices such that for all $i_j \in I$ the rule $\ell_{i_j} \to r_{i_j}$ is taken from $\mathcal{P}$. Then

$$\begin{array}{l} s\mu^0 \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} t^0_{i_1} \to_{\mathcal{P},\varepsilon} t^0_{i_1+1} \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} t^0_{i_2} \to_{\mathcal{P},\varepsilon} t^0_{i_2+1} \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} \ldots \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} t^0_{i_k} \to_{\mathcal{P},\varepsilon} t^0_{i_k+1} \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} \\ s\mu^1 \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} t^1_{i_1} \to_{\mathcal{P},\varepsilon} t^1_{i_1+1} \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} t^1_{i_2} \to_{\mathcal{P},\varepsilon} t^1_{i_2+1} \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} \ldots \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} t^1_{i_k} \to_{\mathcal{P},\varepsilon} t^1_{i_k+1} \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{R}} \\ \ldots \end{array}$$

is an infinite reduction where all $t^n_{i_j}$ are in $\mathcal{Q}$-normal form. Since all $\mathcal{P}$ steps are at the root, this clearly corresponds to an infinite chain. Thus, completeness is proven and for soundness there is nothing to show. □

Proof of Theorem 8.6. Let $\mathcal{P} = DP(\mathcal{R})$.

We first prove the difficult direction of the theorem that $\mathcal{Q}$-loopingness implies loopingness of $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$. So, let there be m, s, C, μ, $\ell_i \to r_i$, p, and $p_1, \ldots, p_m$ such that $p = i_1 \ldots i_k$ is the position of the hole in C and that for all n we have the following looping reduction.

$$s\mu^n \xrightarrow{\mathcal{Q}}_{\ell_1 \to r_1, p_1} \circ \xrightarrow{\mathcal{Q}}_{\ell_2 \to r_2, p_2} \circ \cdots \circ \xrightarrow{\mathcal{Q}}_{\ell_m \to r_m, p_m} C\mu^n[s\mu^{n+1}]_p$$

Then clearly, if we define $\rhd_i$ to be the direct subterm relation of the i-th argument then we additionally obtain

$$C\mu^n[s\mu^{n+1}]_p \rhd_{i_1} \circ \cdots \circ \rhd_{i_k} s\mu^{n+1}$$

Hence, if $\mathcal{R}$ is $\mathcal{Q}$-looping then for every n there is a reduction from $s\mu^n$ to $s\mu^{n+1}$ using every time the same sequence of $\xrightarrow{\mathcal{Q}}_{\mathcal{R}} \cup \{\rhd_i \mid i \in \mathbb{N}\}$-reductions. W.l.o.g. we consider such a reduction with a minimal number m of $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$-steps and under all these minimal reductions we assume that s is a smallest term.

We first show that we can reorder the reduction such that after a $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$-step at a non-root position there never is a $\rhd_i$-step. To this end, notice that whenever we have the reductions $t\mu^n \xrightarrow{\mathcal{Q}}_{\ell \to r, p} t_n$ for all n then $t_n = t_0\mu^n$, and similarly whenever $t\mu^n \rhd_i s_n$ then $s_n = s_0\mu^n$. Hence, we know that in the reduction sequence we only have to consider terms of the form $t\mu^n$.

To reorder the whole reduction we use the following auxiliary property for every local reduction $u\mu^n \xrightarrow{\mathcal{Q}}_{\ell \to r, p} v\mu^n \rhd_i w\mu^n$ where p is a non-root position.

$$p = ip' \text{ and } u\mu^n \rhd_i \circ \xrightarrow{\mathcal{Q}}_{\ell \to r, p'} w\mu^n. \qquad (\star)$$

To prove $(\star)$ we first investigate the structure of u and v. Let $p = jp'$. Since $u \xrightarrow{\mathcal{Q}}_{\ell \to r, p} v$ we know that $u = f(u_1, \ldots, u_j, \ldots, u_k)$ and $v = f(u_1, \ldots, v_j, \ldots, u_k)$ where $u_j\mu^n \xrightarrow{\mathcal{Q}}_{\ell \to r, p'} v_j\mu^n$ for all n. We distinguish two cases. If $i \neq j$ then clearly $w = u_i$ and hence, $u\mu^n \rhd_i w\mu^n$. This is a contradiction to the minimality of m since now we have loop with $m - 1$ $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$-steps. In the other case $i = j$. Then $w = v_j$ and we conclude $u\mu^n \rhd_i u_i\mu^n = u_j\mu^n \xrightarrow{\mathcal{Q}}_{\ell \to r, p'} v_j\mu^n = w\mu^n$ which finishes the proof of $(\star)$.

Using $(\star)$ repeatedly, we can reorder the sequence until there is no $\rhd$-reduction directly after a $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$-reduction at a non-root position. Hence, we have proven that there is some s' such that $s\mu^n \trianglerighteq s'\mu^n \; ((\xrightarrow{\mathcal{Q}}_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \cup \xrightarrow{\mathcal{Q}}_{\mathcal{R},>\varepsilon})^m \; s\mu^{n+1}$ where for every n the same rules are applied at the same positions.

One can argue that $s' = s$, because otherwise $s'\mu^n \; ((\xrightarrow{\mathcal{Q}}_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \cup \xrightarrow{\mathcal{Q}}_{\mathcal{R},>\varepsilon})^m \; s\mu^{n+1} \rhd s'\mu^{n+1}$ is a loop with minimal number of $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ steps which contradicts the minimality of s. Thus, from now on we only have to consider looping reductions using the relation $(\xrightarrow{\mathcal{Q}}_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \cup \xrightarrow{\mathcal{Q}}_{\mathcal{R},>\varepsilon}$. Later in this proof we will see that the first kind of reduction corresponds to a dependency pair and the latter are the reductions that are used to go from one dependency pair to the next.

In order to prove that $(\mathcal{P},\mathcal{Q},\mathcal{R})$ is looping we need at least one $\mathcal{P}$-step. Therefore, we now show that there must be at least one reduction step at the root. We perform a proof by contradiction. Suppose, there is no reduction at the root. Then we obtain $s\mu^n \xrightarrow{\mathcal{Q}}{}^{m}_{\mathcal{R},>\varepsilon} s\mu^{n+1}$. Hence, $s = f(s_1,\dots,s_k)$ and $s\mu^n = f(s_1\mu^n,\dots,s_k\mu^n) \xrightarrow{\mathcal{Q}}{}^{m}_{\mathcal{R},>\varepsilon} f(s_1,\dots,s_k)\mu^{n+1} = f(s_1\mu^{n+1},\dots,s_k\mu^{n+1})$. Since, $m > 0$ there must be at least one i such that $s_i\mu^n \xrightarrow{\mathcal{Q}}{}^{m'}_{\mathcal{R}} s_i\mu^{n+1}$ with $0 < m' \le m$. This is a contradiction to the minimality of m or, if $m = m'$, to the minimality of s.

As we want to map $\xrightarrow{\mathcal{Q}}_{\mathcal{R},\varepsilon} \circ \trianglerighteq$-steps to dependency pairs we must know that in the looping reduction we only take subterms with $\trianglerighteq$ which have a defined root symbol. This is proven as follows. First note that the root of s must be defined, since s starts a reduction w.r.t. $((\xrightarrow{\mathcal{Q}}_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \cup \xrightarrow{\mathcal{Q}}_{\mathcal{R},>\varepsilon})$ that contains a root step. Moreover, for any term s' with $s'\ ((\xrightarrow{\mathcal{Q}}_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \cup \xrightarrow{\mathcal{Q}}_{\mathcal{R},>\varepsilon})^*\ s\mu$, the root of s' must also be defined. The reason is that otherwise, s' can only be reduced by $\xrightarrow{\mathcal{Q}}_{\mathcal{R},>\varepsilon}$-steps, but then one cannot reduce to the term $s\mu$ which has a defined root.

So, for $s_0 = s$ and $s_m = s\mu$ we know that

$$\begin{aligned} s\mu^n = s_0\mu^n \quad & ((\xrightarrow{\mathcal{Q}}_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \cup \xrightarrow{\mathcal{Q}}_{\mathcal{R},>\varepsilon}) \quad s_1\mu^n \\ & ((\xrightarrow{\mathcal{Q}}_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \cup \xrightarrow{\mathcal{Q}}_{\mathcal{R},>\varepsilon}) \quad \dots \\ & ((\xrightarrow{\mathcal{Q}}_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \cup \xrightarrow{\mathcal{Q}}_{\mathcal{R},>\varepsilon}) \quad s_m\mu^n = s\mu^{n+1} \end{aligned}$$

where all s_i have a defined root symbol and where at least one root-reduction is used. We show that the DP problem $(\mathcal{P},\mathcal{Q},\mathcal{R})$ is looping by proving that the following reduction exists.

$$s_0^\sharp\mu^n\ (\xrightarrow{\mathcal{Q}}_{\mathcal{P},\varepsilon} \cup \xrightarrow{\mathcal{Q}}_{\mathcal{R}})\ s_1^\sharp\mu^n\ (\xrightarrow{\mathcal{Q}}_{\mathcal{P},\varepsilon} \cup \xrightarrow{\mathcal{Q}}_{\mathcal{R}}) \dots (\xrightarrow{\mathcal{Q}}_{\mathcal{P},\varepsilon} \cup \xrightarrow{\mathcal{Q}}_{\mathcal{R}})\ s_m^\sharp\mu^n$$

Therefore, it suffices to show that whenever $s_i\mu^n\ ((\xrightarrow{\mathcal{Q}}_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \cup \xrightarrow{\mathcal{Q}}_{\mathcal{R},>\varepsilon})\ s_{i+1}\mu^n$ then $s_i^\sharp\mu^n\ (\xrightarrow{\mathcal{Q}}_{\mathcal{P},\varepsilon} \cup \xrightarrow{\mathcal{Q}}_{\mathcal{R}})\ s_{i+1}^\sharp\mu^n$. If $s_i\mu^n \xrightarrow{\mathcal{Q}}_{\mathcal{R},>\varepsilon} s_{i+1}\mu^n$ then $s_i^\sharp\mu^n \xrightarrow{\mathcal{Q}}_{\mathcal{R}} s_{i+1}^\sharp\mu^n$ is obviously satisfied, since $^\sharp$ only changes the root symbol. Next we show that $s_i\mu^n\ (\xrightarrow{\mathcal{Q}}_{\mathcal{R},\varepsilon} \circ \trianglerighteq)\ s_{i+1}\mu^n$ implies $s_i^\sharp\mu^n \xrightarrow{\mathcal{Q}}_{\mathcal{P},\varepsilon} s_{i+1}^\sharp\mu^n$. So, let $s_i = \ell\sigma \to r\sigma \trianglerighteq_p s_{i+1}$ for some rule $\ell \to r \in \mathcal{R}$ and some position p. First note that due to the minimality of m, the term s_{i+1} cannot be a proper subterm of s_i. Hence, p is a non-variable position of r and $r|_p$ is not a proper subterm of ℓ. Thus, $s_{i+1} = r|_p\sigma$. As the root of s_{i+1} is defined this shows that $\ell^\sharp \to r|_p^\sharp$ is a dependency pair of $\mathcal{R}$. Thus, $s_i^\sharp\mu^n = \ell^\sharp\sigma\mu^n \to_{\mathcal{P},\varepsilon} r|_p^\sharp\sigma\mu^n = s_{i+1}^\sharp\mu^n$. Since all proper subterms of $s_i\mu^n$ are in $\mathcal{Q}$-normal form and since the tuple-symbols are fresh symbols not occurring in $\mathcal{Q}$ we also know that $s_i^\sharp\mu^n$ is in $\mathcal{Q}$-normal form. Thus we have successfully proven that $(\mathcal{P},\mathcal{Q},\mathcal{R})$ is looping.

For the other direction of the theorem let $(\mathcal{P},\mathcal{Q},\mathcal{R})$ be looping. Then there is a term s, a substitution μ, rules $\ell_1 \to r_1, \dots, \ell_m \to r_m \in \mathcal{P} \cup \mathcal{R}$, and positions $p_1,\dots,p_m$ such that

$$s\mu^n = s_0\mu^n \xrightarrow{\mathcal{Q}}_{\ell_1 \to r_1, p_1} s_1\mu^n \xrightarrow{\mathcal{Q}}_{\ell_2 \to r_2, p_2} \dots \xrightarrow{\mathcal{Q}}_{\ell_m \to r_m, p_m} s_m\mu^n = s\mu^{n+1}$$

Since there is at least one $\mathcal{P}$-reduction at the root and since both sides of a dependency pair have a tuple-symbol as root, we know that at least one term s_i in the reduction has a tuple-symbol as root. But since the tuple-symbols are fresh and do not occur in $\mathcal{R}$, all terms s_i in the reductions have a tuple-symbol as root, and the $\mathcal{R}$-steps are only possible below the root. Hence, when replacing all these tuple-symbols by their corresponding defined symbols, i.e., we consider the terms $s_i^\flat$ where $(t^\sharp)^\flat = t$, then we can still perform all $\mathcal{R}$-steps on the terms $s_i^\flat$. To be more precise, $s_i\mu^n \xrightarrow{\mathcal{Q}}_{\mathcal{R}} s_{i+1}\mu^n$ implies $s_i^\flat\mu^n \xrightarrow{\mathcal{Q}}_{\mathcal{R}} s_{i+1}^\flat\mu^n$. Moreover, whenever we reduce with a dependency pair $\ell^\sharp \to r|_p^\sharp$ then there is a corresponding rule $\ell \to r \in \mathcal{R}$. Thus, whenever $s_i\mu^n \xrightarrow{\mathcal{Q}}_{\mathcal{P},\varepsilon} s_{i+1}\mu^n$ then $s_i^\flat\mu^n \xrightarrow{\mathcal{Q}}_{\mathcal{R}} r[s_{i+1}^\flat\mu^n]_p$. Collecting all these contexts $r[\cdot]_p$ for all the $\mathcal{P}$-steps in the loop results in one combined context C and we obtain $s^\flat\mu^n \overset{\mathcal{Q}}{\to}{}^{m}_{\mathcal{R}} C\mu^n[s^\flat\mu^{n+1}]$ where in every iteration the same rules are applied at the same positions. Thus, $\mathcal{R}$ is $\mathcal{Q}$-looping. □

Proof of Theorem 8.9. Soundness is due to Lemma 2.4 and $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\varnothing)$.

For the completeness we consider two cases. First, if $\mathcal{R}$ is not $\mathcal{Q}$-terminating, then there is nothing to show. In the other case $\mathcal{R}$ is $\mathcal{Q}$-terminating. Thus, using the last requirement of the processor we also know that $\mathcal{R}$ must be terminating. Now, if $(\mathcal{P}, \varnothing, \mathcal{R}, f)$ is infinite then due to termination of $\mathcal{R}$ there must be an infinite minimal $(\mathcal{P}, \varnothing, \mathcal{R})$-chain. Using the soundness of the processor to switch to innermost termination (Theorem 3.14) we obtain an infinite $(\mathcal{P}, \mathit{lhs}(\mathcal{R}), \mathcal{R})$-chain. Thus, again using Lemma 2.4 and $\mathit{NF}(\mathit{lhs}(\mathcal{R})) = \mathit{NF}(\mathcal{R}) \subseteq \mathit{NF}(\mathcal{Q})$ one can finally conclude that there must be an infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain. □

Proof of Theorem 8.13. If q is a variable then the result is obviously true, so let $q \notin \mathcal{V}$. We consider both directions separately.

First, let $(u \gtrdot q, \mu)$ be solvable, i.e., there are σ and n such that $u\mu^n = q\sigma$. If u is a subterm of s, i.e., $u = s|_p$ then $s\mu^n|_p = s|_p\mu^n = u\mu^n = q\sigma$ proves that $(s \mathrel{|\!\gtrdot} q, \mu)$ is solvable. Otherwise, if u is a subterm of some $x\mu$ with $x \in \mathcal{W}$ then there is some i such that $x \in \mathcal{V}(s\mu^i)$. Hence, there is a position p such that $s\mu^{i+1}|_p = u$. Again, $s\mu^{i+1+n}|_p = u\mu^n = q\sigma$ proves that $(s \mathrel{|\!\gtrdot} q, \mu)$ is solvable.

For the other direction of the equivalence we assume that $(s \mathrel{|\!\gtrdot} q, \mu)$ is solvable, so let n, p, σ be given with $s\mu^n|_p = q\sigma$. If p is a non-variable position of s then we are done as the matching problem $(u \gtrdot q, \mu)$ for the corresponding subterm $u = s|_p$ is obviously satisfiable.

Otherwise, there must be a number $0 \leq i < n$ such that p is a non-variable position of $s\mu^{i+1}$ and either p is a variable position of $s\mu^i$ or $p \notin \mathit{Pos}(s\mu^i)$. In both cases there must be a variable x and a position p' such that $x \in \mathcal{V}(s\mu^i) \subseteq \mathcal{W}$ and $x\mu|_{p'} = s\mu^{i+1}|_p$. We choose the non-variable subterm $u = x\mu|_{p'}$ of $x\mu$. Then indeed the matching problem $(u \gtrdot q, \mu)$ is solvable since

$$u\mu^{n-(i+1)} = x\mu|_{p'}\mu^{n-(i+1)} = s\mu^{i+1}|_p\mu^{n-(i+1)} = s\mu^n|_p = q\sigma. \qquad \square$$

Proof of Theorem 8.17.

(i) To prove confluence one can show that $\Rightarrow$ has the diamond-property by a simple case-distinction.

To show termination of $\Rightarrow$ first note that no transformation rule increases the terms in the right-hand sides of a matching problems. Thus, the last three rules can only be applied finitely often. But since every sequence of transformations with rule (i) in the end triggers an application of rule (iii) or (iv), also rule (i) cannot be used infinitely often.

(ii) If $\mathcal{M} \Rightarrow \bot$ due to rule (iii) then $s \gtrdot q \in \mathcal{M}$ with $s = f(\dots)$ and $q = g(\dots)$ with $f \neq q$. But then for every $n \in \mathbb{N}$ the terms $s\mu^n = f(\dots)$ and $q\sigma = g(\dots)$ are different. Hence, $\mathcal{M}$ is not solvable.

If $\mathcal{M} \Rightarrow \bot$ due to rule (ii) then $x \gtrdot q \in \mathcal{M}$ with $x \in \mathcal{V} \setminus \mathcal{V}_{\mathit{incr}}$ and $q = f(\dots)$. But since x is not an increasing variable we know that $x\mu^n \in \mathcal{V}$ for all $n \in \mathbb{N}$. Thus, the terms $x\mu^n$ and $q\sigma = f(\dots)$ are different for all n. Hence, $\mathcal{M}$ is not solvable.

(iii) We first consider rule (i) for $\mathcal{M} = \{s_1 \gtrdot q_1, \dots, s_k \gtrdot q_k\}$.

$$\begin{aligned} \mathcal{M} \text{ is solvable} &\Leftrightarrow \exists n, \sigma : s_1\mu^n = q_1\sigma \wedge \cdots \wedge s_k\mu^n = q_k\sigma \\ &\Leftrightarrow \exists n, \sigma' : s_1\mu^{n+1} = q_1\sigma' \wedge \cdots \wedge s_k\mu^{n+1} = q_k\sigma' \\ &\Leftrightarrow \exists n, \sigma' : (s_1\mu)\mu^n = q_1\sigma' \wedge \cdots \wedge (s_k\mu)\mu^n = q_k\sigma' \\ &\Leftrightarrow \mathcal{M}' = \{s_1\mu \gtrdot q_1, \dots, s_1\mu \gtrdot q_1\} \text{ is solvable} \end{aligned}$$

For rule (iv) the result follows from the fact that $f(s_1, \dots, s_k)\mu^n = f(q_1, \dots, q_k)\sigma$ iff $s_i\mu^n = q_i\sigma$ for all $1 \leq i \leq k$.

(iv) If $\mathcal{M}$ is solvable then due to (ii) and (iii) $\mathcal{M}$ cannot be reduced to $\perp$ by $\Rightarrow$. So, let $\mathcal{M}'$ be a normal form of $\mathcal{M}$ w.r.t. $\Rightarrow$. Then, obviously $\mathcal{M}'$ has the form $\{s_1 \gtrdot x_1, \dots, s_k \gtrdot x_k\}$ and $\mathcal{M}'$ is solvable due to (iii). Thus, there is a number n and a substitution σ such that for all $1 \leq i \leq k$ the equality $s_i\mu^n = x_i\sigma$ is valid. Hence, for all $i \neq j$ with $x_i = x_j$ the identity problem $(s_i \approxeq s_j, \mu)$ is solvable.

For the other direction let $\mathcal{M} \Rightarrow^* \mathcal{M}' = \{s_1 \gtrdot x_1, \dots, s_k \gtrdot x_k\}$ where for every $i \neq j$ with $x_i = x_j$ there is some n_{ij} with $s_i\mu^{n_{ij}} = s_j\mu^{n_{ij}}$. Let n be the maximum of all n_{ij}. Then, obviously $s_i\mu^n = s_j\mu^n$ for all these i and j. We define $\sigma = \{x_1/s_1\mu^n, \dots, x_k/s_k\mu^n\}$. First note, that by construction σ contains no conflicting assignments. But as then $s_i\mu^n = x_i\sigma$ is valid for all $1 \leq i \leq k$ we know that $\mathcal{M}'$ is solvable. Using (iii) we finally conclude that $\mathcal{M}$ is solvable. □

Proof of Theorem 8.20. One can easily show that in the k-th iteration the set S contains the elements of the set S_k.

$$\begin{aligned} S_k = \{(x, p, u) \mid\ & x \in \mathcal{V}_{incr} \wedge x \neq u \wedge \exists m \leq k : \\ & (s\mu^m|_p = x \wedge u = t\mu^m|_p) \vee (t\mu^m|_p = x \wedge u = s\mu^m|_p)\} \end{aligned}$$

Since the correctness of steps (i)-(vi) was already explained in the explanation of the algorithm, we only prove the correctness of step (viii). So, let (x, p_1, u_1) and (x, p_2, u_2) be elements of some S_k and let m_1, m_2 be given such that w.l.o.g. for both $i = 1$ and $i = 2$ we have $s\mu^{m_i}|_{p_i} = x$ and $t\mu^{m_i}|_{p_i} = u_i$ where x is an increasing variable with $x \neq u_i$. If the identity problem $(s \approxeq t, \mu)$ is not solvable then there is nothing to show. Otherwise, there is some n with $s\mu^n = t\mu^n$. Since, $s\mu^{m_i} \neq t\mu^{m_i}$ we know that $n > m_i$ for both i. Hence, we can conclude the following equalities.

$$x\mu^{n-m_i} = s\mu^{m_i}|_{p_i}\mu^{n-m_i} = s\mu^n|_{p_i} = t\mu^n|_{p_i} = t\mu^{m_i}|_{p_i}\mu^{n-m_i} = u_i\mu^{n-m_i}$$

If we have applied (viii–a) then this directly leads to a contradiction since $u_1\mu^n = x\mu^n = u_2\mu^n$ proves that u_1 and u_2 are unifiable.

Otherwise, we have applied (viii–b) where w.l.o.g. $p_1 < p_2$. Since $s\mu^{m_1}|_{p_1}$ is the variable x we must apply μ at least one more time to reach the position p_2 and thus, $m_1 < m_2$. And as $x\mu^{n-m_1} = u_1\mu^{n-m_1}$ there must be some smallest number n' such that $x\mu^{n'-m_1} = u_1\mu^{n'-m_1}$ is valid. From $x \neq u_1$ we conclude $n' > m_1$ and from $s\mu^{n'}|_{p_1} = x\mu^{n'-m_1} = u_1\mu^{n'-m_1} = t\mu^{n'}|_{p_1}$ we derive that also the subterms $s\mu^{n'}|_{p_2}$ of $s\mu^{n'}|_{p_1}$ and $t\mu^{n'}|_{p_2}$ of $t\mu^{n'}|_{p_1}$ are identical. Again, $n' > m_2$ must hold and we obtain $x\mu^{n'-m_2} = u_2\mu^{n'-m_2}$. But this is a contradiction to the minimality of n' since $u_1 = u_2$ and $n' - m_2 < n' - m_1$.

To prove termination of the algorithm we already have argued in the explanation that we can detect all solvable identity problems and all those problems which have a stationary conflict. Thus, it remains to prove that all infinite problems can be detected. To this end we start with giving three observations on infinite identity problems.

First, if $(s \approxeq t, \mu)$ is infinite then $(s\mu \approxeq t\mu, \mu)$ is infinite.

Second, if $(s \approxeq t, \mu)$ is infinite then there is no position p such that $(s|_p \approxeq t|_p, \mu)$ has a stationary conflict. The reason is that one would obtain a stationary conflict of $(s \approxeq t, \mu)$ in contradiction to $(s \approxeq t, \mu)$ being infinite.

And third, whenever $(s \approxeq t, \mu)$ is infinite then there is some position p such that $s|_p \neq t|_p$, at least one of the terms $s|_p$ or $t|_p$ is an increasing variable, and $(s|_p\mu \approxeq t|_p\mu, \mu)$ is infinite. This can be proven as follows. Since $(s \approxeq t, \mu)$ is not solvable there must be at least one maximal shared position p of s and t such that $(s|_p \approxeq t|_p, \mu)$ is not solvable. Due to the second observation we know that $(s|_p \approxeq t|_p, \mu)$ again is infinite. Moreover, using the maximality of p we conclude that one of the terms $s|_p$ or $t|_p$ is a variable. And since $(s|_p \approxeq t|_p, \mu)$ is infinite this variable must be increasing. Finally, by the first observation $(s\mu|_p \approxeq t\mu|_p, \mu)$ is infinite.

Now we show that in a hypothetical infinite run of the algorithm we will put an infinite sequence of triples into S where the corresponding positions $p_0, p_0p_1, p_0p_1p_2, \dots$ are getting longer and longer. Since $(s \approxeq t, \mu)$ is infinite due to the third observation we obtain a position p_0 such that a triple $(x_0, p_0, s|_{p_0})$ or $(x_0, p_0, t|_{p_0})$ will be added to S. Moreover, $(s\mu|_{p_0} \approxeq t\mu|_{p_0}, \mu)$ is infinite. Hence, again using the third observation we obtain a position p_1 such that $(s\mu|_{p_0}\mu|_{p_1} \approxeq t\mu|_{p_0}\mu|_{p_1}, \mu) = (s\mu^2|_{p_0p_1} \approxeq t\mu^2|_{p_0p_1},)$ is infinite and such that one of the (different) terms $s\mu^2|_{p_0p_1}$ or $t\mu^2|_{p_0p_1}$ is an infinite variable x_1. Thus, again the corresponding triple $(x_1, p_0p_1, \dots)$ is added to S. By iterating this reasoning we obviously obtain the desired infinite sequence of triples in S.

As there are only finitely many increasing variables there must be some x which occurs infinitely often in this sequence. Thus, we obtain an infinite subsequence $(x, p_0 \dots p_{i_1}, u_{i_1})$, $(x, p_0 \dots p_{i_2}, u_{i_2})$, $\dots$ where w.l.o.g. $i_1 < i_2 < \dots$ and $p_0 \dots p_{i_1} < p_0 \dots p_{i_2} < \dots$. Due to Kruskal's tree theorem [Kru60] there must be some i_j and i_k such that $i_j < i_k$ and u_{i_j} is embedded in u_{i_k}. If $u_{i_j} = u_{i_k}$ then this is a contradiction to an infinite run of the algorithm since then step (viii–b) will trigger. Otherwise, u_{i_j} is strictly embedded in u_{i_k}. But then u_{i_j} cannot be unified with u_{i_k} since the embedding relation is closed under substitutions. Hence, in that case step (viii–a) will stop the algorithm. ☐

Proof of Corollary 8.22. Let there be a looping reduction of a TRS $\mathcal{R}$ of the following form.

$$s_1 \rightarrow_{\mathcal{R},p_1} s_2 \rightarrow_{\mathcal{R},p_2} \dots s_m \rightarrow_{\mathcal{R},p_m} s_{m+1} = C[s_1\mu]$$

To check whether this reduction is also $\mathcal{Q}$-looping one has to check whether

$$s_1\mu^n \overset{\mathcal{Q}}{\rightarrow}_{\mathcal{R},p_1} s_2\mu^n \overset{\mathcal{Q}}{\rightarrow}_{\mathcal{R},p_2} \dots s_m\mu^n \overset{\mathcal{Q}}{\rightarrow}_{\mathcal{R},p_m} s_{m+1}\mu^n = C\mu^n[s_1\mu^{n+1}]$$

is a valid reduction for all $n \in \mathbb{N}$. This is the case iff all direct subterms u of all $s_i\mu^n|_{p_i} = s_i|_{p_i}\mu^n$ with $1 \leq i \leq m$ are in $\mathcal{Q}$-normal form. And this is the same as demanding that for every $q \in \mathcal{Q}$ the redex problem $(u \mathrel{|\!\!>} q, \mu)$ is not solvable. But solvability of redex problems can be decided using Theorems 8.13, 8.17 (iv), and 8.20.

The reasoning for loops of DP problems is completely identical. ☐

List of Processors

$\mathcal{A}$-Transformation 90, 92

Argument Filter 62

Dependency Graph 19

Edge Deletion by Head Symbols 62

Forward Instantiation 72

Full Labeling 111

Graph Decomposition 19

Instantiation 71

Loop-Detection 132

Needed Rules 45

Needed Rules and Reduction Pairs 51

Needed Rules for Applicative DP Problems 95

Positional Narrowing 79

$\mathcal{Q}$-Reduction 33, 34

Reduction Pair 38

Reduction Pair and Needed Rules 50

Reduction Pair and Needed Rules w.r.t. an Argument Filter 57

Reduction Pair and Needed Rules w.r.t. an Argument Filter for Applicative DP Problems 99

Reduction Pair and Usable Rules w.r.t. an Argument Filter 55

Rewriting 75

Rule Removal 53

Semantic Labeling 107, 113

Semantic Labeling and Unlabeling 124

Semantic Labeling for Quasi-Models 115, 117

Subterm Criterion 63
Switch to Innermost Termination 24
Switch to Termination 134
Usable Rules 28

Index

$a\text{-}ar(\cdot)$, 88
$\mathcal{A}$-Transformation, 88
$\mathcal{A}_\pi$-Transformation, 98
Algebra, 105
Applicative Arity, 87
Applicative DP Problem, 85
Applicative Signature, 85, 88
Applicative TRS, 85
$ar(\cdot)$, 7
Argument Filter, 38
Arity, 7

Cap-Function, 22
    Improved Estimation, 23
$\mathcal{C}_\varepsilon$, 43
Chain, 11
Chain Identifying Processor, 123
Collapsing Rule, 7
$Comp(\cdot)$, 43
Cycle
    of a Graph, 19
    of a Substitution, 138

Decreasing Rules, $\mathcal{D}ecr$, 114
    $\mathcal{D}ecr_{\mathcal{H}}, \mathcal{D}ecr_{\neg\mathcal{H}}$, 116
Defined Symbol, 10
Dependency Graph, 18
    Estimation, 18, 22
    Star-Estimation, 31
Dependency Pair, 10
Domain, $Dom(\cdot)$, 7
$DP(\cdot)$, 10
DP Problem, 13

$ECap(\cdot)$, 22
$enfc(\cdot)$, 78
$\mathcal{EU}(\cdot)$, 28, 54

Full Labeling, 110
Full Rewriting, 8

Generalized TRS, 22

$\mathcal{H}(\cdot)$, 61
Head Symbol, 61

$\mathcal{I}$-Transformation, 44
$\mathcal{I}_\pi$-Transformation, 58
$\mathcal{I}'$-Transformation, 94
$\mathcal{I}'_\pi$-Transformation, 99
$ICap(\cdot)$, 23
Identity Problem, 135
Increasing Variable, 137
Infinite Identity Problem, 139
$\mathcal{IU}(\cdot)$, 29, 55

$Lab(\cdot)$, 105
Labeling Function, 105
Labeling Map, 105
Left-Linear, 7
$lhs(\cdot)$, 7
Linear, 7
Loop, 130, 131

Matching Problem, 135, 136
Model, 105

$\mathcal{N}(\cdot)$, 42, 56
$\mathcal{N}'(\cdot)$, 99
Narrowing, 79
Needed Rules, 42
    w.r.t. an Argument Filter, 56
    w.r.t. an Argument Filter for Applicative DP Problems, 99
$NF(\cdot)$, 7, 8
$nfc(\cdot)$, 74
Normal Form, 7

$\overline{\mathcal{P}}$, 105
$\underline{\mathcal{P}}$, 117
Pair-Graph, 11
π-Proper, 98
$Pos(\cdot)$, 7
Processor, 13
Proper, 88

$\overline{\mathcal{Q}}$, 105
$\underline{\mathcal{Q}}$, 112
$\mathcal{Q}$-Looping, 131
$\mathcal{Q}$-Normal Form, 8
$\mathcal{Q}$-Restricted Rewriting, 8
$\mathcal{Q} - \mathcal{R}$ Normal Form Condition, 74
  Estimation, 78
$\mathcal{Q}$-Termination, 8
Quasi-Labeled Pair-Graph, 117
Quasi-Model, 114
Quasi-Simplification Order, 41

$\overline{\mathcal{R}}$, 105
Redex, 7
Redex Problem, 135
Reduction Order, 37
Reduction Pair, 37
Regarded Position, 54
$RegPos_\pi(\cdot)$, 54
Right-Linear, 7
Root, $root(\cdot)$, 7

SCC, 19
Simplification Order, 37
Stationary Conflict, 139
Substitution, 7

Termination, 7
Tuple Symbol, 10

$\mathcal{U}(\cdot)$, 28, 54
$Unlab(\cdot)$, 105
Usable Rules, 28
  Improved Estimation, 29
Usable Rules w.r.t. an Argument Filter, 54
  Improved Estimation, 55

$\mathcal{V}(\cdot)$, 7
Variable Assignment, 105
Variable Condition, 7

$\mathcal{Y}$-Transformation, 89

$\mathcal{Z}$-Transformation, 91

Curriculum Vitae

Name	René Thiemann
Geburtsdatum	2. November 1976
Geburtsort	Stadtlohn

Bildungsgang

1987–1996	St. Pius Gymnasium Coesfeld Abschluss: Allgemeine Hochschulreife
1996–1997	Zivildienst im Martinistift Nottuln
1997–2002	Studium der Informatik an der RWTH Aachen Abschluss: Diplom
2002–2007	Wissenschaftlicher Angestellter am Lehr- und Forschungsgebiet Informatik 2, RWTH Aachen
2007–	Wissenschaftlicher Mitarbeiter am Institut für Informatik, Computational Logic, Universität Innsbruck

Aachener Informatik-Berichte

This list contains all technical reports published during the past five years. A complete list of reports dating back to 1987 is available from `http://aib.informatik.rwth-aachen.de/`. To obtain copies consult the above URL or send your request to: Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Email: `biblio@informatik.rwth-aachen.de`

2001-01 *	Jahresbericht 2000
2001-02	Benedikt Bollig, Martin Leucker: Deciding LTL over Mazurkiewicz Traces
2001-03	Thierry Cachat: The power of one-letter rational languages
2001-04	Benedikt Bollig, Martin Leucker, Michael Weber: Local Parallel Model Checking for the Alternation Free mu-Calculus
2001-05	Benedikt Bollig, Martin Leucker, Thomas Noll: Regular MSC Languages
2001-06	Achim Blumensath: Prefix-Recognisable Graphs and Monadic Second-Order Logic
2001-07	Martin Grohe, Stefan Wöhrle: An Existential Locality Theorem
2001-08	Mareike Schoop, James Taylor (eds.): Proceedings of the Sixth International Workshop on the Language-Action Perspective on Communication Modelling
2001-09	Thomas Arts, Jürgen Giesl: A collection of examples for termination of term rewriting using dependency pairs
2001-10	Achim Blumensath: Axiomatising Tree-interpretable Structures
2001-11	Klaus Indermark, Thomas Noll (eds.): Kolloquium Programmiersprachen und Grundlagen der Programmierung
2002-01 *	Jahresbericht 2001
2002-02	Jürgen Giesl, Aart Middeldorp: Transformation Techniques for Context-Sensitive Rewrite Systems
2002-03	Benedikt Bollig, Martin Leucker, Thomas Noll: Generalised Regular MSC Languages
2002-04	Jürgen Giesl, Aart Middeldorp: Innermost Termination of Context-Sensitive Rewriting
2002-05	Horst Lichter, Thomas von der Maßen, Thomas Weiler: Modelling Requirements and Architectures for Software Product Lines
2002-06	Henry N. Adorna: 3-Party Message Complexity is Better than 2-Party Ones for Proving Lower Bounds on the Size of Minimal Nondeterministic Finite Automata
2002-07	Jörg Dahmen: Invariant Image Object Recognition using Gaussian Mixture Densities
2002-08	Markus Mohnen: An Open Framework for Data-Flow Analysis in Java
2002-09	Markus Mohnen: Interfaces with Default Implementations in Java
2002-10	Martin Leucker: Logics for Mazurkiewicz traces

2002-11	Jürgen Giesl, Hans Zantema: Liveness in Rewriting
2003-01 *	Jahresbericht 2002
2003-02	Jürgen Giesl, René Thiemann: Size-Change Termination for Term Rewriting
2003-03	Jürgen Giesl, Deepak Kapur: Deciding Inductive Validity of Equations
2003-04	Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke: Improving Dependency Pairs
2003-05	Christof Löding, Philipp Rohde: Solving the Sabotage Game is PSPACE-hard
2003-06	Franz Josef Och: Statistical Machine Translation: From Single-Word Models to Alignment Templates
2003-07	Horst Lichter, Thomas von der Maßen, Alexander Nyßen, Thomas Weiler: Vergleich von Ansätzen zur Feature Modellierung bei der Softwareproduktlinienentwicklung
2003-08	Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke: Mechanizing Dependency Pairs
2004-01 *	Fachgruppe Informatik: Jahresbericht 2003
2004-02	Benedikt Bollig, Martin Leucker: Message-Passing Automata are expressively equivalent to EMSO logic
2004-03	Delia Kesner, Femke van Raamsdonk, Joe Wells (eds.): HOR 2004 – 2nd International Workshop on Higher-Order Rewriting
2004-04	Slim Abdennadher, Christophe Ringeissen (eds.): RULE 04 – Fifth International Workshop on Rule-Based Programming
2004-05	Herbert Kuchen (ed.): WFLP 04 – 13th International Workshop on Functional and (Constraint) Logic Programming
2004-06	Sergio Antoy, Yoshihito Toyama (eds.): WRS 04 – 4th International Workshop on Reduction Strategies in Rewriting and Programming
2004-07	Michael Codish, Aart Middeldorp (eds.): WST 04 – 7th International Workshop on Termination
2004-08	Klaus Indermark, Thomas Noll: Algebraic Correctness Proofs for Compiling Recursive Function Definitions with Strictness Information
2004-09	Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Parameterized Power Domination Complexity
2004-10	Zinaida Benenson, Felix C. Gärtner, Dogan Kesdogan: Secure Multi-Party Computation with Security Modules
2005-01 *	Fachgruppe Informatik: Jahresbericht 2004
2005-02	Maximillian Dornseif, Felix C. Gärtner, Thorsten Holz, Martin Mink: An Offensive Approach to Teaching Information Security: “Aachen Summer School Applied IT Security”
2005-03	Jürgen Giesl, René Thiemann, Peter Schneider-Kamp: Proving and Disproving Termination of Higher-Order Functions
2005-04	Daniel Mölle, Stefan Richter, Peter Rossmanith: A Faster Algorithm for the Steiner Tree Problem
2005-05	Fabien Pouget, Thorsten Holz: A Pointillist Approach for Comparing Honeypots
2005-06	Simon Fischer, Berthold Vöcking: Adaptive Routing with Stale Information

2005-07	Felix C. Freiling, Thorsten Holz, Georg Wicherski: Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks
2005-08	Joachim Kneis, Peter Rossmanith: A New Satisfiability Algorithm With Applications To Max-Cut
2005-09	Klaus Kursawe, Felix C. Freiling: Byzantine Fault Tolerance on General Hybrid Adversary Structures
2005-10	Benedikt Bollig: Automata and Logics for Message Sequence Charts
2005-11	Simon Fischer, Berthold Vöcking: A Counterexample to the Fully Mixed Nash Equilibrium Conjecture
2005-12	Neeraj Mittal, Felix Freiling, S. Venkatesan, Lucia Draque Penso: Efficient Reductions for Wait-Free Termination Detection in Faulty Distributed Systems
2005-13	Carole Delporte-Gallet, Hugues Fauconnier, Felix C. Freiling: Revisiting Failure Detection and Consensus in Omission Failure Environments
2005-14	Felix C. Freiling, Sukumar Ghosh: Code Stabilization
2005-15	Uwe Naumann: The Complexity of Derivative Computation
2005-16	Uwe Naumann: Syntax-Directed Derivative Code (Part I: Tangent-Linear Code)
2005-17	Uwe Naumann: Syntax-directed Derivative Code (Part II: Intraprocedural Adjoint Code)
2005-18	Thomas von der Maßen, Klaus Müller, John MacGregor, Eva Geisberger, Jörg Dörr, Frank Houdek, Harbhajan Singh, Holger Wußmann, Hans-Veit Bacher, Barbara Paech: Einsatz von Features im Software-Entwicklungsprozess - Abschlußbericht des GI-Arbeitskreises “Features”
2005-19	Uwe Naumann, Andre Vehreschild: Tangent-Linear Code by Augmented LL-Parsers
2005-20	Felix C. Freiling, Martin Mink: Bericht über den Workshop zur Ausbildung im Bereich IT-Sicherheit Hochschulausbildung, berufliche Weiterbildung, Zertifizierung von Ausbildungsangeboten am 11. und 12. August 2005 in Köln organisiert von RWTH Aachen in Kooperation mit BITKOM, BSI, DLR und Gesellschaft fuer Informatik (GI) e.V.
2005-21	Thomas Noll, Stefan Rieger: Optimization of Straight-Line Code Revisited
2005-22	Felix Freiling, Maurice Herlihy, Lucia Draque Penso: Optimal Randomized Fair Exchange with Secret Shared Coins
2005-23	Heiner Ackermann, Alantha Newman, Heiko Röglin, Berthold Vöcking: Decision Making Based on Approximate and Smoothed Pareto Curves
2005-24	Alexander Becher, Zinaida Benenson, Maximillian Dornseif: Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks
2006-01 *	Fachgruppe Informatik: Jahresbericht 2005
2006-02	Michael Weber: Parallel Algorithms for Verification of Large Systems
2006-03	Michael Maier, Uwe Naumann: Intraprocedural Adjoint Code Generated by the Differentiation-Enabled NAGWare Fortran Compiler
2006-04	Ebadollah Varnik, Uwe Naumann, Andrew Lyons: Toward Low Static Memory Jacobian Accumulation

2006-05	Uwe Naumann, Jean Utke, Patrick Heimbach, Chris Hill, Derya Ozyurt, Carl Wunsch, Mike Fagan, Nathan Tallent, Michelle Strout: Adjoint Code by Source Transformation with OpenAD/F
2006-06	Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Divide-and-Color
2006-07	Thomas Colcombet, Christof Löding: Transforming structures by set interpretations
2006-08	Uwe Naumann, Yuxiao Hu: Optimal Vertex Elimination in Single-Expression-Use Graphs
2006-09	Tingting Han, Joost-Pieter Katoen: Counterexamples in Probabilistic Model Checking
2006-10	Mesut Günes, Alexander Zimmermann, Martin Wenig, Jan Ritzerfeld, Ulrich Meis: From Simulations to Testbeds - Architecture of the Hybrid MCG-Mesh Testbed
2006-11	Bastian Schlich, Michael Rohrbach, Michael Weber, Stefan Kowalewski: Model Checking Software for Microcontrollers
2006-12	Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker: Replaying Play in and Play out: Synthesis of Design Models from Scenarios by Learning
2006-13	Wong Karianto, Christof Löding: Unranked Tree Automata with Sibling Equalities and Disequalities
2006-14	Danilo Beuche, Andreas Birk, Heinrich Dreier, Andreas Fleischmann, Heidi Galle, Gerald Heller, Dirk Janzen, Isabel John, Ramin Tavakoli Kolagari, Thomas von der Maßen, Andreas Wolfram: Report of the GI Work Group "Requirements Management Tools for Product Line Engineering"
2006-15	Sebastian Ullrich, Jakob T. Valvoda, Torsten Kuhlen: Utilizing optical sensors from mice for new input devices
2006-16	Rafael Ballagas, Jan Borchers: Selexels: a Conceptual Framework for Pointing Devices with Low Expressiveness
2006-17	Eric Lee, Henning Kiel, Jan Borchers: Scrolling Through Time: Improving Interfaces for Searching and Navigating Continuous Audio Timelines
2007-01 *	Fachgruppe Informatik: Jahresbericht 2006
2007-02	Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl: SAT Solving for Termination Analysis with Polynomial Interpretations
2007-03	Jürgen Giesl, René Thiemann, Stephan Swiderski, and Peter Schneider-Kamp: Proving Termination by Bounded Increase
2007-04	Jan Buchholz, Eric Lee, Jonathan Klein, and Jan Borchers: coJIVE: A System to Support Collaborative Jazz Improvisation
2007-05	Uwe Naumann: On Optimal DAG Reversal
2007-06	Joost-Pieter Katoen, Thomas Noll, and Stefan Rieger: Verifying Concurrent List-Manipulating Programs by LTL Model Checking
2007-07	Alexander Nyßen, Horst Lichter: MeDUSA - MethoD for UML2-based Design of Embedded Software Applications
2007-08	Falk Salewski and Stefan Kowalewski: Achieving Highly Reliable Embedded Software: An empirical evaluation of different approaches

2007-09	Tina Kraußer, Heiko Mantel, and Henning Sudbrock: A Probabilistic Justification of the Combining Calculus under the Uniform Scheduler Assumption
2007-10	Martin Neuhäußer, Joost-Pieter Katoen: Bisimulation and Logical Preservation for Continuous-Time Markov Decision Processes
2007-11	Klaus Wehrle: 6. Fachgespräch Sensornetzwerke
2007-12	Uwe Naumann: An L-Attributed Grammar for Adjoint Code
2007-13	Uwe Naumann, Michael Maier, Jan Riehme, and Bruce Christianson: Second-Order Adjoints by Source Code Manipulation of Numerical Programs
2007-14	Jean Utke, Uwe Naumann, Mike Fagan, Nathan Tallent, Michelle Strout, Patrick Heimbach, Chris Hill, and Carl Wunsch: OpenAD/F: A Modular, Open-Source Tool for Automatic Differentiation of Fortran Codes
2007-15	Volker Stolz: Temporal assertions for sequential and current programs
2007-16	Sadeq Ali Makram, Mesut Güneç, Martin Wenig, Alexander Zimmermann: Adaptive Channel Assignment to Support QoS and Load Balancing for Wireless Mesh Networks

* These reports are only available as a printed version.
Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.