

12-1-2009

Term rewriting with built-in numbers and collection data structures

Stephan Falke

Follow this and additional works at: https://digitalrepository.unm.edu/cs_etds

Recommended Citation

Falke, Stephan. "Term rewriting with built-in numbers and collection data structures." (2009). https://digitalrepository.unm.edu/cs_etds/5

This Dissertation is brought to you for free and open access by the Engineering ETDs at UNM Digital Repository. It has been accepted for inclusion in Computer Science ETDs by an authorized administrator of UNM Digital Repository. For more information, please contact disc@unm.edu.

Stephan Falke

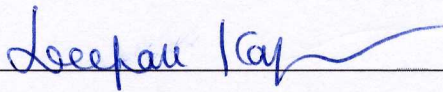
Candidate

Computer Science

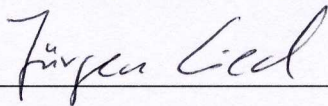
Department

This dissertation is approved, and it is acceptable in quality and form for publication:

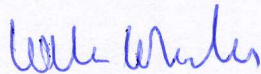
Approved by the Dissertation Committee:



Deepak Kapur, Chairperson



Jürgen Giesl



William McCune



Robert Veroff

Term Rewriting with Built-In Numbers and Collection Data Structures

by

Stephan Falke

Dipl.-Inform., RWTH Aachen University, Germany, 2004

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

December, 2009

©2009, Stephan Falke

Acknowledgments

First and foremost, I would like to thank Deepak Kapur for his support and help in developing the ideas presented in this dissertation. His insistence on intuitive explanations and illustrative examples has greatly improved the presentation of the technical material.

Special thanks go to Jürgen Giesl for detailed and helpful comments on drafts of this dissertation, for our enlightening discussions in Aachen, and for inviting me to present my work at several MOVES seminars.

Thanks are also due to Bob Veroff for valuable comments on this dissertation and to Bill McCune for his support in experimenting with `Mace4` for the automatic generation of polynomial interpretations.

Sincere thanks go to Peter Schneider-Kamp for proof-reading much of this document and commenting on my ideas.

The implementation of parts of this work in the termination prover `AProVE` would not have been possible without the help of the following members of the `AProVE`-team: Peter Schneider-Kamp, Carsten Fuhs, Carsten Otto, and Lars Noschinski.

Finally, my thanks go to Lynne Jacobsen and Lourdes McKenna for their administrative assistance.

This work has been partially supported by NSF grants CCF-0113611, CCF-0541315, and CNS-0831462.

Stephan Falke

Term Rewriting with Built-In Numbers and Collection Data Structures

by

Stephan Falke

ABSTRACT OF DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

December, 2009

Term Rewriting with Built-In Numbers and Collection Data Structures

by

Stephan Falke

Dipl.-Inform., RWTH Aachen University, Germany, 2004

Ph.D., Computer Science, University of New Mexico, 2009

Abstract

Term rewrite systems have been extensively used in order to model computer programs for the purpose of formal verification. This is in particular true if the termination behavior of computer programs is investigated, and automatic termination proving for term rewrite systems has received increased interest in recent years. Ordinary term rewrite systems, however, exhibit serious drawbacks. First, they do not provide a tight integration of natural numbers or integers. Since the pre-defined semantics of these primitive data types cannot be utilized, reasoning about termination of ordinary term rewrite systems operating on numbers is often cumbersome or even impossible. Second, ordinary term rewrite system cannot accurately model collection data structures such as sets or multisets which are supported by many high-level programming languages such as *Maude* or *OCaml*.

This dissertation introduces a new class of term rewrite systems that addresses both of these drawbacks and thus makes it possible to accurately model computer

programs using a high level of abstraction in a natural formalism. Then, the problem of automatically proving termination for this new class of term rewrite systems is investigated. The resulting dependency pair framework provides a flexible and modular method for proving termination. In addition to unrestricted rewriting, termination of rewriting with the innermost strategy or a context-sensitive rewriting strategy is investigated as well.

The techniques for proving termination that are developed in this dissertation have been implemented in the well-known termination prover **AProVE**. An empirical evaluation shows that the implementation succeeds in automatically proving termination of a large collection of computer programs that are modeled using the new class of term rewrite systems developed in this work.

Next, the use of this new class of term rewrite systems in the context of inductive theorem proving is investigated. This makes it possible to reason about the semantics of computer programs. The inductive theorem proving method developed in this dissertation provides a tight integration of inductive reasoning with a decision procedure, thus resulting in a high degree of automation.

Finally, conditions under which the inductive theorem proving method is guaranteed to succeed in proving or disproving a conjecture without any user intervention are identified. Thus, the inductive theorem proving method can be applied as a “black box” if these conditions are satisfied.

The inductive theorem proving method and checks for the conditions under which it provides a decision procedure have been implemented in the prototype prover **Sail2**. An empirical evaluation shows that **Sail2** is very efficient, and the high degree of automation makes it possible to use **Sail2** in a push-button mode for formal program verification.

Contents

| | |
|---|-----------|
| List of Figures | xv |
| 1 Introduction | 1 |
| 1.1 Using TRSs for Formal Verification | 2 |
| 1.2 Shortcomings of Ordinary TRSs | 3 |
| 1.3 Overview of Contributions | 7 |
| 1.4 Detailed Outline | 9 |
| 1.4.1 Termination and Operational Termination | 11 |
| 1.4.2 Termination under Strategies | 13 |
| 1.4.3 Inductive Reasoning | 17 |
| 1.5 Structure of the Dissertation | 18 |
| 1.6 Published Contributions and New Contributions | 23 |
| 1.7 Related Work | 25 |
| 2 Many-Sorted Term Rewriting | 27 |

Contents

| | | |
|----------|--|-----------|
| 2.1 | Terms and Substitutions | 27 |
| 2.2 | Relations | 32 |
| 2.3 | Equations and Rewrite Rules | 33 |
| 2.4 | Equational Rewriting | 36 |
| 3 | Constrained Equational Rewrite Systems | 41 |
| 3.1 | Built-In Numbers as Canonizable Theories | 41 |
| 3.2 | Canonizable Collection Data Structures | 47 |
| 3.3 | Constrained Equational Rewrite Systems | 48 |
| 3.4 | Innermost and Restricted Rewriting | 54 |
| 3.5 | Summary | 58 |
| 4 | Translating Imperative Programs | 59 |
| 4.1 | A Simple Imperative Language | 60 |
| 4.2 | Translating Imperative Programs into CERSs | 62 |
| 4.3 | Utilizing Static Program Analysis | 65 |
| 4.4 | Summary | 67 |
| 5 | The Dependency Pair Framework | 68 |
| 5.1 | Dependency Pairs | 69 |
| 5.2 | DP Framework | 72 |
| 5.3 | Summary | 75 |

Contents

| | | |
|----------|--|------------|
| 6 | DP Processors Operating on Dependency Pairs | 76 |
| 6.1 | Unsatisfiable Constraints | 77 |
| 6.2 | Reducible Left-Hand Sides | 78 |
| 6.3 | Dependency Graphs | 79 |
| 6.4 | Reducing Right-Hand Sides | 83 |
| 6.5 | Dependency Pair Narrowing | 84 |
| 6.6 | Subterm Criterion | 87 |
| 6.7 | Summary | 91 |
| 7 | Reduction Pairs | 93 |
| 7.1 | Ordinary Reduction Pairs | 94 |
| 7.2 | $Th_{\mathbb{N}}$ -Reduction Pairs | 97 |
| 7.3 | $Th_{\mathbb{Z}}$ -Reduction Pairs | 103 |
| 7.4 | Summary | 111 |
| 8 | Usable Rules and Function Dependencies | 113 |
| 8.1 | Usable Rules | 114 |
| 8.2 | Removal of Rules | 116 |
| 8.3 | Function Dependencies | 120 |
| 8.4 | Removal of Rules Revisited | 127 |
| 8.5 | Summary | 128 |

| | | |
|-----------|---|------------|
| 9 | Implementation | 130 |
| 9.1 | Implementing EDG | 130 |
| 9.2 | Generation of $\mathcal{Th}_{\mathbb{Z}}$ -Polynomial Interpretations | 133 |
| 9.3 | Summary | 140 |
| 10 | Conditional Rewriting | 142 |
| 10.1 | Conditional CERSs | 144 |
| 10.2 | Termination and Operational Termination | 147 |
| 10.3 | Elimination of Conditions | 151 |
| 10.4 | Summary | 154 |
| 11 | Context-Sensitive Rewriting and Dependency Pairs | 156 |
| 11.1 | Context-Sensitive Rewriting | 158 |
| 11.2 | Context-Sensitive Conditional Rewriting | 163 |
| 11.3 | Context-Sensitive Dependency Pairs | 164 |
| 11.4 | Summary | 171 |
| 12 | Context-Sensitive DP Processors | 173 |
| 12.1 | Dependency Graphs | 174 |
| 12.2 | Subterm Criterion | 177 |
| 12.3 | Reduction Pairs | 178 |
| 12.4 | Function Dependencies | 180 |

Contents

| | | |
|-----------|---|------------|
| 12.5 | Function Dependencies for Strongly Conservative Systems | 184 |
| 12.6 | Implementation | 189 |
| 12.7 | Summary | 189 |
| 13 | Inductive Theorem Proving with CERSs | 191 |
| 13.1 | Preliminaries | 194 |
| 13.2 | Quasi-Reductivity and Confluence | 198 |
| 13.3 | Inductive Theorem Proving | 201 |
| 13.4 | Summary | 210 |
| 14 | Inductive Theorem Proving as a Decision Procedure | 211 |
| 14.1 | Simple Decidable Conjectures | 212 |
| 14.2 | Simple Decidable Conjectures with Nesting | 218 |
| 14.3 | Safe Generalizations | 224 |
| 14.4 | Complex Conjectures | 230 |
| 14.5 | Implementation | 235 |
| 14.6 | Summary | 236 |
| 15 | Conclusions and Evaluation | 238 |
| 15.1 | Empirical Evaluation | 240 |
| 15.1.1 | Termination Analysis | 240 |
| 15.1.2 | Inductive Theorem Proving | 241 |

Contents

| | |
|---|------------|
| 15.2 Future Work | 243 |
| A Proofs | 245 |
| A.1 Proofs from Chapter 2 | 245 |
| A.2 Proofs from Chapter 3 | 248 |
| A.3 Proofs from Chapter 4 | 251 |
| A.4 Proofs from Chapter 5 | 251 |
| A.5 Proofs from Chapter 6 | 254 |
| A.6 Proofs from Chapter 7 | 261 |
| A.7 Proofs from Chapter 8 | 266 |
| A.8 Proofs from Chapter 9 | 273 |
| A.9 Proofs from Chapter 10 | 274 |
| A.10 Proofs from Chapter 11 | 281 |
| A.11 Proofs from Chapter 12 | 291 |
| A.12 Proofs from Chapter 13 | 309 |
| A.13 Proofs from Chapter 14 | 316 |
| B Evaluation | 326 |
| B.1 Termination | 326 |
| B.2 Context-Sensitive Termination | 329 |
| B.3 Induction | 330 |

Contents

References

332

List of Figures

| | | |
|------|--|-----|
| 1.1 | Maude module for Example 1.3. | 7 |
| 1.2 | OCaml program for Example 1.3. | 8 |
| 1.3 | Maude module for Example 1.8. | 15 |
| 3.1 | Numbers as canonizable theories. | 46 |
| 3.2 | Commonly used canonizable collection data structures. | 48 |
| 4.1 | Grammar for a simple imperative programming language. | 61 |
| 9.1 | Transformation rules for the generation of polynomial interpretations. | 137 |
| 9.2 | Obtaining conditions on the parameters. | 139 |
| 10.1 | Derivation rules for the generation of proof trees. | 150 |
| 11.1 | Replacement maps allowed for context-sensitive rewriting. | 161 |
| 13.1 | The inference system \mathcal{I} | 205 |
| 14.1 | The inference system \mathcal{I}' | 225 |

Chapter 1

Introduction

Computer programs, whether in the form of software or hardware, have become ubiquitous. They are embedded in medical equipment, car and aircraft control systems, and electrical power systems, to name just a few examples. For many of these systems, failures would cause severe loss of money, time, or even human life. Therefore, it is mandatory that such systems operate correctly and reliably.

Due to the complexity of modern computer programs, ensuring correctness is a challenging task. It is thus not surprising that computer programs are error-prone and often contain subtle mistakes that are difficult to detect and repair. Several examples of mistakes in computer programs and the losses caused by them are given in [172].

Currently, ensuring reliability of computer programs is usually attempted using extensive simulation and testing. While these approaches have their merits, they are typically inadequate for complex computer programs due to the computational cost of the task. It is prohibitively expensive to perform exhaustive testing on all or even a substantial fraction of possible scenarios due to the large (or even infinite) number of possible configurations of a typical computer program. Due to this limitation,

testing does not provide a proof of correctness.

Formal verification provides an alternative to testing. The aim of formal verification is a (mathematical) proof that the computer program behaves correctly. For this, computer programs are modelled using a mathematical formalism and it is formally proved that this model satisfies its specification. Thus, if formal verification is successful, then it provides a mathematical guarantee of correctness of the computer program, up to the accuracy of the formal model. Notice that, in contrast to testing, formal verification considers *all* possible scenarios.

One of the most important questions about a computer program is whether it is *terminating*, i.e., whether it always produces an answer and does not diverge. One of the oldest results in theoretical computer science (which even predates the first general, programmable computer) states that this “halting problem” is undecidable [166]. Termination is also of interest for reactive systems which are typically assumed to run continuously. For such computer systems, termination means that the system reacts to an input within a finite amount of time, i.e., that the functions performing the actions of the reactive system are terminating.

Complementary to termination is *partial correctness* of a computer program. Partial correctness denotes the property that a computer program behaves correctly (w.r.t. its specification). Termination and partial correctness together constitute *total correctness* of a computer program.

1.1 Using TRSs for Formal Verification

Term rewrite systems (TRSs) have been extensively used in order to provide a convenient model of computer programs that makes it possible to use formal verification techniques. Using TRSs, the correctness of computer programs can be investigated

by considering questions about TRSs.

If a computer program is modeled by a TRS in a non-termination preserving way, i.e., if termination of the TRS implies termination of the computer program, then methods for proving termination of TRSs can be used for proving termination of computer programs. The problem of proving termination of TRSs has been studied extensively in the past, see [173] for a recent survey. An important advantage of recent methods is that they can be fully automated, thus making it possible to prove termination of computer programs without any user intervention.

In contrast to proving termination, investigating the partial correctness of computer programs is more challenging and typically requires substantial user intervention. First, it needs to be determined which properties the computer program should satisfy. This requires insight and expertise, and it is unlikely that this part can be automated. Once the requirements on the computer program have been determined, it needs to be shown that the computer program does indeed satisfy them. Since the requirements are often stated in (extensions of) first-order predicate logic, a complete automation of this task is not possible, either.

Thus, attention is often restricted to certain classes of properties. For computer programs modeled by TRSs, these are often properties that can be expressed by (implicitly universally quantified) equations that express certain aspects of the functions that are defined by the TRS. Since the functions in a TRS are often defined using recursion, proving these equations typically requires inductive reasoning.

1.2 Shortcomings of Ordinary TRSs

Most computer programs and algorithms make use of natural numbers or integers. Ordinary TRSs, however, only support an inelegant handling of those primitives.

Chapter 1. Introduction

Natural numbers can be modeled using a Peano representation with \mathcal{O} and \mathbf{s} (successor), but then ordinary TRSs require a specification of commonly used operators on natural numbers (such as $+$ and $>$) in the form of rewrite rules as well. Using this approach, even basic knowledge about properties of natural numbers is not directly available, thus making reasoning about algorithms more complicated and cumbersome.

Example 1.1. Consider the following simple `while`-loop from an imperative program operating on natural numbers:

```
while (x >= y) {  
    y++  
}
```

Using an ordinary TRS with a Peano representation of natural numbers results in the following rewrite rules:

$$\begin{aligned}\text{eval}(x, y) &\rightarrow \text{if}(\text{geq}(x, y), x, y) \\ \text{if}(\text{true}, x, y) &\rightarrow \text{eval}(x, \mathbf{s}(y)) \\ \text{geq}(x, \mathcal{O}) &\rightarrow \text{true} \\ \text{geq}(\mathcal{O}, \mathbf{s}(y)) &\rightarrow \text{false} \\ \text{geq}(\mathbf{s}(x), \mathbf{s}(y)) &\rightarrow \text{geq}(x, y)\end{aligned}$$

While the `while`-loop is clearly terminating since the difference $x - y$ is decreasing and bounded by 0, it is much harder to (automatically) establish termination of the TRS. Indeed, only a single termination tool for TRSs, namely the well-known tool AProVE [84], could establish termination of this example in the *Termination Competition 2007* [161]. To do so, AProVE relies on a complicated and specialized technique that re-discovers the semantics of `geq` [89]. \triangle

Automatically establishing termination of TRSs is even harder if integers instead of natural numbers are considered. A representation using \mathcal{O} , \mathbf{s} , and \mathbf{p} (predecessor)

Chapter 1. Introduction

results in *non-free constructors* since, e.g., the syntactically distinct term \mathcal{O} , $\mathfrak{s}(\mathfrak{p}(\mathcal{O}))$, and $\mathfrak{p}(\mathfrak{s}(\mathcal{O}))$ all denote the integer 0. This non-freeness can be modeled using the equations $\mathfrak{p}(\mathfrak{s}(x)) \approx \mathfrak{s}(\mathfrak{p}(x))$ and $\mathfrak{p}(\mathfrak{s}(x)) \approx x$ which identify distinct terms that represent the same integer. Automated termination analysis of TRSs with non-free constructors is rarely investigated and much less developed than for ordinary TRSs. The case of *AC*-rewriting (i.e., if certain function symbols are assumed to be associative and commutative) has been studied extensively, see, e.g., [108, 150, 129, 118, 130, 31]. Since the equations for integers are not *AC*, these methods do not apply. The more general method of [81] cannot be applied to integers, either, since the equation $\mathfrak{p}(\mathfrak{s}(x)) \approx x$ is collapsing.

Example 1.2. If the imperative program from Example 1.1 is considered on integers instead of natural numbers, then the resulting rewrite rules are more complex:

$$\begin{aligned}
 \text{eval}(x, y) &\rightarrow \text{if}(\text{geq}(x, y), x, y) \\
 \text{if}(\text{true}, x, y) &\rightarrow \text{eval}(x, \mathfrak{s}(y)) \\
 \text{geq}(\mathcal{O}, \mathcal{O}) &\rightarrow \text{true} \\
 \text{geq}(\mathfrak{p}(\mathcal{O}), \mathcal{O}) &\rightarrow \text{false} \\
 \text{geq}(x, \mathfrak{s}(y)) &\rightarrow \text{geq}(\mathfrak{p}(x), y) \\
 \text{geq}(x, \mathfrak{p}(y)) &\rightarrow \text{geq}(\mathfrak{s}(x), y) \\
 \text{geq}(x, \mathcal{O}) \rightarrow^* \text{true} \mid \text{geq}(\mathfrak{s}(x), \mathcal{O}) &\rightarrow \text{true} \\
 \text{geq}(x, \mathcal{O}) \rightarrow^* \text{false} \mid \text{geq}(\mathfrak{p}(x), \mathcal{O}) &\rightarrow \text{false}
 \end{aligned}$$

Notice that *conditional rewrite rules* are needed in order to define geq . These conditional rules can be transformed into unconditional rules for the purpose of proving termination (see [137] and Chapter 10), but the rewrite relation modulo the set of equations $\mathcal{E} = \{\mathfrak{p}(\mathfrak{s}(x)) \approx \mathfrak{s}(\mathfrak{p}(x)), \mathfrak{p}(\mathfrak{s}(x)) \approx x\}$ is non-terminating because $\text{geq}(\mathcal{O}, \mathfrak{s}(\mathcal{O})) \sim_{\mathcal{E}} \text{geq}(\mathcal{O}, \mathfrak{p}(\mathfrak{s}(\mathfrak{s}(\mathcal{O})))) \xrightarrow{2}_{\mathcal{R}} \text{geq}(\mathfrak{p}(\mathfrak{s}(\mathcal{O})), \mathfrak{s}(\mathcal{O})) \sim_{\mathcal{E}} \text{geq}(\mathcal{O}, \mathfrak{s}(\mathcal{O}))$.¹ \triangle

¹Here, $\sim_{\mathcal{E}}$ denotes equivalence up to the equations in \mathcal{E} and $\rightarrow_{\mathcal{R}}$ denotes application of a rewrite rule from \mathcal{R} , replacing an instance of a left-hand side by the corresponding

In addition to modelling integers, non-free constructors can also be used to model *collection data structures* such as sets or multisets where, for instance, the set $\{a, b\}$ can be represented by the equivalent, but distinct, terms $\text{ins}(a, \text{ins}(b, \emptyset))$ and $\text{ins}(b, \text{ins}(a, \emptyset))$. Notice that many real-life programming languages such as Java and OCaml support collection data structures. Furthermore, the declarative specification and programming language Maude [43] makes it possible to use collection data structures by specifying suitable equational attributes. Thus, support for collection data structures is mandatory in order to accurately model programs written in these high-level programming languages.

Example 1.3. Figure 1.1 contains a Maude module that defines the functions `length` and `reverse` operating on lists. Here, lists are built using the constructors `nil` (empty list), `elem` (single-element lists containing an integer), and `++` (concatenation of two lists). This representation of lists makes it possible to easily parallelize computations, in contrast to the more intuitive representation using `nil` and `cons`. Notice that concatenation of lists is associative and has `nil` as a unit element. Since this unit element gives rise to the collapsing equations $x ++ \text{nil} \approx x$ and $\text{nil} ++ y \approx y$, termination cannot be investigated using existing methods. The same example can also be written in OCaml using the pre-processor Moca [29], see Figure 1.2. These examples can be modeled using the following rewrite rules (where $\langle \cdot \rangle$ is used instead of `elem` and `++` is written $++$):

$$\begin{aligned} \text{length}(\text{nil}) &\rightarrow 0 \\ \text{length}(\langle n \rangle) &\rightarrow 1 \\ \text{length}(k ++ l) &\rightarrow \text{length}(k) + \text{length}(l) \\ \text{reverse}(\text{nil}) &\rightarrow \text{nil} \\ \text{reverse}(\langle n \rangle) &\rightarrow \langle n \rangle \\ \text{reverse}(k ++ l) &\rightarrow \text{reverse}(l) ++ \text{reverse}(k) \end{aligned}$$

instance of a right-hand side. These concepts are formally defined in Chapter 2.

```

fmod LISTS is
  protecting INT .

  sorts List .

  op nil : -> List [ ctor ] .
  op elem : Int -> List [ ctor ] .
  op _ ++ _ : List List -> List [ ctor assoc id: nil ] .
  op length : List -> Int .
  op reverse : List -> List .

  var N : Int .
  var K L : List .

  eq length(nil) = 0 .
  eq length(elem(N)) = 1 .
  eq length(K ++ L) = length(K) + length(L) .
  eq reverse(nil) = nil .
  eq reverse(elem(N)) = elem(N) .
  eq reverse(K ++ L) = reverse(L) ++ reverse(K) .
endfm

```

Figure 1.1: Maude module for Example 1.3.

In order to model the semantical properties of list concatenation, the set of equations $\mathcal{E} = \{x ++ (y ++ z) \approx (x ++ y) ++ z, x ++ \text{nil} \approx x, \text{nil} ++ y \approx y\}$ can be used.² \triangle

1.3 Overview of Contributions

In order to obtain TRSs that can elegantly model natural numbers, integers, and collection data structures, this dissertation introduces a generalized form of TRSs.

²Notice that rewriting modulo these properties is non-terminating since $\text{reverse}(\text{nil}) \sim_{\mathcal{E}} \text{reverse}(\text{nil} ++ \text{nil}) \rightarrow_{\mathcal{R}} \text{reverse}(\text{nil}) + \text{reverse}(\text{nil}) \sim_{\mathcal{E}} \text{reverse}(\text{nil} ++ \text{nil}) + \text{reverse}(\text{nil}) \rightarrow_{\mathcal{R}} \dots$. The class of rewrite systems considered in this dissertation thus uses a different rewrite relation that uses the same idea that is also used in Maude and Moca. This rewrite relation is formally defined in Chapter 3.

```
type 'a list = private
| Nil
| Element of 'a
| Concat of 'a list * 'a list
begin
  associative
  neutral (Nil)
end

let rec length x =
  match x with
  | Nil -> 0
  | Element n -> 1
  | Concat (k, l) -> length(k) + length(l)

let rec reverse x =
  match x with
  | Nil -> Nil
  | Element n -> Element n
  | Concat (k, l) -> Concat (reverse l, reverse k)
```

Figure 1.2: OCaml program for Example 1.3.

For these TRSs, numbers or integers are *built-in*, thus making basic knowledge about them directly available. Furthermore, this class of TRSs makes it possible to use *collection data structures*.

As discussed above, one of the main questions about TRSs is termination, and most of this dissertation is concerned with methods for showing termination of TRSs with built-in numbers and collection data structures. In addition to full rewriting, termination under strategies is investigated as well since this is often needed to accurately model the semantics of programming languages. This includes the innermost strategy and context-sensitive strategies.

Having built-in numbers often makes reasoning about termination conceptually easier than using a Peano representation (or a representation using non-free con-

structors for integers). Indeed, termination of the `while`-loop from Example 1.1 can easily be established with an intuitive proof using the methods developed in this dissertation, regardless of whether built-in natural numbers or integers are considered. Furthermore, this dissertation presents the first powerful methods for reasoning about termination of TRSs operating on collection data structures.

In addition to termination analysis, this dissertation also contributes to the use of inductive reasoning in the context of showing partial correctness of computer programs that are modeled using TRSs with built-in numbers and collection data structures. Here, an additional interest is to identify a class of equational conjectures where inductive reasoning provides a decision procedure.

The contributions of this dissertation have been fully implemented, thus proving their practicality. The termination techniques have been implemented as part of the award-winning termination tool `AProVE` [84]. The inductive proof procedure and methods for determining whether a given equational conjecture falls into the class of conjectures whose inductive validity has been identified to be decidable have been implemented in the tool `Sail2`.

1.4 Detailed Outline

The first contribution of this dissertation is the definition of a class of conditional rewrite systems with built-in natural numbers or integers. Basic knowledge about those numbers is available in the form of quantifier-free formulas from Presburger arithmetic that are attached to the rewrite rules in the form of constraints. Application of a rewrite rule is then only possible if the constraint becomes true in the natural numbers or integers after being instantiated by the matching substitution.

Example 1.4. Consider the `while`-loop from Example 1.1. It can be translated

Chapter 1. Introduction

into the following rewrite rule:

$$\text{eval}(x, y) \rightarrow \text{eval}(x, y + 1) \llbracket x \geq y \rrbracket$$

Notice that the condition of the `while`-loop is directly translated into a constraint from Presburger arithmetic. Termination of this system can easily be shown using the methods developed in this dissertation by observing that $x - y$ on the left side of the rule is bigger than $x - (y + 1)$ on the right side, where the constraint $x \geq y$ additionally implies that $x - y$ is bounded by 0. \triangle

While the class of rewrite systems with built-in numbers already deserves interest in its own right, it is furthermore extended in order to make it possible to use collection data structures such as sets or multisets, giving rise to *conditional constrained equational rewrite systems (CCERSs)*. This extension is quite natural and does not introduce additional conceptual problems.

Example 1.5. This example shows a quicksort algorithm that takes a finite set and produces a sorted list of its elements. For this, sets are built from the empty set \emptyset using the constructor `ins` to add an element to a set.

$$\begin{aligned} \text{app}(\text{nil}, zs) &\rightarrow zs \\ \text{app}(\text{cons}(x, ys), zs) &\rightarrow \text{cons}(x, \text{app}(ys, zs)) \\ \text{split}(x, \emptyset) &\rightarrow \langle \emptyset, \emptyset \rangle \\ \text{split}(x, zs) \rightarrow^* \langle zl, zh \rangle \mid \text{split}(x, \text{ins}(y, zs)) &\rightarrow \langle \text{ins}(y, zl), zh \rangle \llbracket x > y \rrbracket \\ \text{split}(x, zs) \rightarrow^* \langle zl, zh \rangle \mid \text{split}(x, \text{ins}(y, zs)) &\rightarrow \langle zl, \text{ins}(y, zh) \rangle \llbracket x \not> y \rrbracket \\ \text{qsort}(\emptyset) &\rightarrow \text{nil} \\ \text{split}(x, ys) \rightarrow^* \langle yl, yh \rangle \mid \text{qsort}(\text{ins}(x, ys)) &\rightarrow \text{app}(\text{qsort}(yl), \text{cons}(x, \text{qsort}(yh))) \end{aligned}$$

Here, `split`(x, ys) returns a pair of sets $\langle yl, yh \rangle$ where yl contains all $y \in ys$ such that $x > y$ and yh contains all $y \in ys$ such that $x \not> y$. Intuitively, the condition

$\text{split}(x, ys) \rightarrow^* \langle yl, yh \rangle$ of the second `qsort`-rule means that $\text{split}(x, ys)$ first needs to be rewritten recursively until it matches $\langle yl, yh \rangle$ (thus giving a binding to these variables) before $\text{qsort}(\text{ins}(x, ys))$ may be reduced using that rule. \triangle

CCERSs make it possible to model a wide class of programs in an intuitive and natural way. The main part of this dissertation investigates the automated termination analysis of such systems. Furthermore, inductive reasoning with a restricted class of these systems is investigated.

1.4.1 Termination and Operational Termination

Termination and operational termination of term rewriting has been extensively studied (see, e.g., [173] for a recent survey), but none of the formerly developed methods can directly be applied to the class of conditional rewrite systems considered here due to the built-in numbers and the use of collection data structures.

First, only unconditional CCERSs (i.e., *CERSs*) are considered and automated termination analysis methods for CERSs are developed. For ordinary TRSs, one of the most widely used methods for termination analysis is the dependency pair method [12]. The main idea of the dependency pair method involves showing the absence of infinite chains build from recursive calls. A variety of techniques to this extent have been developed (see, e.g., [88, 95]), and the *dependency pair framework* [86] provides a way for combining these techniques in a flexible manner.

It is shown in this dissertation that the main idea of the dependency pair method (i.e., termination if there are no infinite chains of recursive calls) can be extended to CERSs, giving rise to a dependency pair framework. Furthermore, it is shown that the most important techniques employed in the dependency pair framework for ordinary TRSs can be generalized to CERSs:

Chapter 1. Introduction

1. The dependency graph technique [12], which decomposes a termination problem into several independent termination problems. The newly obtained termination problems can then be handled independently of each other.
2. The subterm technique [95], which can easily handle many termination problems, in particular termination problems obtained from functions defined using primitive recursion.
3. The reduction pair technique [117, 12], which applies well-founded relations in order to simplify a termination problem. The main novelty of this dissertation in this regard is a simple and intuitive way to use polynomial interpretations [119] with negative coefficients in such a way that the constraints from Presburger arithmetic that are attached to the rewrite rules are fully utilized.

In addition to the techniques based on methods initially developed for ordinary TRSs, this dissertation also introduces several techniques that are specifically tailored towards CERSs.

In order to show termination of CCERSs, it is shown in this dissertation that *operational termination* [127, 59] of CCERSs can be reduced to termination of unconditional CERSs by a simple syntactic transformation. Operational termination differs from (regular) termination by also ensuring that evaluation of the conditions of a rewrite rule does not diverge. This property has been investigated for ordinary conditional TRSs and is typically handled using a simulation of the evaluation of the conditions using an unconditional rewrite system [136, 79].

Example 1.6. Continuing Example 1.5, this example illustrates the syntactic transformation from CCERSs into CERSs. The conditional rewrite rule

$$\text{split}(x, ys) \rightarrow^* \langle yl, yh \rangle \mid \text{qsort}(\text{ins}(x, ys)) \rightarrow \text{app}(\text{qsort}(yl), \text{cons}(x, \text{qsort}(yh)))$$

is transformed into the following two (unconditional) rewrite rules:

$$\begin{aligned} \text{qsort}(\text{ins}(x, ys)) &\rightarrow \text{U}(\text{split}(x, ys), x, ys) \\ \text{U}(\langle yl, yh \rangle, x, ys) &\rightarrow \text{app}(\text{qsort}(yl), \text{cons}(x, \text{qsort}(yh))) \end{aligned}$$

Here, U is a fresh function symbol that enforces that $\text{split}(x, ys)$ is reduced to $\langle yl, yh \rangle$ before the reduction of $\text{qsort}(\text{ins}(x, ys))$ produces $\text{app}(\text{qsort}(yl), \text{cons}(x, \text{qsort}(yh)))$. The other conditional rewrite rules are handled similarly. \triangle

1.4.2 Termination under Strategies

In order to model the semantics of programming languages more closely using the term rewriting framework, termination of CERSs under the following reduction strategies is investigated.

Innermost Strategy. In the innermost strategy, a reduction is only allowed if all proper subterms below the position where the reduction takes place are in normal form, i.e., cannot be reduced any further. This strategy corresponds to the eager evaluation strategy used by many functional programming languages such as OCaml or SML. Also, virtually all imperative programming languages employ a call-by-value semantics, which, at the level of TRSs or CERSs, can be modelled using the innermost strategy.

For standard TRSs, termination under the innermost strategy has been considered in the dependency pair framework [12, 86, 88]. It is well-known that there exist TRSs which are terminating using the innermost strategy, but that are not terminating for full rewriting. But even for TRSs that are terminating for both innermost rewriting and full rewriting, it is often easier to establish this for the innermost case. In particular, this is true for automated methods such as the ones considered in this dissertation.

Example 1.7. The following ordinary TRS is due to Toyama [165]:

$$\begin{aligned} f(\mathbf{a}, \mathbf{b}, z) &\rightarrow f(z, z, z) \\ g(x, y) &\rightarrow x \\ g(x, y) &\rightarrow y \end{aligned}$$

Rewriting with this TRS is not terminating because of the following cyclic reduction: $f(\mathbf{a}, \mathbf{b}, g(\mathbf{a}, \mathbf{b})) \rightarrow_{\mathcal{R}} f(g(\mathbf{a}, \mathbf{b}), g(\mathbf{a}, \mathbf{b}), g(\mathbf{a}, \mathbf{b})) \rightarrow_{\mathcal{R}} f(\mathbf{a}, g(\mathbf{a}, \mathbf{b}), g(\mathbf{a}, \mathbf{b})) \rightarrow_{\mathcal{R}} f(\mathbf{a}, \mathbf{b}, g(\mathbf{a}, \mathbf{b}))$. Notice that the first step in this reduction is not allowed by the innermost strategy since $g(\mathbf{a}, \mathbf{b})$ is not in normal form. Indeed, this TRS is terminating using the innermost strategy, and termination tools based on the dependency pair framework can determine this automatically. \triangle

Given this motivation, it is investigated how the methods discussed in Section 1.4.1 can be adapted to show termination of rewriting with CERSs using the innermost strategy.

Context-Sensitive Strategies. Context-sensitive rewriting [121, 123] has been introduced as a flexible paradigm that provides a bridge between the abstract world of term rewriting and the more applied setting of declarative specification and programming languages such as *Maude* [43] (see [122] for the close relationship between context-sensitive rewriting and *Maude*'s `strat`-annotations). In context-sensitive rewriting, a replacement map specifies which arguments of a function symbol may be reduced and which arguments are “frozen”.

Example 1.8. The *Maude* module in Figure 1.3 defines a function `from` that lazily generates the infinite list of integers bigger than its argument. Furthermore, a function `take` that extracts a finite prefix of a lazy list is defined. The following rewrite rules are easily obtained from the *Maude* module:

```

fmod LAZY-LISTS is
  protecting INT .

  sorts List LazyList .

  op nil : -> List [ ctor ] .
  op cons : Int List -> List [ ctor ] .
  op lazycons : Int LazyList -> LazyList [ ctor strat (1) ] .
  op from : Int -> LazyList .
  op take : Int LazyList -> List .

  var M N : Int .
  var LL : LazyList .

  eq from(N) = lazycons(N, from(N + 1)) .
  ceq take(N, LL) = nil if N <= 0 .
  ceq take(N, lazycons(M, LL)) = cons(M, take(N - 1, LL)) if N > 0 .
endfm

```

Figure 1.3: Maude module for Example 1.8.

$$\begin{aligned}
\text{from}(n) &\rightarrow \text{lazycons}(n, \text{from}(n + 1)) \\
\text{take}(n, ll) &\rightarrow \text{nil} \llbracket n \leq 0 \rrbracket \\
\text{take}(n, \text{lazycons}(m, ll)) &\rightarrow \text{cons}(m, \text{take}(n - 1, ll)) \llbracket n > 0 \rrbracket
\end{aligned}$$

Notice that regular rewriting with these rules is not terminating since $\text{from}(0) \rightarrow_{\mathcal{R}} \text{lazycons}(0, \text{from}(0 + 1)) \rightarrow_{\mathcal{R}} \text{lazycons}(0, \text{lazycons}(0 + 1, \text{from}(0 + 1 + 1))) \rightarrow_{\mathcal{R}} \dots$

The reason for this is that the lazy behavior employed by **Maude** is not accurately modeled. The `strat`-annotation for the constructor `lazycons` specifies that the second argument of `lazycons` is frozen, i.e., reductions may not be applied in this argument. This behavior can be modeled using context-sensitive rewriting with a replacement map where $\mu(\text{lazycons}) = \{1\}$ since this disallows reductions in the second argument of `lazycons` (see Chapter 11 for details). \triangle

Context-sensitive rewriting is relevant for (eager) functional and imperative pro-

gramming languages as well. For example, context-sensitive rewriting is useful for modeling the following:

1. The non-strict semantics of `if-then-else`, where the `then`- or the `else`-part is only evaluated *after* it has been established whether the condition of the `if`-statement evaluates to true or false. This non-strict behavior can easily be modelled by a ternary function symbol `if` whose second and third arguments are frozen:

$$\text{if}(\text{true}, x, y) \rightarrow x$$

$$\text{if}(\text{false}, x, y) \rightarrow y$$

2. The short-cut semantics of the Boolean connectives “and” and “or”, where conjuncts or disjuncts are evaluated from left to right and only if they are needed. This can easily be modelled using binary function symbols `and` and `or` whose second argument is frozen:

$$\text{and}(\text{true}, y) \rightarrow y$$

$$\text{and}(\text{false}, y) \rightarrow \text{false}$$

$$\text{or}(\text{true}, y) \rightarrow \text{true}$$

$$\text{or}(\text{false}, y) \rightarrow y$$

Furthermore, context-sensitive rewriting makes it possible to model lazy evaluation as used in functional programming languages such as `Haskell` (for more on the relationship between lazy evaluation and context-sensitive rewriting, see [124]).

Dependency pair methods for proving context-sensitive termination for ordinary rewriting have recently been developed [2, 1]. In this dissertation, the methods discussed in Section 1.4.1 are adapted in order to show context-sensitive termination of CERSs. Furthermore, it is shown that the transformation from CCERSs to CERSs can be applied in combination with context-sensitive rewriting strategies.

1.4.3 Inductive Reasoning

Inductive reasoning has been widely used in the context of showing partial correctness of programs, with the majority of research in the area being done in the 1980's and 1990's (see [170, 39, 46] for surveys on inductive reasoning). One of the main drawbacks of inductive reasoning as identified by that research is the need for user interaction in the proof process, i.e., a complete automation is nearly impossible. Due to this, inductive reasoning cannot be used in a push-button mode and is typically only applicable by trained experts who can assist the inductive reasoning tool.

More recently, research on identifying conditions under which inductive reasoning provides a decision procedure has been initiated [111, 81, 82, 104].³ If the conditions identified in this work are satisfied, then an inductive reasoning tool can be used in a push-button mode without any user interaction. This previous work, however, imposes the following strong restrictions:

1. The function definitions in the term rewrite systems have to be of a very simple shape. In particular, a function may only make recursive calls to itself and not to any other auxiliary function. Furthermore, this restriction also disallows mutually recursive function definitions.
2. The equational conjectures whose inductive validity is to be determined need to be *linear*, i.e., each variable may only occur once in each side of the conjecture.

The contributions of this dissertation for inductive reasoning based on the term rewriting framework are two-fold:

1. An inductive proof method for a restricted class of CERSs is developed. This proof method combines inductive reasoning with a decision procedure, thus

³The dual problem of determining conditions under which a proof attempt is guaranteed to fail has been investigated in [158, 159, 160]. Furthermore, methods to automatically fix failed proof attempts are presented in [158, 160].

Chapter 1. Introduction

obtaining a powerful proof method that can be used in a push-button mode and does not require any user intervention.

2. The class of conjectures where inductive reasoning provides a decision procedure is increased significantly compared to [111, 81, 82, 104]. In particular, the two shortcomings discussed above are removed.

Example 1.9. For two lists built using `nil` and `cons`, `prefix(xs, ys)` computes the longest prefix p of xs such that all elements of p occur in ys in the same order as in p (but not necessarily consecutively).

$$\begin{aligned}\text{prefix}(\text{nil}, ys) &\rightarrow \text{nil} \\ \text{prefix}(\text{cons}(x, xs), \text{nil}) &\rightarrow \text{nil} \\ \text{prefix}(\text{cons}(x, xs), \text{cons}(y, ys)) &\rightarrow \text{cons}(x, \text{prefix}(xs, ys)) \llbracket x \simeq y \rrbracket \\ \text{prefix}(\text{cons}(x, xs), \text{cons}(y, ys)) &\rightarrow \text{prefix}(\text{cons}(x, xs), ys) \llbracket x \not\approx y \rrbracket\end{aligned}$$

Notice the use of (dis-)equality constraints in the definition of `prefix`, which requires an inductive proof method that can utilize these constraints.

The inductive validity of the (false) conjecture `prefix(xs, ys) \equiv xs` can be seen to be decidable by (a slight adaptation of) the conditions developed in [111, 81, 82, 104]. Using [111, 81, 82, 104], it is not known whether the inductive validity of the (true) conjecture `prefix(xs, xs) \equiv xs` is decidable since this conjecture is not linear. Using the new conditions developed in this dissertation, it can be established a priori that the inductive validity of this conjecture is decidable. \triangle

1.5 Structure of the Dissertation

Chapter 1. The research topic of this dissertation is introduced and motivated in this chapter. The significance of the topic is illustrated using several example pro-

Chapter 1. Introduction

grams written in OCaml and Maude. Related work is discussed and the contributions of this dissertation are summarized.

Chapter 2. Since (many-sorted) term rewrite systems are used as a convenient model of computer programs in this dissertation, the second chapter recalls the relevant concepts and definitions.

Chapter 3. As shown in Section 1.2, ordinary TRSs have (at least) two severe shortcomings:

1. Knowledge about natural numbers or integers is not available, thus making reasoning about the computer programs modeled by TRSs cumbersome and complicated.
2. Collection data structures such as sets or multisets are not suitably supported by ordinary TRSs since their specification using equational attributes typically results in a non-terminating rewrite relation.

This chapter introduces *constrained equational rewrite systems (CERSs)*, a new class of term rewrite systems that addresses both of these shortcomings.

Chapter 4. In order to show the versatility of the class of term rewrite systems introduced in Chapter 3, this brief chapter presents a translation from a simple class of imperative programs into CERSs. This translation is sound for termination proving, i.e., the methods to prove termination of CERSs that are developed in this dissertation can then be applied in order to show termination of imperative programs.

Chapter 5. This chapter starts the investigation of methods to (automatically) prove termination and innermost termination of CERSs. To this extent, the modular

Chapter 1. Introduction

dependency pair framework [12, 86] is adapted to CERSs and it is shown that the absence of infinite chains of recursive calls is equivalent to termination. This is true for both unrestricted rewriting and rewriting with the innermost strategy.

Chapter 6. Here, first techniques that can be applied within the dependency pair framework for CERS are developed. These techniques include:

- An adaptation of the dependency graph technique [12], which decomposes a termination problem into several independent termination problems. The newly obtained termination problems can then be handled independently of each other.
- An adaptation of the subterm technique [95], which can easily handle many termination problems, in particular termination problems obtained from functions defined using primitive recursion.
- A new technique, called *dependency pair narrowing*, which combines a recursive call from f to g with a recursive call from g to h , resulting in a recursive call from f to h .

This chapter also presents several additional techniques.

Chapter 7. Many commonly used techniques for showing termination are based on well-founded relations. In particular, one of the most important techniques used in the dependency pair framework is based on *reduction pairs* [117, 12]. This chapter first shows that these (ordinary) reduction pairs can be applied to CERSs as well. Then, specialized notions of reduction pairs that take advantage of the built-in natural numbers or integers are developed. These new kinds of reduction pairs are often crucial for a successful termination proof and provide a simple and intuitive way to use polynomial interpretations [119] with negative coefficients.

Chapter 1. Introduction

Chapter 8. When using the techniques based on reduction pairs from Chapter 7, it becomes necessary to satisfy certain conditions that are derived from *all* rewrite rules of the CERS whose termination is to be shown. These conditions are often hard or impossible to satisfy, and this chapter shows that it suffices to satisfy these conditions only for a syntactically determined subset of all rewrite rules. This makes it easier to satisfy these conditions and may result in a successful termination proof which is not possible without the refinement presented in this chapter.

Chapter 9. After presenting various techniques for proving termination in Chapters 6–8, this chapter discusses methods to implement two of these techniques, namely the dependency graph technique and the automatic generation of the new kind of reduction pairs based on polynomial interpretations with negative coefficients. While an implementation of the remaining techniques is relatively straightforward, these two techniques require the development of dedicated methods.

Chapter 10. This chapter extends CERSs to *conditional CERSs (CCERSs)*. In conditional rewriting, the rewrite rules are equipped with conditions that need to be established by recursively rewriting them before a rewrite rule may be applied. Conditional rewriting needs a more complex notion of termination, *operational termination*, that also ensures that the evaluation of the conditions is terminating. It is shown that operational termination of a CCERS can be reduced to (regular) termination of an unconditional CERS. Thus, the techniques from Chapters 5–8 become applicable for proving operational termination of CCERSs and it is not necessary to develop new methods in order to reason about operational termination of CCERSs.

Chapter 11. After considering unrestricted and innermost rewriting with CERSs in Chapters 3–8, this chapter considers context-sensitive rewriting strategies for CERSs. In context-sensitive rewriting, a reduction may only take place in certain

Chapter 1. Introduction

argument positions of the function symbols, resulting in a fine-grained control of the evaluation order. This makes it very challenging to reason about termination of context-sensitive rewriting with CERSs. In this chapter, a dependency pair framework for context-sensitive CERSs is developed and it is shown that the main idea of the dependency pair method can be extended to context-sensitive CERSs.

Chapter 12. After developing a dependency pair framework for context-sensitive CERSs in the previous chapter, this chapter adapts the most important techniques from Chapters 6–8 to the context-sensitive case. This includes the dependency graph technique, the subterm technique, and the refined version of the technique based on reduction pairs from Chapter 8.

Chapter 13. This chapter develops an inductive proof method for CERSs which can be used to verify properties of the functions specified by a CERS. The inductive proof method couples inductive reasoning with a decision procedure, thus resulting in a powerful and completely automatic proof method.

Chapter 14. It is well-known that inductive theorem proving often requires substantial user interaction in order to obtain a successful proof. This is undesirable in many cases since this kind of interaction typically requires a trained expert. It is thus important to identify a class of conjectures where the inductive proof method is known to succeed without any user interaction. This chapter develops several sufficient conditions for this, substantially generalizing previous work on identifying conjectures with this property.

Chapter 15. The methods presented in this dissertation have been fully implemented as part of the termination tool **AProVE** and in the inductive reasoning tool

Sail2, respectively. Here, an empirical evaluation of these implementations is summarized. The detailed empirical evaluation is available online and in Appendix B. Additionally, some ideas for future work are presented.

Appendix A. In order to not disturb the presentation in the body of this dissertation, all proofs are collected in this appendix.

Appendix B. This appendix contains the detailed empirical evaluation of the implementations in AProVE and Sail2.

1.6 Published Contributions and New Contributions

Parts of the research presented in this dissertation have already been published in the proceedings of international conferences and workshops:

1. Term rewrite systems with collection data structures and the dependency pair method for such systems have been presented at the *21st International Conference on Automated Deduction (CADE 2007)* [66].
2. At the *19th International Conference on Rewriting Techniques and Applications (RTA 2008)*, this work was extended to support built-in natural numbers [67].
3. Operational termination of conditional CERSs with built-in numbers and its reduction to termination of unconditional CERSs has been presented at the *8th International Workshop on Reduction Strategies in Rewriting and Programming (WRS 2008)* [68].
4. Parts of the results on termination of context-sensitive rewriting with CERSs

Chapter 1. Introduction

have been presented at the *18th International Workshop on Functional and (Constraint) Logic Programming (WFLP 2009)* [70].

5. The translation from imperative programs into CERSs as presented in Chapter 4, together with a simplified version of the termination techniques presented in this dissertation, has been presented at the *22nd International Conference on Automated Deduction (CADE 2009)* [69].
6. First results on relaxing the conditions imposed by [111, 81, 82, 104] for identifying equational conjectures whose inductive validity is decidable have been presented at the *13th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2006)* [65].

Even though some results have been published before, this dissertation nonetheless contains several new results:

1. The results from the publications 1–3 given above have been generalized and now support built-in natural numbers or built-in integers.
2. This dissertation presents termination techniques such as *dependency pair narrowing* that are not yet discussed in [66, 67, 68] Furthermore, some of the techniques from [66, 67, 68] have been improved by generalizing and/or simplifying them.
3. The previous publications [66, 67, 68] do not contain empirical evaluations since an implementation was not yet available. In contrast to this, the work presented in this dissertation has been fully implemented, and an empirical evaluation shows the efficiency of the methods.
4. The results on identifying equational conjectures whose inductive validity is decidable have been generalized substantially compared to [65]. Furthermore, this work has been reformulated using CERSs instead of ordinary TRSs. In particular, an inductive proof method for CERSs has been developed.

1.7 Related Work

The integration of natural numbers, integers, or other predefined algebras into the term rewriting framework or the algebraic specification framework has been considered before. The work of Vorobyov [168] and Ayala-Rincón [14, 15] integrates integers, but imposes the strong restriction that no user-defined function can have the set of integers as its resulting sort. This restriction has been relaxed in the work of Antimirov & Degtyarev [5, 6] and Avenhaus & Becker [13], which furthermore make it possible to integrate more general predefined algebras in an abstract setting. This dissertation shares the possibility of having user-defined functions with the set of integers as the resulting sort with [5, 6, 13].

While this dissertation shares some of the general ideas with [168, 14, 15, 5, 6, 13], it widely differs in its intention. All of the previous work is mostly concerned with semantical issues of the integration of integers. With the exception of [13], termination of such rewrite system is not considered, and [13] only provides a very weak method for proving termination that is based on lexicographic path orders [103]. It is well-known that lexicographic path orders are not powerful enough to show termination of many natural and simple examples.

More recently, and after most of the work on proving termination presented in this dissertation had been finished, the integration of integers into the term rewriting framework for the purpose of proving termination has been considered in [75].⁴ In general, the work presented in this dissertation and the work presented in [75] are incomparable. On the one hand, [75] provides a more complete integration of integers since multiplication and division are supported, whereas this dissertation is restricted to linear arithmetic. On the other hand, [75] does not support collec-

⁴The main contribution of the author of this dissertation to [75] is the automatic generation of polynomial interpretations with negative coefficients. Indeed, the method presented in [75] is partially based on the method developed in Chapter 9 of this dissertation.

Chapter 1. Introduction

tion data structures or context-sensitive rewriting and does not consider inductive theorem proving with built-in integers. An empirical comparison of the termination methods presented in this dissertation and the methods from [75] on examples where both approaches are applicable shows that an implementation of the methods presented in this dissertation is much faster on these examples than an implementation of the methods from [75], with both methods having nearly the same power.

The integration of linear arithmetic into generic first-order theorem proving methods has also received increased interest in recent years [116, 151, 25, 4].

In contrast to integers, the integration of collection data structures into term rewriting has not received much attention. Rewriting modulo a set of equations [120] and extended rewriting with a set of equations [139] make it possible to model collection data structures, but this integration is unsatisfactory since the resulting rewrite relation is typically non-terminating (recall Footnote 2). The recent work of [61] was published after the work presented in this dissertation had been finished. In [61], rewriting with collection data structures (or more general rewriting modulo certain equations) is transformed to *AC*-rewriting with an enlarged rewrite system that can often be computed automatically. Then, methods for showing termination of *AC*-rewriting such as [108, 150, 129, 118, 130, 31] can be applied. Since an implementation of [61] is not available, an empirical comparison of the methods developed in this dissertation and the methods presented in [61] is not possible.

Further related work is also discussed throughout Chapters 2–15.

Chapter 2

Many-Sorted Term Rewriting

This chapter discusses basic concepts of term rewriting that are used in the remainder of this dissertation. Since it is impossible to provide a comprehensive treatment of these concepts, this chapter only fixes notation and terminology for future use. The notation and terminology is mostly consistent with [17], which provides an in-depth treatment of term rewriting.

2.1 Terms and Substitutions

On a purely syntactic level, this section introduces terms and substitutions in the many-sorted setting. For more information on this, see [142, 17, 56].

Terms are built from function symbols and variables. The available function symbols and their respective sort declarations are collected in a *signature*.

Definition 2.1 (Signatures). *A signature $\mathcal{F} = (S, F)$ consists of:*

1. *A finite non-empty set S of sorts.*
2. *A finite non-empty set F of function symbols, where each $f \in F$ is associated*

Chapter 2. Many-Sorted Term Rewriting

with a sort declaration of the form $s_1 \times \dots \times s_n \rightarrow s$ such that $n \geq 0$ and $s_1, \dots, s_n, s \in S$. The resulting sort of f is s , and $\text{arity}(f) = n$ denotes the arity of f . If $\text{arity}(f) = n$, then f is said to be n -ary.

Since the set S of sorts can be derived from the sort declarations, a signature $\mathcal{F} = (S, F)$ is often identified with the set F of function symbols.

Example 2.2. In order to model finite sets of natural numbers, a signature with the sorts `nat` and `set` and the following function symbols can be used:

$$\begin{aligned} \mathcal{O} & : \rightarrow \text{nat} \\ \mathbf{s} & : \text{nat} \rightarrow \text{nat} \\ \emptyset & : \rightarrow \text{set} \\ \{\cdot\} & : \text{nat} \rightarrow \text{set} \\ \cup & : \text{set} \times \text{set} \rightarrow \text{set} \end{aligned}$$

Here, \mathcal{O} and \mathbf{s} are used to model natural numbers using a Peano representation and \emptyset , $\{\cdot\}$, and \cup are used to model finite sets of natural numbers, where $\{\cdot\}$ is used to construct a singleton set. △

In the following, it is assumed that F contains at least one constant symbol c_s with sort declaration $c_s : \rightarrow s$ for each $s \in S$. Using a signature and a disjoint set of variables, *terms* can be built. Furthermore, each term s is associated with a sort.

Definition 2.3 (Terms). *Let $\mathcal{F} = (S, F)$ be a signature and let \mathcal{V} be a countably infinite set of variables with $\mathcal{V} \cap F = \emptyset$ where each $v \in \mathcal{V}$ is associated with a sort $\text{sort}(v) \in S$. The set $\mathcal{T}(\mathcal{F}, \mathcal{V})$ of terms over \mathcal{F} and \mathcal{V} and the extension of the mapping sort to $\mathcal{T}(\mathcal{F}, \mathcal{V})$ are inductively defined by:*

1. $\mathcal{V} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$
2. If $f \in F$ with sort declaration $s_1 \times \dots \times s_n \rightarrow s$ and $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ with $\text{sort}(t_i) = s_i$, then $f(t_1, \dots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and $\text{sort}(f(t_1, \dots, t_n)) = s$.

Terms that do not contain variables are called *ground*. For a signature \mathcal{F} , the set of all ground terms over \mathcal{F} is denoted by $\mathcal{T}(\mathcal{F})$.

Example 2.4. For the signature from Example 3.13, $\{x\} \cup \emptyset$ is a term and $\{\mathcal{O}\} \cup \emptyset$ is a ground term. △

A tuple of terms s_1, \dots, s_n for some $n \geq 0$ is denoted by s^* , and notations for terms are extended to tuples of terms component-wise.

In order to access the *subterms* of a given term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, it is convenient to introduce the concept of *positions* in a term. This also makes it possible to define *replacements*.

Definition 2.5 (Positions, Sizes, Subterms, and Replacements).

1. The set of positions $\mathcal{Pos}(t)$ of a term t is a set of strings over \mathbb{N}^+ that is inductively defined by:

(a) If $t \in \mathcal{V}$, then $\mathcal{Pos}(t) = \{\Lambda\}$, where Λ denotes the empty string.

(b) If $t = f(t_1, \dots, t_n)$, then $\mathcal{Pos}(t) = \{\Lambda\} \cup \bigcup_{i=1}^n \{i.p \mid p \in \mathcal{Pos}(t_i)\}$.

The position Λ is called the *root position*. A position p is *above* a position q if $p \leq q$, where \leq denotes the prefix order on strings. If $p \leq q$, then q is also said to be *below* p . If neither $p \leq q$ nor $q \leq p$, then p and q are *independent*, written $p \parallel q$.

2. The size $|t|$ of a term t is defined as the cardinality of $\mathcal{Pos}(t)$.

3. For $p \in \mathcal{Pos}(t)$, the symbol of t at position p , denoted $t(p)$, is defined by:

(a) If $t \in \mathcal{V}$, then $t(\Lambda) = t$.

(b) $f(t_1, \dots, t_n)(\Lambda) = f$

(c) $f(t_1, \dots, t_n)(i.q) = t_i(q)$

The symbol $t(\Lambda)$ is called the *root symbol* of t . This is also written as $\text{root}(t)$.

Chapter 2. Many-Sorted Term Rewriting

4. For $p \in \mathcal{P}os(t)$, the subterm of t at position p , denoted $t|_p$, is defined by:

$$(a) \ t|_{\Lambda} = t$$

$$(b) \ f(t_1, \dots, t_n)|_{i.q} = t_i|_q$$

If $t|_p = s$ for some position $p \in \mathcal{P}os(t)$, then s is a subterm of t , written $t \supseteq s$.

If additionally $p \neq \Lambda$, then s is a strict subterm of t , written $t \triangleright s$.

5. For $p \in \mathcal{P}os(t)$ and a term s with $\text{sort}(s) = \text{sort}(t|_p)$, the term obtained from t by replacing the subterm at position p by s is denoted $t[s]_p$ and defined by:

$$(a) \ t[s]_{\Lambda} = s$$

$$(b) \ f(t_1, \dots, t_i, \dots, t_n)[s]_{i.q} = f(t_1, \dots, t_i[s]_q, \dots, t_n)$$

The set of variables occurring in a term t is denoted by $\mathcal{V}(t)$. Similarly, the set of function symbols occurring in t is denoted by $\mathcal{F}(t)$. This naturally extends to pairs of terms, sets of terms, etc. A term t is *linear* if each $v \in \mathcal{V}(t)$ occurs only once in t .

Example 2.6. Continuing Example 2.4,

$$1. \ \mathcal{P}os(\{x\} \cup \emptyset) = \{\Lambda, 1, 1.1, 2\}$$

$$2. \ |\{x\} \cup \emptyset| = 4$$

$$3. \ (\{x\} \cup \emptyset)(\Lambda) = \cup, (\{x\} \cup \emptyset)(1) = \{\cdot\}, (\{x\} \cup \emptyset)(1.1) = x, \text{ and } (\{x\} \cup \emptyset)(2) = \emptyset$$

$$4. \ (\{x\} \cup \emptyset)|_{\Lambda} = \{x\} \cup \emptyset, (\{x\} \cup \emptyset)|_1 = \{x\}, (\{x\} \cup \emptyset)_{1.1} = x, \text{ and } (\{x\} \cup \emptyset)_2 = \emptyset$$

$$5. \ (\{x\} \cup \emptyset)[s(\mathcal{O})]_{1.1} = \{s(\mathcal{O})\} \cup \emptyset \text{ and } (\{x\} \cup \emptyset)[y \cup z]_2 = \{x\} \cup (y \cup z)$$

$$6. \ \mathcal{V}(\{x\} \cup \emptyset) = \{x\}$$

$$7. \ \mathcal{F}(\{x\} \cup \emptyset) = \{\{\cdot\}, \cup, \emptyset\}$$

The term $\{x\} \cup \emptyset$ is linear. △

A *context* over \mathcal{F} is a term $C \in \mathcal{T}(\mathcal{F} \cup \bigcup_{s \in S} \{\square_s\}, \mathcal{V})$. Here, $\square_s : \rightarrow s$ is a fresh constant symbol of sort s , called a *hole*. If the sort of the holes can be derived or

is not important, then \square will be used to stand for any of the \square_s . If C is a context with n holes and t_1, \dots, t_n are terms of the appropriate sorts, then $C[t_1, \dots, t_n]$ is the result of replacing the occurrences of holes by t_1, \dots, t_n “from left to right”.

Example 2.7. The term $C = \square \cup (\square \cup z)$ is a context. $C[x, y] = x \cup (y \cup z)$. \triangle

Variables in a term may be replaced by *substitutions*.

Definition 2.8 (Substitutions). A substitution is a function $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $\text{sort}(x) = \text{sort}(\sigma(x))$ for all $x \in \mathcal{V}$. If there are only finitely many variables x with $\sigma(x) \neq x$, then σ is written as $\{x \mapsto \sigma(x) \mid \sigma(x) \neq x\}$. The same notation is used if there are only finitely many variables that are “of interest”. A substitution σ is ground iff $\sigma(x) \in \mathcal{T}(\mathcal{F})$ for all variables x .

Application of a substitution is usually written using postfix notation. Each substitution σ uniquely extends to a homomorphism $\sigma : \mathcal{T}(\mathcal{F}, \mathcal{V}) \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{V})$ by letting $f(t_1, \dots, t_n)\sigma = f(t_1\sigma, \dots, t_n\sigma)$.

Example 2.9. The mapping $\sigma = \{x \mapsto y, y \mapsto x\}$ is a substitution. $(x \cup y)\sigma = y \cup x$. Notice that the substitution is applied *simultaneously* to all variables. \triangle

Substitutions are the key ingredient to define rewrite relations in Sections 2.3 and 2.4. For this, the following concept is used.

Definition 2.10 (Matching). A term s matches a term t if there exists a substitution σ such that $s\sigma = t$.

Example 2.11. The term $x \cup y$ matches the term $\{\mathcal{O}\} \cup (\{\mathbf{s}(\mathcal{O})\} \cup \{\mathbf{s}(\mathbf{s}(\mathcal{O}))\})$ since $(x \cup y)\sigma = \{\mathcal{O}\} \cup (\{\mathbf{s}(\mathcal{O})\} \cup \{\mathbf{s}(\mathbf{s}(\mathcal{O}))\})$ using the substitution $\sigma = \{x \mapsto \{\mathcal{O}\}, y \mapsto \{\mathbf{s}(\mathcal{O})\} \cup \{\mathbf{s}(\mathbf{s}(\mathcal{O}))\}\}$. \triangle

Notice that the substitution is only applied to one of the terms in matching. If the substitution is applied to both terms, *unification* is obtained.

Definition 2.12 (Unification). *Two terms s and t are unifiable iff there exists a substitution σ such that $s\sigma = t\sigma$. In this case, σ is a unifier of s and t . A unifier σ is a most general unifier of s and t iff for any unifier σ' of s and t there exists a substitution δ such that $\sigma' = \sigma\delta$.¹*

Example 2.13. The terms $x \cup (\{s(\mathcal{O})\} \cup z)$ and $(\{l\} \cup z) \cup (y \cup z)$ are unifiable and $\sigma = \{x \mapsto \{l\} \cup z, y \mapsto \{s(\mathcal{O})\}\}$ is a unifier. △

As is well-known, a most general unifier of s and t is unique up to variable-renamings, i.e., if σ and σ' are two most general unifiers, then $\sigma' = \sigma\delta$ for a variable-renaming δ , i.e., a bijective substitution $\delta : \mathcal{V} \rightarrow \mathcal{V}$. This property justifies the notation $\text{mgu}(s, t)$ for any most general unifier of s and t . A most general unifier $\text{mgu}(s, t)$ (if it exists) is always computable, see, e.g., [17].

2.2 Relations

In this section, some notation for relations is fixed. For arbitrary sets A, B and C and arbitrary binary relations $R \subseteq A \times B$ and $S \subseteq B \times C$, the *composition* of R and S is defined by:

$$R \circ S := \{(x, z) \in A \times C \mid \exists y \in B. (x, y) \in R \wedge (y, z) \in S\}$$

It is of course also possible and useful to consider the composition of a relation $\rightarrow \subseteq A \times A$ with itself, and the process of composition may be iterated.

Definition 2.14 (Closures of Relations, Well-Foundedness). *Let $\rightarrow \subseteq A \times A$ be a binary relation. Then the following notation is introduced:*

$$\begin{array}{ll} \rightarrow^0 & := \{(x, x) \mid x \in A\} & \text{identity} \\ \rightarrow^n & := \rightarrow^{n-1} \circ \rightarrow & n\text{-fold composition, } n \geq 1 \end{array}$$

¹Here, the composition of two substitutions is defined by $(\sigma\delta)(x) = \delta(\sigma(x))$.

| | | |
|---------------------|---|--|
| \rightarrow^+ | $:= \bigcup_{i \geq 1} \rightarrow^i$ | transitive closure |
| \rightarrow^* | $:= \rightarrow^+ \cup \rightarrow^0$ | reflexive-transitive closure |
| $\rightarrow^=$ | $:= \rightarrow \cup \rightarrow^0$ | reflexive closure |
| \rightarrow^{-1} | $:= \{(y, x) \mid x \rightarrow y\}$ | inverse |
| \leftarrow | $:= \rightarrow^{-1}$ | inverse |
| \leftrightarrow | $:= \rightarrow \cup \leftarrow$ | symmetric closure |
| \leftrightarrow^* | $:= (\rightarrow \cup \leftarrow)^*$ | reflexive-transitive-symmetric closure |
| $\rightarrow^!$ | $:= \{(x, y) \mid x \rightarrow^* y \wedge \neg \exists z. y \rightarrow z\}$ | normalization |

For any $a \in A$, the expression $a \downarrow$ denotes any normal form of a , i.e., any element with $a \rightarrow^! a \downarrow$. Furthermore, \rightarrow is well-founded (terminating) if there is no infinite sequence $a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow \dots$.

2.3 Equations and Rewrite Rules

The basic idea of rewriting is to replace a term by another term that is “equal” to it. This idea is made formal by introducing *equations*.

Definition 2.15 (Equations). *An equation is a pair $(u, v) \in \mathcal{T}(\mathcal{F}, \mathcal{V}) \times \mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $\text{sort}(u) = \text{sort}(v)$. Equations are usually written as $u \approx v$.*

Example 2.16. Continuing Example 2.2, the following equations specify properties that are expected to hold for sets:

$$\begin{array}{ll}
 (1) & x \cup y \approx y \cup x \\
 (2) & x \cup (y \cup z) \approx (x \cup y) \cup z \\
 (3) & x \cup x \approx x \\
 (4) & x \cup \emptyset \approx x
 \end{array}$$

Thus, set union is specified to be associative, commutative, and idempotent. Furthermore, the empty set acts as a unit element. △

Chapter 2. Many-Sorted Term Rewriting

A set \mathcal{E} of equations induces a *rewrite relation* on terms. For this, an equation $u \approx v$ may be applied to a term s if its left-hand side u matches a subterm of s .

Definition 2.17 (Rewrite Relations). *Let \mathcal{E} be a set of equations. Then $s \rightarrow_{\mathcal{E}} t$ iff there exist an equation $u \approx v \in \mathcal{E}$, a position $p \in \mathcal{P}os(s)$, and a substitution σ such that*

1. $s|_p = u\sigma$, and
2. $t = s[v\sigma]_p$.

Example 2.18. Using the equations from Example 5.2, $x \cup (\emptyset \cup y) \rightarrow_{\mathcal{E}} (x \cup \emptyset) \cup y \rightarrow_{\mathcal{E}} x \cup y \rightarrow_{\mathcal{E}} y \cup x$. Here, the first step uses equation (2), the second step uses equation (4), and the third step uses equation (1). \triangle

Notice that the condition $\text{sort}(u) = \text{sort}(v)$ that is imposed on the equation $u \approx v$ guarantees that this replacement is well-defined. The main interest is usually not in $\rightarrow_{\mathcal{E}}$, but in its symmetric and reflexive-transitive-symmetric closures, which are denoted by $\vdash_{\mathcal{E}}$ and $\sim_{\mathcal{E}}$, respectively. Since $\sim_{\mathcal{E}}$ is a congruence relation, it defines a partition of $\mathcal{T}(\mathcal{F}, \mathcal{V})$ into equivalence classes. The members of $\mathcal{T}(\mathcal{F}, \mathcal{V})/\sim_{\mathcal{E}}$ are called *\mathcal{E} -equivalence classes*.

For two terms s and t , writing $s \xrightarrow{>\Lambda}_{\mathcal{E}} t$ means that $s = f(s^*)$ and $t = f(t^*)$ such that $s^* \rightarrow_{\mathcal{E}} t^*$, i.e., the $\rightarrow_{\mathcal{E}}$ -step is applied below the root. This notation extends to $\vdash_{\mathcal{E}}$ and $\sim_{\mathcal{E}}$ (and other relations introduced later) in the obvious way.

The following syntactic properties of equations are used in the remainder of this dissertation.

Definition 2.19 (Properties of Equations). *Let $u \approx v$ be an equation.*

1. $u \approx v$ is collapse-free if neither u nor v is a variable.
2. $u \approx v$ is regular if $\mathcal{V}(u) = \mathcal{V}(v)$.

Chapter 2. Many-Sorted Term Rewriting

3. $u \approx v$ has identical unique variables (is i.u.v.) if it is regular and both u and v are linear.
4. $u \approx v$ is size-preserving if it is i.u.v. and $|u| = |v|$.

A set \mathcal{E} of equations is said to have one of these properties if all equations $u \approx v \in \mathcal{E}$ satisfy that property.

Example 2.20. In Example 2.16, all equations are regular but only (1) and (2) are collapse-free. (1), (2), and (4) are i.u.v. and (1) and (2) are size-preserving. \triangle

If \mathcal{E} is i.u.v., the following technical results about $\sim_{\mathcal{E}}$ can be obtained. Although these results are simple, they have not appeared in the literature before. The first result intuitively states that subterms with a root symbol that is not in $\mathcal{F}(\mathcal{E})$ are preserved (up to $\overset{\Delta}{\sim}_{\mathcal{E}}$) within terms from the same \mathcal{E} -equivalence class.

Lemma 2.21. *Let \mathcal{E} be an i.u.v. set of equations and let $C[f(s^*)] \sim_{\mathcal{E}} t$ for some context C , some term $f(s^*)$ with $f \notin \mathcal{F}(\mathcal{E})$, and some term t . Then $t = C'[f(s'^*)]$ for some context $C' \sim_{\mathcal{E}} C$ and some term $f(s'^*)$ such that $f(s^*) \overset{\Delta}{\sim}_{\mathcal{E}} f(s'^*)$.*

The second result states that the order of application of equations from \mathcal{E}_1 and \mathcal{E}_2 can be interchanged in $\sim_{\mathcal{E}_1 \cup \mathcal{E}_2}$. This observation is useful in order to simplify proofs of other results.

Lemma 2.22. *Let $\mathcal{E}_1, \mathcal{E}_2$ be i.u.v. sets of equations such that \mathcal{E}_1 and \mathcal{E}_2 are signature-disjoint. Then $\sim_{\mathcal{E}_1 \cup \mathcal{E}_2} = \sim_{\mathcal{E}_1} \circ \sim_{\mathcal{E}_2}$.*

Notice that the assumption that \mathcal{E}_1 and \mathcal{E}_2 are i.u.v. is essential for this result. To see this, let $\mathcal{E}_1 = \{f(x, x) \approx x\}$ and $\mathcal{E}_2 = \{b \approx a, c \approx a\}$. Notice that \mathcal{E}_1 is not i.u.v. Then $f(b, c) \sim_{\mathcal{E}_1 \cup \mathcal{E}_2} a$, but the only terms obtainable from $f(b, c)$ using $\sim_{\mathcal{E}_1} \circ \sim_{\mathcal{E}_2}$ are $f(b, c)$, $f(a, c)$, $f(b, a)$, and $f(a, a)$.

A (*rewrite*) *rule* is an equation $l \approx r$ such that l is not a variable and $\mathcal{V}(r) \subseteq \mathcal{V}(l)$. Rewrite rules are usually written as $l \rightarrow r$. In contrast to the rewrite relation of a set \mathcal{E} of equations, the main interest for a set \mathcal{R} of rules is in $\rightarrow_{\mathcal{R}}$, its transitive closure $\rightarrow_{\mathcal{R}}^+$, and its reflexive-transitive closure $\rightarrow_{\mathcal{R}}^*$. A set \mathcal{R} of rules is usually called a *term rewrite system (TRS)*.

2.4 Equational Rewriting

Equational rewriting uses both a set \mathcal{E} of equations and a set \mathcal{R} of rules. Intuitively, \mathcal{E} is used to model “structural” properties, while \mathcal{R} is used to model “simplifying” properties. Here, simplifying properties are equations that result in a simpler term if they are applied as rewrite rules. In Example 2.16, (3) and (4) are simplifying if they are used as rewrite rules oriented from left to right. The equations (1) and (2), on the other hand, express structural properties.

There are (at least) two ways to define a rewrite relation using \mathcal{E} and \mathcal{R} that are commonly used in the literature. The first rewrite relation is defined in terms of the \mathcal{E} -equivalence classes.

Definition 2.23 (Rewriting Modulo \mathcal{E} [120]). *Let \mathcal{R} be a TRS and let \mathcal{E} be a set of equations. Then $s \rightarrow_{\mathcal{R}/\mathcal{E}} t$ if there exist terms s', t' , a rule $l \rightarrow r \in \mathcal{R}$, a position $p \in \text{Pos}(s')$, and a substitution σ such that*

1. $s \sim_{\mathcal{E}} s'$,
2. $s'|_p = l\sigma$,
3. $t' = s'[r\sigma]_p$, and
4. $t' \sim_{\mathcal{E}} t$.

Thus, $s \rightarrow_{\mathcal{R}/\mathcal{E}} t$ iff there exist a term s' in the \mathcal{E} -equivalence class of s and a

term t' in the \mathcal{E} -equivalence class of t such that $s' \rightarrow_{\mathcal{R}} t'$, or, more concisely, iff $s \sim_{\mathcal{E}} \circ \rightarrow_{\mathcal{R}} \circ \sim_{\mathcal{E}} t$.

Example 2.24. Recall the following equations from Example 2.16:

$$\begin{array}{ll} x \cup y \approx y \cup x & x \cup x \approx x \\ x \cup (y \cup z) \approx (x \cup y) \cup z & x \cup \emptyset \approx x \end{array}$$

It is then possible to obtain a TRS \mathcal{R} and a set of equations \mathcal{E} from them as follows:

$$\begin{array}{ll} \mathcal{R}: & x \cup x \rightarrow x \\ & x \cup \emptyset \rightarrow x \\ \mathcal{E}: & x \cup y \approx y \cup x \\ & x \cup (y \cup z) \approx (x \cup y) \cup z \end{array}$$

Now consider the term $t = \{\mathbf{s}(\mathcal{O})\} \cup (\{\mathcal{O}\} \cup \{\mathbf{s}(\mathcal{O})\})$. Then $t \sim_{\mathcal{E}} \{\mathcal{O}\} \cup (\{\mathbf{s}(\mathcal{O})\} \cup \{\mathbf{s}(\mathcal{O})\}) \rightarrow_{\mathcal{R}} \{\mathcal{O}\} \cup \{\mathbf{s}(\mathcal{O})\} \sim_{\mathcal{E}} \{\mathbf{s}(\mathcal{O})\} \cup \{\mathcal{O}\}$ and thus $t \rightarrow_{\mathcal{R}/\mathcal{E}} \{\mathbf{s}(\mathcal{O})\} \cup \{\mathcal{O}\}$. \triangle

Notice that deciding whether $s \rightarrow_{\mathcal{R}/\mathcal{E}} t$ is true might be hard since the whole equivalence classes of s and t need to be considered. This is in particular true if the \mathcal{E} -equivalence classes are infinite.

\mathcal{E} -extended rewriting is a restriction of rewriting modulo \mathcal{E} that provides better decidability results in practice. First, the search space is restricted since the position p where the reduction takes place is fixed *before* considering the equations in \mathcal{E} . Furthermore, \mathcal{E} is only applied below that position p in s , i.e., the \mathcal{E} -equivalence class of $s|_p$ is considered. Finally, the \mathcal{E} -equivalence class of t is not considered at all.

Definition 2.25 (\mathcal{E} -Extended Rewriting [139]). *Let \mathcal{R} be a TRS and let \mathcal{E} be a set of equations. Then $s \rightarrow_{\mathcal{E}\setminus\mathcal{R}} t$ if there exist a rule $l \rightarrow r \in \mathcal{R}$, a position $p \in \text{Pos}(s)$, and a substitution σ such that*

1. $s|_p \sim_{\mathcal{E}} l\sigma$, and
2. $t = s[r\sigma]_p$.

Example 2.26. Continuing Example 2.24, consider $t' = \{\mathbf{s}(\mathcal{O})\} \cup (\emptyset \cup \{\mathcal{O}\})$. Then $t'|_2 \sim_{\mathcal{E}} \{\mathcal{O}\} \cup \emptyset$ and thus $t' \rightarrow_{\mathcal{E} \setminus \mathcal{R}} \{\mathbf{s}(\mathcal{O})\} \cup \{\mathcal{O}\}$ using the second rule from \mathcal{R} . \triangle

In general, rewriting modulo \mathcal{E} and \mathcal{E} -extended rewriting do not coincide. To see this, again consider the term $t = \{\mathbf{s}(\mathcal{O})\} \cup (\{\mathcal{O}\} \cup \{\mathbf{s}(\mathcal{O})\})$ from Example 2.24. Recall that $t \rightarrow_{\mathcal{R}/\mathcal{E}} \{\mathbf{s}(\mathcal{O})\} \cup \{\mathcal{O}\}$ and notice that t is irreducible by $\rightarrow_{\mathcal{E} \setminus \mathcal{R}}$ since no rewrite rule can be applied at any position.

Another view on the definition of $\rightarrow_{\mathcal{E} \setminus \mathcal{R}}$ can be obtained by noticing that $\rightarrow_{\mathcal{E} \setminus \mathcal{R}}$ differs from $\rightarrow_{\mathcal{R}}$ by replacing matching with \mathcal{E} -matching, where a term s \mathcal{E} -matches a term t if there exists a substitution σ such that $s\sigma \sim_{\mathcal{E}} t$. Thus, if it is possible to compute an \mathcal{E} -matching substitution, then $\rightarrow_{\mathcal{E} \setminus \mathcal{R}}$ is much easier to compute than $\rightarrow_{\mathcal{R}/\mathcal{E}}$. It will be shown in Lemma 3.20 that this is always possible for the sets \mathcal{E} of equations that are considered in this dissertation, and the remainder of this work therefore uses \mathcal{E} -extended rewriting.

Similarly to the concept of matching, the concept of unification can also be extended to obtain \mathcal{E} -unification. Here, a substitution σ is an \mathcal{E} -unifier of s and t iff $s\sigma \sim_{\mathcal{E}} t\sigma$. Whether it is decidable if two terms are \mathcal{E} -unifiable and whether an \mathcal{E} -unifier is computable depends on the set \mathcal{E} of equations. See [100, 19, 20] for surveys on decidability results.

Given \mathcal{E} and \mathcal{R} , an investigation of the normal forms of $\rightarrow_{\mathcal{E} \setminus \mathcal{R}}$ is of interest. For modeling built-in numbers and collection data structures as proposed in this dissertation, it is highly desirable that all terms of the same \mathcal{E} -equivalence class have normal forms that are again in the same \mathcal{E} -equivalence class, i.e., that $\rightarrow_{\mathcal{E} \setminus \mathcal{R}}$ does not make a distinction between terms that are equivalent up to $\sim_{\mathcal{E}}$. Under suitable assumptions, this goal can indeed be achieved.

Definition 2.27 (\mathcal{E} -Confluence, \mathcal{E} -Convergence, Strong \mathcal{E} -Coherence). *Let \mathcal{R} be a TRS and let \mathcal{E} be a set of equations.*

Chapter 2. Many-Sorted Term Rewriting

1. $\rightarrow_{\mathcal{E}\setminus\mathcal{R}}$ is \mathcal{E} -confluent iff $\leftarrow_{\mathcal{E}\setminus\mathcal{R}}^* \circ \rightarrow_{\mathcal{E}\setminus\mathcal{R}}^* \subseteq \rightarrow_{\mathcal{E}\setminus\mathcal{R}}^* \circ \sim_{\mathcal{E}} \circ \leftarrow_{\mathcal{E}\setminus\mathcal{R}}^*$.
2. $\rightarrow_{\mathcal{E}\setminus\mathcal{R}}$ is \mathcal{E} -convergent iff it is terminating and \mathcal{E} -confluent.
3. $\rightarrow_{\mathcal{E}\setminus\mathcal{R}}$ is strongly \mathcal{E} -coherent iff $\sim_{\mathcal{E}}$ commutes over $\rightarrow_{\mathcal{E}\setminus\mathcal{R}}$, i.e., iff $\sim_{\mathcal{E}} \circ \rightarrow_{\mathcal{E}\setminus\mathcal{R}} \subseteq \rightarrow_{\mathcal{E}\setminus\mathcal{R}} \circ \sim_{\mathcal{E}}$.

Intuitively, \mathcal{E} -confluence means that whenever a term can be reduced using two reduction sequences, then the resulting terms can be further reduced to terms that are in the same \mathcal{E} -equivalence class. Strong \mathcal{E} -coherence means that if two terms are in the same \mathcal{E} -equivalence class and one of the terms can be reduced, then the other term can also be reduced and the resulting terms are furthermore in the same equivalence class as well. Notice that strong \mathcal{E} -coherence is a special case of coherence modulo \mathcal{E} [101]. Using \mathcal{E} -convergence and strong \mathcal{E} -coherence, the following characterization of normal forms is obtained. It states that two terms from the same \mathcal{E} -equivalence class have normal forms that are again in the same \mathcal{E} -equivalence class.

Lemma 2.28. *Let \mathcal{R} be a TRS and let \mathcal{E} be a set of equations such that $\rightarrow_{\mathcal{E}\setminus\mathcal{R}}$ is \mathcal{E} -convergent and strongly \mathcal{E} -coherent. If $s \sim_{\mathcal{E}} t$, $s \rightarrow_{\mathcal{E}\setminus\mathcal{R}}^! \hat{s}$, and $t \rightarrow_{\mathcal{E}\setminus\mathcal{R}}^! \hat{t}$, then $\hat{s} \sim_{\mathcal{E}} \hat{t}$.*

Example 2.29. The relation $\rightarrow_{\mathcal{E}\setminus\mathcal{R}}$ using \mathcal{E} and \mathcal{R} from Example 2.24 is not strongly \mathcal{E} -coherent (and, in fact, the statement of Lemma 2.28 does not hold). To see this, consider the terms $t = \{\mathbf{s}(\mathcal{O})\} \cup (\{\mathcal{O}\} \cup \{\mathcal{O}\})$ and $t' = \{\mathcal{O}\} \cup (\{\mathbf{s}(\mathcal{O})\} \cup \{\mathcal{O}\})$. Then $t' \sim_{\mathcal{E}} t \rightarrow_{\mathcal{E}\setminus\mathcal{R}} \{\mathbf{s}(\mathcal{O})\} \cup \{\mathcal{O}\}$, but t' cannot be reduced using $\rightarrow_{\mathcal{E}\setminus\mathcal{R}}$. If the rewrite rule $(x \cup x) \cup y \rightarrow x \cup y$ is added to \mathcal{R} , then strong \mathcal{E} -coherence is achieved and $t' \rightarrow_{\mathcal{E}\setminus\mathcal{R}} \{\mathcal{O}\} \cup \{\mathbf{s}(\mathcal{O})\} \sim_{\mathcal{E}} \{\mathbf{s}(\mathcal{O})\} \cup \{\mathcal{O}\}$. \triangle

The observation of Example 2.29 can be generalized, i.e., it may be possible to achieve strong \mathcal{E} -coherence by adding *extended rules* [139, 81]. This is not going to be discussed further in this dissertation, but it will be mentioned when extended rules need to be added.

Chapter 2. Many-Sorted Term Rewriting

Apart from Lemma 2.28, strong \mathcal{E} -coherence also provides a close connection between rewriting modulo \mathcal{E} and \mathcal{E} -extended rewriting [81]. This property is not needed in the remainder of this dissertation, but it serves as a further justification for restricting attention to \mathcal{E} -extended rewriting.

Lemma 2.30. *Let \mathcal{R} be a TRS and let \mathcal{E} be a set of equations such that $\rightarrow_{\mathcal{E}\setminus\mathcal{R}}$ is strongly \mathcal{E} -coherent. If $s \rightarrow_{\mathcal{R}/\mathcal{E}} t$, then $s \rightarrow_{\mathcal{E}\setminus\mathcal{R}} t'$ for some $t' \sim_{\mathcal{E}} t$.*

The built-in numbers and collection data structures considered in this dissertation will be modeled using TRSs \mathcal{R}_i and sets \mathcal{E}_i of equations such that $\rightarrow_{\mathcal{E}_i\setminus\mathcal{R}_i}$ is \mathcal{E}_i -confluent and strongly \mathcal{E}_i -coherent for each data structure on its own. If more than one of these data structures is to be used at the same time, the question whether $\rightarrow_{\mathcal{E}\setminus\mathcal{R}}$ is \mathcal{E} -confluent and strongly \mathcal{E} -coherent arises, where $\mathcal{R} = \bigcup_{i=1}^n \mathcal{R}_i$ and $\mathcal{E} = \bigcup_{i=1}^n \mathcal{E}_i$. Proving these properties automatically seems to be non-trivial, but it turns out that this is not necessary since it can be shown that \mathcal{E} -confluence and strong \mathcal{E} -coherence are modular properties in the following sense. Building upon results from [101, 99], this lemma is the main technical contribution of this chapter.

Lemma 2.31. *Let $\mathcal{R}_1, \mathcal{R}_2$ be TRSs and let $\mathcal{E}_1, \mathcal{E}_2$ be collapse-free i.u.v. sets of equations such that*

1. $\mathcal{R}_1 \cup \mathcal{E}_1$ is signature-disjoint from $\mathcal{R}_2 \cup \mathcal{E}_2$,
2. $\rightarrow_{\mathcal{E}_1\setminus\mathcal{R}_1}$ is \mathcal{E}_1 -convergent and strongly \mathcal{E}_1 -coherent, and
3. $\rightarrow_{\mathcal{E}_2\setminus\mathcal{R}_2}$ is \mathcal{E}_2 -convergent and strongly \mathcal{E}_2 -coherent.

If $\rightarrow_{\mathcal{E}_1\cup\mathcal{E}_2\setminus\mathcal{R}_1\cup\mathcal{R}_2}$ is terminating, then $\rightarrow_{\mathcal{E}_1\cup\mathcal{E}_2\setminus\mathcal{R}_1\cup\mathcal{R}_2}$ is $\mathcal{E}_1 \cup \mathcal{E}_2$ -convergent and strongly $\mathcal{E}_1 \cup \mathcal{E}_2$ -coherent.

Chapter 3

Constrained Equational Rewrite Systems

This chapter introduces the class of term rewrite systems that is used for modeling algorithms in this dissertation. Both built-in numbers and collection data structures are modeled using \mathcal{E} -extended rewriting as in Definition 2.25. It is first discussed how this is done for built-in numbers. The same approach can then be applied to collection data structures as well.

3.1 Built-In Numbers as Canonizable Theories

In order to model the set of integers using terms and equations, recall that \mathbb{Z} is an Abelian group under addition with unit element 0 that is generated using the element 1. A signature for modeling integers thus consists of the following function symbols:

$$0 : \rightarrow \text{int}$$

$$1 : \rightarrow \text{int}$$

Chapter 3. Constrained Equational Rewrite Systems

$$- : \text{int} \rightarrow \text{int}$$

$$+ : \text{int} \times \text{int} \rightarrow \text{int}$$

The defining properties of Abelian groups can easily be stated as equations:

$$x + y \approx y + x$$

$$x + (y + z) \approx (x + y) + z$$

$$x + 0 \approx x$$

$$x + (-x) \approx 0$$

This set of equations is in general unsuitable for modeling algorithms using extended rewriting since the last equation is not regular. To avoid this problem, the idea is to keep only associativity and commutativity as equations and to turn the remaining properties (idempotency and inverse elements) into rewrite rules. For this, it does not suffice to just orient the last two equations as rules. Instead, the well-known method of *equational completion* [101, 21] needs to be applied to obtain “equivalent” sets \mathcal{S} of rewrite rules and \mathcal{E} of equations. Here, the goal is that two terms are in the same equivalence class of the initial set of equations if and only if their normal forms w.r.t. $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ are in the same \mathcal{E} -equivalence class. Applying equational completion to the above properties of the integers produces the following rewrite rules and equations:

$$\mathcal{S} : \quad x + 0 \rightarrow x$$

$$- - x \rightarrow x$$

$$-0 \rightarrow 0$$

$$-(x + y) \rightarrow (-x) + (-y)$$

$$x + (-x) \rightarrow 0$$

$$(x + (-x)) + y \rightarrow 0 + y$$

$$\mathcal{E} : \quad x + y \approx y + x$$

$$x + (y + z) \approx (x + y) + z$$

Chapter 3. Constrained Equational Rewrite Systems

This idea can be generalized from the integers, and it can furthermore be made more useful by allowing predefined predicate symbols that can then be used as constraints in the rewrite rules used for modeling algorithms. In order to model built-in numbers and other built-in data structures, the concept of a *theory* is used.

Definition 3.1 (Theories). A theory $Th = (\mathcal{F}_{Th}, \mathbf{P}_{Th}, \mathcal{M}_{Th})$ consists of:

1. A finite signature \mathcal{F}_{Th} over a single sort **base**.
2. A finite set \mathbf{P}_{Th} of predicate symbols, each coming with an arity.
3. A structure $\mathcal{M}_{Th} = (M, (f^{Th})_{f \in \mathcal{F}_{Th}}, (P^{Th})_{P \in \mathbf{P}_{Th}})$ over a set M that interprets the symbols in \mathcal{F}_{Th} and \mathbf{P}_{Th} , i.e., each n -ary $f \in \mathcal{F}_{Th}$ is mapped to a function $f^{Th} : M^n \rightarrow M$ and each n -ary $P \in \mathbf{P}_{Th}$ is mapped to a subset $P^{Th} \subseteq M^n$.

Notice that the interpretations f^{Th} of the function symbols define an interpretation of each ground term $s \in \mathcal{T}(\mathcal{F}_{Th})$. This interpretation is denoted by s^{Th} .

Example 3.2. For integers, $Th_{\mathbb{Z}} = (\mathcal{F}_{Th_{\mathbb{Z}}}, \mathbf{P}_{Th_{\mathbb{Z}}}, \mathcal{M}_{Th_{\mathbb{Z}}})$ with

1. $\mathcal{F}_{Th_{\mathbb{Z}}} = \{0, 1, -, +\}$
2. $\mathbf{P}_{Th_{\mathbb{Z}}} = \{\simeq, \geq, >\}$
3. $\mathcal{M}_{Th_{\mathbb{Z}}} = (\mathbb{Z}, 0^{Th_{\mathbb{Z}}}, 1^{Th_{\mathbb{Z}}}, -^{Th_{\mathbb{Z}}}, +^{Th_{\mathbb{Z}}}, \simeq^{Th_{\mathbb{Z}}}, \geq^{Th_{\mathbb{Z}}}, >^{Th_{\mathbb{Z}}})$ where
 - $0^{Th_{\mathbb{Z}}} = 0$
 - $1^{Th_{\mathbb{Z}}} = 1$
 - $-^{Th_{\mathbb{Z}}}(x) = -x$
 - $+^{Th_{\mathbb{Z}}}(x, y) = x + y$
 - $\simeq^{Th_{\mathbb{Z}}} = \{(x, y) \mid x = y\}$
 - $\geq^{Th_{\mathbb{Z}}} = \{(x, y) \mid x \geq y\}$
 - $>^{Th_{\mathbb{Z}}} = \{(x, y) \mid x > y\}$

is a theory. For this theory, $(1 + (1 + (-0)))^{\mathcal{Th}_{\mathbb{Z}}} = 2$. \triangle

As already mentioned above, the rewrite rules that are used for modeling algorithms will have constraints from \mathcal{Th} that guard when a rewrite step may be performed.

Definition 3.3 (Syntax of \mathcal{Th} -Constraints). *An atomic \mathcal{Th} -constraint has the form $Pt_1 \dots t_n$ for a predicate symbol $P \in \mathcal{P}_{\mathcal{Th}}$ and terms $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F}_{\mathcal{Th}}, \mathcal{V})$. The set of \mathcal{Th} -constraints is inductively defined as follows:*

1. \top is a \mathcal{Th} -constraint.
2. Every atomic \mathcal{Th} -constraint is a \mathcal{Th} -constraint.
3. If φ is a \mathcal{Th} -constraint, then $\neg\varphi$ is a \mathcal{Th} -constraint.
4. If φ_1, φ_2 are \mathcal{Th} -constraints, then $\varphi_1 \wedge \varphi_2$ is a \mathcal{Th} -constraint.

Example 3.4. For the theory of integers $\mathcal{Th}_{\mathbb{Z}}$ from Example 3.2, $x+1 \simeq y \wedge \neg(x > z)$ is a $\mathcal{Th}_{\mathbb{Z}}$ -constraint. \triangle

The Boolean connectives \vee , \Rightarrow , and \Leftrightarrow can be defined as usual. Also, \mathcal{Th} -constraints have the expected semantics. The main interest is in *satisfiability* (i.e., the constraint is true for some instantiation of its variables) and *validity* (i.e., the constraint is true for all instantiations of its variables).

Definition 3.5 (Semantics of \mathcal{Th} -Constraints). *A variable-free \mathcal{Th} -constraint φ is \mathcal{Th} -valid iff*

1. φ has the form \top , or
2. φ is an atomic \mathcal{Th} -constraint of the form $Pt_1 \dots t_n$ and $P^{\mathcal{Th}}t_1^{\mathcal{Th}} \dots t_n^{\mathcal{Th}}$ is true,
or
3. φ has the form $\neg\varphi'$ and φ' is not \mathcal{Th} -valid, or
4. φ has the form $\varphi_1 \wedge \varphi_2$ and both φ_1 and φ_2 are \mathcal{Th} -valid.

Chapter 3. Constrained Equational Rewrite Systems

A \mathcal{Th} -constraint φ with variables is \mathcal{Th} -valid iff $\varphi\sigma$ is \mathcal{Th} -valid for all ground substitutions $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F}_{\mathcal{Th}})$. A \mathcal{Th} -constraint φ is \mathcal{Th} -satisfiable iff there exists a ground substitution $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F}_{\mathcal{Th}})$ such that $\varphi\sigma$ is \mathcal{Th} -valid. Otherwise, φ is \mathcal{Th} -unsatisfiable.

Example 3.6. The $\mathcal{Th}_{\mathbb{Z}}$ -constraint $\varphi = x + 1 \simeq y \wedge \neg(x > z)$ from Example 3.4 is $\mathcal{Th}_{\mathbb{Z}}$ -satisfiable since its ground instantiation using the substitution $\{x \mapsto 0, y \mapsto 1, z \mapsto 1\}$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid. The constraint φ is not $\mathcal{Th}_{\mathbb{Z}}$ -valid since its ground instantiation using the substitution $\{x \mapsto 0, y \mapsto 0, z \mapsto 0\}$ is not $\mathcal{Th}_{\mathbb{Z}}$ -valid. \triangle

Within this dissertation it is assumed that each theory contains the equality predicate \simeq and that \simeq can be axiomatized using a finite set $\mathcal{E}_{\mathcal{Th}}$ of equations.

Definition 3.7 (Theories with Equality). *A theory with equality is a theory \mathcal{Th} with a predicate symbol $\simeq \in \mathcal{P}_{\mathcal{Th}}$ which is interpreted as the equality in $\mathcal{M}^{\mathcal{Th}}$. Furthermore, there exists a finite set $\mathcal{E}_{\mathcal{Th}}$ that axiomatizes \simeq , i.e., for all terms $s, t \in \mathcal{T}(\mathcal{F}_{\mathcal{Th}}, \mathcal{V})$, the constraint $s \simeq t$ is \mathcal{Th} -valid iff $s \sim_{\mathcal{E}_{\mathcal{Th}}} t$.*

Example 3.8. The theory $\mathcal{Th}_{\mathbb{Z}}$ from Example 3.2 is a theory with equality where \simeq is axiomatized by the set of equations $\mathcal{E}_{\mathcal{Th}_{\mathbb{Z}}} = \{x + y \approx y + x, x + (y + z) \approx (x + y) + z, x + 0 \approx x, x + (-x) \approx 0\}$. \triangle

In the following, it is assumed that every theory is a theory with equality. Notice that the definition of a theory with equality does not impose any restrictions on the equations in $\mathcal{E}_{\mathcal{Th}}$. For reasons of practicality, attention is restricted to the case where $\mathcal{E}_{\mathcal{Th}}$ is *canonizable* in the following sense (recall the discussion on the defining properties of Abelian groups from above).

Definition 3.9 (Canonizable Sets of Equations). *A finite set \mathcal{E} of equations is canonizable iff there exist finite sets $\vec{\mathcal{E}}$ of rewrite rules and $\hat{\mathcal{E}}$ of equations such that*

1. $\vec{\mathcal{E}} \cup \hat{\mathcal{E}}$ is equivalent to \mathcal{E} , i.e., $\sim_{\mathcal{E}} = \sim_{\vec{\mathcal{E}} \cup \hat{\mathcal{E}}}$,

| | \mathcal{F}_{Th} | \mathcal{P}_{Th} | $\overrightarrow{\mathcal{E}}_{Th}$ and $\widehat{\mathcal{E}}_{Th}$ |
|-----------------|--------------------|--------------------|--|
| Natural numbers | $0, 1, +$ | $\simeq, \geq, >$ | $x + 0 \rightarrow x$ $x + y \approx y + x$ $x + (y + z) \approx (x + y) + z$ |
| Integers | $0, 1, +, -$ | $\simeq, \geq, >$ | $x + 0 \rightarrow x$ $- - x \rightarrow x$ $-0 \rightarrow 0$ $-(x + y) \rightarrow (-x) + (-y)$ $x + (-x) \rightarrow 0$ $(x + (-x)) + y \rightarrow 0 + y$ $x + y \approx y + x$ $x + (y + z) \approx (x + y) + z$ |

Figure 3.1: Numbers as canonizable theories.

2. $\widehat{\mathcal{E}}$ is i.u.v.,
3. $\rightarrow_{\widehat{\mathcal{E}} \setminus \overrightarrow{\mathcal{E}}}$ is $\widehat{\mathcal{E}}$ -convergent, and
4. $\rightarrow_{\widehat{\mathcal{E}} \setminus \overrightarrow{\mathcal{E}}}$ is strongly $\widehat{\mathcal{E}}$ -coherent.

For a canonizable set \mathcal{E} of equations, $s \sim_{\mathcal{E}} t$ if and only if $s \xrightarrow{!}_{\widehat{\mathcal{E}} \setminus \overrightarrow{\mathcal{E}}} \circ \sim_{\widehat{\mathcal{E}}} \circ \xleftarrow{!}_{\widehat{\mathcal{E}} \setminus \overrightarrow{\mathcal{E}}} t$ [101]. The sets $\overrightarrow{\mathcal{E}}$ and $\widehat{\mathcal{E}}$ can be obtained from \mathcal{E} using *equational completion* [101, 21]. A theory Th is said to be canonizable if \mathcal{E}_{Th} is.

Example 3.10. The theory $Th_{\mathbb{Z}}$ is canonizable since the set $\mathcal{E}_{Th_{\mathbb{Z}}}$ from Example 3.8 can be completed into $\overrightarrow{\mathcal{E}}_{Th_{\mathbb{Z}}} = \{x + 0 \rightarrow x, - - x \rightarrow x, -0 \rightarrow 0, -(x + y) \rightarrow (-x) + (-y), x + (-x) \rightarrow 0, (x + (-x)) + y \rightarrow 0 + y\}$ and $\widehat{\mathcal{E}}_{Th_{\mathbb{Z}}} = \{x + y \approx y + x, x + (y + z) \approx (x + y) + z\}$ which satisfy the conditions from Definition 3.9. \triangle

Figure 3.1 lists two of the most important canonizable theories: $Th_{\mathbb{N}}$ is the (linear) theory of natural numbers, and $Th_{\mathbb{Z}}$ is the (linear) theory of integers. For these theories, the structures \mathcal{M}_{Th} use \mathbb{N} and \mathbb{Z} as universes, respectively. The function symbols in \mathcal{F}_{Th} and the predicate symbols in \mathcal{P}_{Th} are interpreted in the natural way.

For $\mathcal{Th}_{\mathbb{Z}}$, the rewrite rule $(x + (-x)) + y \rightarrow 0 + y$ is an *extension* of $x + (-x) \rightarrow 0$ that is needed to make $\rightarrow_{\widehat{\mathcal{E}} \setminus \overline{\mathcal{E}}}$ strongly $\widehat{\mathcal{E}}$ -coherent. This dissertation takes some liberties in writing terms for $\mathcal{Th}_{\mathbb{N}}$ and $\mathcal{Th}_{\mathbb{Z}}$. For example, $x - 2$ is shorthand for any term that is equivalent to $x + ((-1) + (-1))$. Notice that both $\mathcal{Th}_{\mathbb{N}}$ - and $\mathcal{Th}_{\mathbb{Z}}$ -validity are decidable [146]. This is no longer true for the full theory of natural numbers or integers which, in addition to $+$ and $-$, also contains multiplication [90].

3.2 Canonizable Collection Data Structures

In order to extend \mathcal{Th} by collection data structures and defined functions, $\mathcal{F}_{\mathcal{Th}}$ is extended by a finite signature \mathcal{F} over the sort **base** and a set of new sorts.

Collection data structures can be handled similarly to the built-in theories, i.e., properties of collection data structures are modeled using a finite set $\mathcal{E}_{\mathcal{C}}$ of equations. As in Section 3.1, attention is restricted to the case where $\mathcal{E}_{\mathcal{C}}$ is canonizable. Collection data structures that satisfy this property are also called *canonizable collection data structures*. Some commonly used canonizable collection data structures are given in Figure 3.2. *Compact lists* are lists where the number of contiguous occurrences of the same element does not matter. These kinds of lists have been used in the constraint logic programming framework [58, 57]. Notice that there are typically two possibilities for modeling the “same” collection data structure:

1. Using a list-like representation with an empty collection and a constructor to add an element to a collection.
2. Using an empty collection, singleton collections ($\langle \cdot \rangle$ and $\{ \cdot \}$, respectively), and a constructor to concatenate two collections.

For the second possibility to model sets, the rewrite rule $(x \cup x) \cup y \rightarrow x \cup y$ is an extension of $x \cup x \rightarrow x$ that is needed to make $\rightarrow_{\widehat{\mathcal{E}}_{\mathcal{C}} \setminus \overline{\mathcal{E}}_{\mathcal{C}}}$ strongly $\widehat{\mathcal{E}}_{\mathcal{C}}$ -coherent.

| | Constructors | $\vec{\mathcal{E}}_{\mathcal{C}}$ and $\widehat{\mathcal{E}}_{\mathcal{C}}$ |
|---------------|---------------------------|--|
| Lists | <code>nil, cons</code> | n/a |
| Lists | <code>nil, ⟨·⟩, ++</code> | $x ++ \text{nil} \rightarrow x$ $\text{nil} ++ y \rightarrow y$ $x ++ (y ++ z) \approx (x ++ y) ++ z$ |
| Compact Lists | <code>nil, cons</code> | $\text{cons}(x, \text{cons}(x, ys)) \rightarrow \text{cons}(x, ys)$ |
| Compact Lists | <code>nil, ⟨·⟩, ++</code> | $x ++ \text{nil} \rightarrow x$ $\text{nil} ++ y \rightarrow y$ $\langle x \rangle ++ \langle x \rangle \rightarrow \langle x \rangle$ $x ++ (y ++ z) \approx (x ++ y) ++ z$ |
| Multisets | <code>∅, ins</code> | $\text{ins}(x, \text{ins}(y, zs)) \approx \text{ins}(y, \text{ins}(x, zs))$ |
| Multisets | <code>∅, {·}, ∪</code> | $x \cup \emptyset \rightarrow x$ $x \cup (y \cup z) \approx (x \cup y) \cup z$ $x \cup y \approx y \cup x$ |
| Sets | <code>∅, ins</code> | $\text{ins}(x, \text{ins}(x, ys)) \rightarrow \text{ins}(x, ys)$ $\text{ins}(x, \text{ins}(y, zs)) \approx \text{ins}(y, \text{ins}(x, zs))$ |
| Sets | <code>∅, {·}, ∪</code> | $x \cup \emptyset \rightarrow x$ $x \cup x \rightarrow x$ $(x \cup x) \cup y \rightarrow x \cup y$ $x \cup (y \cup z) \approx (x \cup y) \cup z$ $x \cup y \approx y \cup x$ |

Figure 3.2: Commonly used canonizable collection data structures.

3.3 Constrained Equational Rewrite Systems

In the following, a combination of \mathcal{Th} with none or more (signature-disjoint) canonizable collection data structures $\mathcal{C}_1, \dots, \mathcal{C}_n$ is considered. For this, let $\mathcal{S} = \vec{\mathcal{E}}_{\mathcal{Th}} \cup \bigcup_{i=1}^n \vec{\mathcal{E}}_{\mathcal{C}_i}$ and $\mathcal{E} = \widehat{\mathcal{E}}_{\mathcal{Th}} \cup \bigcup_{i=1}^n \widehat{\mathcal{E}}_{\mathcal{C}_i}$.

A constrained equational rewrite systems contains constrained rewrite rules. As formalized below in Definition 3.15, the \mathcal{Th} -constraint guards when a rewrite step may be performed. In contrast to conditional rewriting (see, e.g., [137]), this will *not* be done by recursively rewriting the constraints. Instead, a decision procedure

for \mathcal{Th} -validity will be employed. This distinction is further discussed in Chapter 10, where conditional constrained rewrite rules are considered.

Definition 3.11 (Constrained Rewrite Rules). *A constrained rewrite rule has the form $l \rightarrow r \llbracket \varphi \rrbracket$ for terms $l, r \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{Th}}, \mathcal{V})$ and a \mathcal{Th} -constraint φ such that $\text{root}(l) \in \mathcal{F} - \mathcal{F}(\mathcal{E} \cup \mathcal{S})$ and $\mathcal{V}(r) \cup \mathcal{V}(\varphi) \subseteq \mathcal{V}(l)$. In a rule $l \rightarrow r \llbracket \top \rrbracket$, the constraint \top will usually be omitted.*

A finite set of constrained rewrite rules and the sets \mathcal{S} and \mathcal{E} for modeling \mathcal{Th} and collection data structures are combined into a *constrained equational rewrite system*.

Definition 3.12 (Constrained Equational Rewrite Systems (CERSs)). *A constrained equational rewrite system (CERS) has the form $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ for a finite set \mathcal{R} of constrained rewrite rules, a finite set \mathcal{E} of equations, and a finite set \mathcal{S} of rewrite rules such that*

1. \mathcal{S} is right-linear, i.e., each variable occurs at most once in r for all $l \rightarrow r \in \mathcal{S}$,
2. $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ is \mathcal{E} -convergent, and
3. $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ is strongly \mathcal{E} -coherent.

Notice that for the combination of a canonizable theory from Figure 3.1 with none or more (signature-disjoint) canonizable collection data structures from Figure 3.2, $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ is \mathcal{E} -convergent and strongly \mathcal{E} -coherent by Lemma 2.31 since $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ is still terminating. Definition 3.12 states these requirements for reference and in order to make the techniques developed in this dissertation more general. Condition 1 in Definition 3.12, i.e., that \mathcal{S} needs to be right-linear, is of a technical nature. It is currently unclear whether it can be relaxed.

Example 3.13. This example shows a quicksort algorithm that takes a set of integers and returns a sorted list of the elements of that set. For this, integers are modeled as in Figure 3.1 and sets are modeled using \emptyset , $\{\cdot\}$, and \cup as in Figure 3.2.

Chapter 3. Constrained Equational Rewrite Systems

Notice that the assumption that the input to `qsort` is a set is enforced by the choice of \mathcal{S} and \mathcal{E} . By choosing different sets \mathcal{S} and \mathcal{E} , the input to `qsort` can be treated as a multiset. How the choice of \mathcal{S} and \mathcal{E} influences the result of `qsort` follows from the definition of the rewrite relation for CERSs (Definition 3.15 below).

$$\begin{array}{l}
 \mathcal{S} : \quad x + 0 \rightarrow x \\
 \quad \quad - - x \rightarrow x \\
 \quad \quad -0 \rightarrow 0 \\
 \quad \quad -(x + y) \rightarrow (-x) + (-y) \\
 \quad \quad x + (-x) \rightarrow 0 \\
 (x + (-x)) + y \rightarrow 0 + y \\
 \quad \quad x \cup \emptyset \rightarrow x \\
 \quad \quad x \cup x \rightarrow x \\
 (x \cup x) \cup y \rightarrow x \cup y \\
 \mathcal{E} : \quad x + y \approx y + x \\
 \quad \quad x + (y + z) \approx (x + y) + z \\
 \quad \quad x \cup y \approx y \cup x \\
 \quad \quad x \cup (y \cup z) \approx (x \cup y) \cup z
 \end{array}$$

The quicksort algorithm is specified by the following constrained rewrite rules:

$$\begin{array}{l}
 \text{app}(\text{nil}, zs) \rightarrow zs \\
 \text{app}(\text{cons}(x, ys), zs) \rightarrow \text{cons}(x, \text{app}(ys, zs)) \\
 \text{low}(x, \emptyset) \rightarrow \emptyset \\
 \text{low}(x, \{y\}) \rightarrow \{y\} \llbracket x > y \rrbracket \\
 \text{low}(x, \{y\}) \rightarrow \emptyset \llbracket x \not> y \rrbracket \\
 \text{low}(x, y \cup z) \rightarrow \text{low}(x, y) \cup \text{low}(x, z) \\
 \text{high}(x, \emptyset) \rightarrow \emptyset \\
 \text{high}(x, \{y\}) \rightarrow \{y\} \llbracket x \not> y \rrbracket
 \end{array}$$

$$\begin{aligned}
 \text{high}(x, \{y\}) &\rightarrow \emptyset \llbracket x > y \rrbracket \\
 \text{high}(x, y \cup z) &\rightarrow \text{high}(x, y) \cup \text{high}(x, z) \\
 \text{qsort}(\emptyset) &\rightarrow \text{nil} \\
 \text{qsort}(\{x\}) &\rightarrow \text{cons}(x, \text{nil}) \\
 \text{qsort}(\{x\} \cup y) &\rightarrow \text{app}(\text{qsort}(\text{low}(x, y)), \text{cons}(x, \text{qsort}(\text{high}(x, y))))
 \end{aligned}$$

Notice that this specification is quite natural. △

The rewrite relation obtained from a CERS is based on the key idea of [128]. Instead of using $\mathcal{E} \cup \mathcal{S}$ -extended rewriting as in Definition 2.25, it operates on terms that are suitably normalized with $\rightarrow_{\mathcal{E} \cup \mathcal{S}}$, thus picking a unique (up to $\sim_{\mathcal{E}}$) representative of each $\mathcal{E} \cup \mathcal{S}$ -equivalence class. This normalization process closely captures the intuition that is typically employed in the development of algorithms. For example, it is typically assumed that sets do not contain duplicate elements. This assumption might even be essential for termination of the algorithm.

The normalization process motivated above is given more formally as follows. First, the subterm where a rule from \mathcal{R} should be applied is normalized with $\xrightarrow{\geq \Lambda}_{\mathcal{E} \cup \mathcal{S}}$. Then, \mathcal{E} -matching is performed. In order to take the constraint of the rewrite rule into account, it is additionally required that this constraint becomes \mathcal{Th} -valid after being instantiated by the matcher. If the matcher instantiates all variables of sort **base** by terms from $\mathcal{T}(\mathcal{F}_{\mathcal{Th}}, \mathcal{V})$, then this question can be answered by a decision procedure for \mathcal{Th} -validity.

Definition 3.14 (*\mathcal{Th} -Based Substitutions*). *A substitution σ is \mathcal{Th} -based iff $\sigma(x) \in \mathcal{T}(\mathcal{F}_{\mathcal{Th}}, \mathcal{V})$ for all variables x of sort **base**.*

The rewrite relation is now restricted to use a \mathcal{Th} -based substitution. This restriction could be slightly relaxed by requiring $\sigma(x) \in \mathcal{T}(\mathcal{F}_{\mathcal{Th}}, \mathcal{V})$ for only those variables of sort **base** that occur in the constraint of the rewrite rule.

Definition 3.15 (Rewrite Relation of a CERS). *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CERS. Then $s \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}} t$ iff there exist a constrained rewrite rule $l \rightarrow r[\varphi] \in \mathcal{R}$, a position $p \in \mathcal{Pos}(s)$, and a \mathcal{Th} -based substitution σ such that*

1. $s|_p \xrightarrow{>\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} l\sigma \circ \gtrsim_{\mathcal{E}}^{\Lambda} l\sigma$,
2. $\varphi\sigma$ is \mathcal{Th} -valid, and
3. $t = s[r\sigma]_p$.

Example 3.16. Continuing Example 3.13, assume that the term $t = \mathbf{qsort}(\{-1\} \cup (\{1\} \cup \{-1\}))$ is to be reduced using $\xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}}$. By considering the substitution $\sigma = \{x \mapsto 1, y \mapsto \{-1\}\}$, the third \mathbf{qsort} -rule can be applied since $t \xrightarrow{>\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \mathbf{qsort}(\{-1\} \cup \{1\}) \gtrsim_{\mathcal{E}}^{\Lambda} \mathbf{qsort}(\{x\} \cup y)\sigma$. Therefore,

$$t \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}} \mathbf{app}(\mathbf{qsort}(\mathbf{low}(1, \{-1\})), \mathbf{cons}(1, \mathbf{qsort}(\mathbf{high}(1, \{-1\}))))$$

Next, $\mathbf{low}(1, \{-1\}) \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}} \{-1\}$ by the second \mathbf{low} -rule since the instantiated constraint $1 > -1$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid. Similarly, $\mathbf{high}(1, \{-1\}) \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}} \emptyset$ using the third \mathbf{high} -rule. Continuing the reduction of t eventually yields $\mathbf{cons}(-1, \mathbf{cons}(1, \mathbf{nil}))$. \triangle

Notice an important consequence of Definition 3.15: $s|_p \xrightarrow{>\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} l\sigma \circ \gtrsim_{\mathcal{E}}^{\Lambda} l\sigma$ implies that $l\sigma$ is irreducible by $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ since $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ is strongly \mathcal{E} -coherent. It is important to keep this observation in mind when modeling algorithms, but as motivated above, this closely corresponds to the assumptions typically employed in the development of algorithms. As illustrated in the following example, this property makes it possible to use pattern matching in the definition of rewrite rules in an intuitive way, thus simplifying the task of modeling algorithms tremendously.

Example 3.17. Consider the example of a function $|\cdot|$ that computes the size of a set, where sets are modeled using the constructors \emptyset , $\{\cdot\}$, and \cup . Let \mathcal{S} and \mathcal{E} be as in Example 3.13. The three constructors for sets naturally give rise to a case analysis with three cases. For \emptyset and $\{\cdot\}$, computing the size of a set is simple:

$$\begin{aligned} |\emptyset| &\rightarrow 0 \\ |\{x\}| &\rightarrow 1 \end{aligned}$$

For the constructor \cup , it is quite intuitive to assume that $x \cup y$ is not the empty set or a singleton set, since both of these cases have already been taken care of. Furthermore, it can be assumed that both x and y are non-empty sets. Finally, computing the size of $x \cup y$ is very easy if the sets x and y are disjoint and much more complicated otherwise. If it can be assumed that x and y are disjoint sets, then

$$|x \cup y| \rightarrow |x| + |y|$$

computes the correct size of a set since the instantiations of the variables x and y will be disjoint non-empty sets due to the normalization with $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$. \triangle

The function symbols occurring at the root position of left-hand sides in \mathcal{R} are of particular interest since they are the only ones that make it possible to apply a rewrite rule. Thus, the following notation is introduced.

Definition 3.18 (Defined Symbols, Constructors). *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CERS. Then the defined symbols of $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ are given by $\mathcal{D}(\mathcal{R}) = \{f \mid f = \text{root}(l) \text{ for some } l \rightarrow r[\varphi] \in \mathcal{R}\}$. The constructors of $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ are $\mathcal{C}(\mathcal{R}) := \mathcal{F} \setminus \mathcal{D}(\mathcal{R})$.*

Notice that the function symbols from \mathcal{F}_{Th} are not members of $\mathcal{C}(\mathcal{R})$.

Example 3.19. In Example 3.13, $\mathcal{D}(\mathcal{R}) = \{\text{app}, \text{low}, \text{high}, \text{qsort}\}$ and $\mathcal{C}(\mathcal{R}) = \{\text{nil}, \text{cons}, \emptyset, \{\cdot\}, \cup\}$. \triangle

It is not immediately obvious whether the rewrite relation $\xrightarrow{\mathcal{S}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}}$ is decidable. In the important case where \mathcal{E} is size-preserving, the following positive answer is obtained. Notice that the canonizable theories from Figure 3.1 and the canonizable collection data structures from Figure 3.2 satisfy this requirement on \mathcal{E} .

Lemma 3.20. *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CERS such that \mathcal{E} is size-preserving and validity of \mathcal{Th} -constraints is decidable.*

1. *If s and t are terms, then it is decidable whether $s \sim_{\mathcal{E}} t$. Furthermore, the \mathcal{E} -equivalence class of s can be computed effectively.*
2. *For any term s , it is decidable whether s is reducible by $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$, and if so, a term t with $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}} t$ is effectively computable.*
3. *For any term s , it is decidable whether s is reducible by $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$, and if so, a term t with $s \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} t$ is effectively computable.*

3.4 Innermost and Restricted Rewriting

As argued in Chapter 1, it is often important to give special attention to the innermost rewriting strategy since this corresponds to the call-by-value semantics of eager functional programming languages. The innermost rewrite relation for CERSs is obtained by the following slight modification of Definition 3.15. The only added condition is \mathfrak{I} , which requires that all proper subterms of the instantiated left-hand side of the rewrite rule are irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$, i.e., the reduction indeed takes place at an innermost position where a reduction is possible.

Definition 3.21 (Innermost Rewrite Relation of a CERS). *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CERS. Then $s \xrightarrow{\mathcal{S}, \mathfrak{I}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} t$ iff there exist a constrained rewrite rule $l \rightarrow r[\varphi] \in \mathcal{R}$, a position $p \in \mathcal{Pos}(s)$, and a \mathcal{Th} -based substitution σ such that*

1. $s|_p \xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}} l\sigma \circ \xrightarrow{\geq \Lambda}_{\mathcal{E}} l\sigma$,
2. $\varphi\sigma$ is \mathcal{Th} -valid,
3. all proper subterms of $l\sigma$ are irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$, and
4. $t = s[r\sigma]_p$.

Example 3.22. In Example 3.13, $\text{app}(\text{nil}, \text{qsort}(\{1\})) \xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}} \text{qsort}(\{1\})$, but this is not allowed using Definition 3.21 since the proper subterm $\text{qsort}(\{1\})$ is reducible by $\xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}}$. Indeed, the only innermost reduction step of $\text{app}(\text{nil}, \text{qsort}(\{1\}))$ is $\text{app}(\text{nil}, \text{qsort}(\{1\})) \xrightarrow{\mathcal{S},i}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}} \text{app}(\text{nil}, \text{cons}(1, \text{nil}))$. \triangle

For termination purposes, the innermost rewrite relation might be terminating even though the full rewrite relation is non-terminating. This is well-known from ordinary term rewriting.¹ For certain classes of ordinary TRSs, however, innermost termination and full termination coincide [91]. Whether a similar relationship is also true for CERSs is currently open and could be investigated in future work.

Notice that actual implementations of rewriting most likely use a slightly different definition of innermost rewriting that checks whether $s|_p$ is irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}}$, i.e., condition \mathcal{I} in Definition 3.21 would be replaced by

\mathcal{I}' . All proper subterms of $s|_p$ are irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}}$.

Using Lemma 3.26 stated below, this results in a more restricted rewrite relation. However, the techniques presented in the remainder of this dissertation for the innermost case only make use of condition \mathcal{I} and cannot take any additional advantage of condition \mathcal{I}' . Notice that conditions \mathcal{I} and \mathcal{I}' coincide if $\mathcal{E} = \mathcal{S} = \emptyset$, i.e., for ordinary rewriting.

Example 3.23. This example illustrates the distinction between conditions \mathcal{I} and \mathcal{I}' . For this, \mathcal{S} and \mathcal{E} only model integers and \mathcal{R} consists of the single rule $\mathbf{f}(x) \rightarrow x$. Consider the term $t = \mathbf{f}(\mathbf{f}(0) + (-\mathbf{f}(0)))$. Using condition \mathcal{I} , $t \xrightarrow{\mathcal{S},i}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}} 0$ since $t \xrightarrow{\geq\Lambda}_1_{\mathcal{E}\setminus\mathcal{S}} \mathbf{f}(0) \xrightarrow{\geq\Lambda}_{\mathcal{E}} \mathbf{f}(x)\sigma$ for $\sigma = \{x \mapsto 0\}$ and all proper subterms of $\mathbf{f}(x)\sigma = \mathbf{f}(0)$ are irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}}$. In contrast, the only one-step reductions of t using condition \mathcal{I}' yield $\mathbf{f}(0 + (-\mathbf{f}(0)))$ and $\mathbf{f}(\mathbf{f}(0) + (-0))$ since the inner occurrences of \mathbf{f} need to be reduced first. \triangle

¹Recall Example 1.7.

Chapter 3. Constrained Equational Rewrite Systems

In order to obtain a form of rewriting that subsumes both the unrestricted rewrite relation considered in Definition 3.15 and the innermost rewrite relation of Definition 3.21, a general restricted rewrite relation is introduced, following an idea from [86] for ordinary rewriting.

Definition 3.24 (*Q-Restricted Rewrite Relation of a CERS*). *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CERS and let \mathcal{Q} be a finite set of constrained rewrite rules. Then $s \xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}} t$ iff there exist a constrained rewrite rule $l \rightarrow r[\varphi] \in \mathcal{R}$, a position $p \in \mathcal{P}os(s)$, and a Th-based substitution σ such that*

1. $s|_p \xrightarrow{\geq \Delta}_{\mathcal{E} \setminus \mathcal{S}} l \sigma \circ \approx_{\mathcal{E}}^{\Delta} l \sigma$,
2. $\varphi \sigma$ is Th-valid,
3. all proper subterms of $l \sigma$ are irreducible by $\xrightarrow{\mathcal{S}}_{Th \parallel \mathcal{E} \setminus \mathcal{Q}}$, and
4. $t = s[r\sigma]_p$.

The unrestricted rewrite relation is obtained from this definition by letting $\mathcal{Q} = \emptyset$, and the innermost rewriting is obtained if $\mathcal{Q} = \mathcal{R}$. Notice that there is in general no assumption on the relationship between \mathcal{Q} and \mathcal{R} . By combining a CERS with a set \mathcal{Q} as considered in Definition 3.24, the following generalization of CERSs is obtained.

Definition 3.25 (*Restricted Constrained Equational Rewrite Systems (RCERSs)*). *A restricted constrained equational rewrite system (RCERS) $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ consists of a finite set \mathcal{Q} of constrained rewrite rules and a CERS $(\mathcal{R}, \mathcal{S}, \mathcal{E})$.*

The following lemma collects several properties of RCERSs that are needed in the remainder of this dissertation. In particular, notice that these properties imply that, as stated above, condition 3 in Definition 3.21 is a slightly looser restriction than condition 3'. This can be seen by letting $\mathcal{Q} = \emptyset$ and observing that all proper

Chapter 3. Constrained Equational Rewrite Systems

subterms of $l\sigma$ are irreducible by $\xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{R}}$ if all proper subterms of $s|_p$ are irreducible by $\xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{R}}$ and $s|_p \xrightarrow{>\Lambda}_1 \mathcal{E}\setminus\mathcal{S} \circ \xrightarrow{>\Lambda}_1 l\sigma$.

Lemma 3.26. *Let $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be an RCERS.*

1. $\sim_{\mathcal{E}} \circ \xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} \subseteq \xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} \circ \sim_{\mathcal{E}}$, where the $\xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}$ -steps can be performed using the same constrained rewrite rule and Th -based substitution.
2. $\rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} \subseteq \xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}^+ \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}}$

The following corollary is easily obtained from this lemma.

Corollary 3.27. *Let $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be an RCERS and let s, t be terms.*

1. If $s \sim_{\mathcal{E}} t$, then s starts an infinite $\xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}$ -reduction iff t starts an infinite $\xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}$ -reduction.
2. If $s \rightarrow_{\mathcal{E}\setminus\mathcal{S}} t$ and t starts an infinite $\xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}$ -reduction, then s starts an infinite $\xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}$ -reduction.

Example 3.28. This example shows that right-linearity of \mathcal{S} is crucial for Corollary 3.27.2 and Lemma 3.26.2. Consider the following RCERS with $\mathcal{Q} = \mathcal{E} = \emptyset$:

$$\begin{array}{lcl} \mathcal{S} : & f(x) & \rightarrow g(x, x) \\ \mathcal{R} : & h(g(a, b)) & \rightarrow h(g(a, b)) \\ & c & \rightarrow a \\ & c & \rightarrow b \end{array}$$

Consider the term $t = h(f(c))$. It is irreducible by $\xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}$ at the root position since $t \xrightarrow{>\Lambda}_1 \mathcal{E}\setminus\mathcal{S} h(g(c, c))$ and $h(g(c, c))$ is not matched by the h -rule. The only one-step reductions of t below the root yield $h(f(a))$ and $h(f(b))$, both of which are irreducible by $\xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}$. But $t \xrightarrow{>\Lambda}_1 \mathcal{E}\setminus\mathcal{S} h(g(c, c))$, where $h(g(c, c))$ starts the infinite $\xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}$ -reduction $h(g(c, c)) \xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} h(g(a, c)) \xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} h(g(a, b)) \xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} h(g(a, b)) \xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} \dots$ △

3.5 Summary

This chapter has introduced the class of rewrite systems that is used for modeling algorithms in this dissertation. The key idea for modeling built-in numbers and collection data structures is to use equational completion [101, 21] in order to obtain a characterization of the defining properties of these data structures using both rewrite rules and equations. The rewrite relation of the defined symbols as specified using a set \mathcal{R} of constrained rewrite rules then utilizes this characterization by using the idea of normalized rewriting [128] before applying a rule from \mathcal{R} .

An innermost rewriting relation for \mathcal{R} can easily be defined by requiring that all proper subterms of the instantiated left-hand side of the rewrite rule are irreducible. This innermost rewrite relation naturally extends to a more general restricted rewrite relation that checks for irreducibility of the proper subterms of the instantiated left-hand side of the rewrite rule using a separate set \mathcal{Q} of rewrite rules. This general restricted rewrite relation encompasses both the full and the innermost rewrite relation of \mathcal{R} and enjoys “good” semantical properties.

Chapter 4

Translating Imperative Programs

Methods for automatically proving termination of imperative programs have received increased attention in recent years. The most commonly used automatic method for this is based on linear ranking functions which linearly combine the values of the program variables in a given state [44, 45, 144, 145, 36]. It was shown in [164, 38] that termination of a simple class of linear programs consisting of a single `while`-loop that does not contain any `if`-statements is decidable. More recently, the combination of abstraction refinement and linear ranking functions has been considered [49, 50, 42]. The tool `Terminator` [51], developed at Microsoft Research and based on this idea, has reportedly been used for showing termination of device drivers.

In order to show that the CERSs as introduced in Chapter 3 are widely applicable for a variety of different tasks, this brief chapter introduces a simple translation of a class of imperative programs into CERSs operating on built-in integers. Then, termination of the CERS implies termination of the imperative program.

Example 4.1. Consider the following imperative program:

```
while (x >= y) {  
    y++  
}
```

It is translated into the constrained rewrite rule $\text{eval}(x, y) \rightarrow \text{eval}(x, y + 1) \llbracket x \geq y \rrbracket$ that simulates the state change occurring during a single execution of the loop body. Notice that the constraint of the rule is obtained from the condition of the loop. \triangle

Using this simple translation, the methods for proving termination of CERSs that are developed in this dissertation can be applied for proving termination of imperative programs as well. The proposed translation should only be considered as a proof of concept. In order to be applicable to full-fledged imperative programming languages such as C or Java, further research is needed.

The idea of (automatically) translating programs from one programming language into another programming language for the purpose of program verification is not new. Translations from imperative programs into functional programs in the context of partial correctness proving have been described in [132, 78]. More closely related to the approach presented in this chapter, translations from real-life programming languages into term rewrite systems for the purpose of termination proving have been applied to the declarative programming languages Prolog [153] and Haskell [85], and [155, 138] contains initial results on translating a fragment of Java into term rewrite systems.

4.1 A Simple Imperative Language

Consider a simple imperative programming language where programs are formed according to the grammar in Figure 4.1. The constructs in this programming language have the standard (operational) semantics, i.e., `skip` denotes a do-nothing statement

| | | | |
|---------------------|------------------|---|---|
| <code>prog</code> | <code>::=</code> | <code>stmt</code> | |
| | | | <code>assume; stmt</code> |
| <code>stmt</code> | <code>::=</code> | <code>skip</code> | |
| | | | <code>assgn</code> |
| | | | <code>stmt; stmt</code> |
| | | | <code>if (cond) {stmt} else {stmt}</code> |
| | | | <code>while (cond) {stmt}</code> |
| | | | <code>break</code> |
| | | | <code>continue</code> |
| | | | <code>either {stmt} or {stmt}</code> |
| <code>assume</code> | <code>::=</code> | <code>assume cond</code> | |
| <code>cond</code> | <code>::=</code> | <code>"$\mathcal{T}h_{\mathbb{Z}}$-constraints"</code> | |
| <code>assgn</code> | <code>::=</code> | <code>(var₁, ..., var_k) := (exp₁, ..., exp_k)</code> | for some $k \geq 1$ |
| <code>var</code> | <code>::=</code> | <code>"variable names"</code> | |
| <code>exp</code> | <code>::=</code> | <code>"linear arithmetic expressions with + and -"</code> | |

Figure 4.1: Grammar for a simple imperative programming language.

and the `either`-statement denotes a non-deterministic choice. The `break`-statement aborts execution of the innermost `while`-loop surrounding it, while the `continue`-statement just aborts the current iteration of that loop and immediately starts the next iteration. The “;” in a concatenation may be omitted if the first statement ends with a “}”. The `assume`-statement is used to state preconditions of the program. The $\mathcal{T}h_{\mathbb{Z}}$ -constraints for `cond` usually only use conjunction (written `&&` in programs), disjunction (written `||` in programs), and negation (written `!` in programs). It is assumed that every parallel assignment contains each variable of the program exactly once on its left-hand side. This can always be achieved by adding dummy assignments that do not change a variable. Furthermore, it is assumed that each parallel assignment statement contains the variables of the program in the same fixed order on its left-hand side. A parallel assignment statement of the form $(x_1, \dots, x_k) := (e_1, \dots, e_k)$ where $e_i \neq x_i$ for exactly one i is also written as $x_i := e_i$. Finally, `x++` is an abbreviation for the assignment `x := x + 1` and `x--` abbreviates `x := x - 1`.

4.2 Translating Imperative Programs into CERSs

The translation of an imperative program into a CERS proceeds as follows. Notice that this is particularly simple since the conditions used in `if`-statements and `while`-loops are identical to the constraints allowed for CERSs.

Assume that the imperative program uses the variables $\mathbf{x}_1, \dots, \mathbf{x}_n$ and contains m control points (i.e., program entry, `while`-loops and `if`-statements¹). Then the i^{th} control point in the program is associated with a function symbol $\text{eval}_i : \text{int} \times \dots \times \text{int} \rightarrow \text{univ}$ with n arguments, where `univ` is a sort distinct from `int`. For simplicity of presentation and without loss of generality it is assumed that each straight-line code segment between two control points is a parallel assignment, `skip`, or empty. A sequence of parallel assignments can be combined into a single parallel assignment in order to satisfy this requirement.

For all $1 \leq i, j \leq m$ such that the j^{th} control point can be reached from the i^{th} control point by a straight-line code segment, each such straight-line code segment gives rise to a constrained rewrite rule of the form

$$\text{eval}_i(\dots) \rightarrow \text{eval}_j(\dots) \llbracket \varphi \rrbracket$$

where the constraint φ is determined as follows. If the i^{th} control point is the program entry, then φ is the condition of the `assume`-statement (if it exists) or \top . If the i^{th} control point is a `while`-statement, then φ is the condition of the `while`-loop or the negated condition of the `while`-loop, depending on whether the loop body is entered to reach the j^{th} control point. If the i^{th} control point is an `if`-statement, then φ is the condition of the `if`-statement or the negated condition of the `if`-statement, depending on whether the then-branch or the else-branch is taken to reach the j^{th} control point.²

¹For termination purposes it is not necessary to consider the program exit.

²It is also possible to merge the control point of an `if`-statement with control points

Chapter 4. Translating Imperative Programs

The constrained rewrite rule that is created now depends on the straight-line code segment.

Case 1: skip or empty. If the straight-line code segment is `skip` or empty, then the rewrite rule just becomes

$$\text{eval}_i(x_1, \dots, x_n) \rightarrow \text{eval}_j(x_1, \dots, x_n) \llbracket \varphi \rrbracket$$

with φ as described above.

Case 2: Parallel assignment. If the straight-line code segment is a parallel assignment $(\mathbf{x}_1, \dots, \mathbf{x}_k) := (e_1, \dots, e_k)$, then the rewrite rule becomes

$$\text{eval}_i(x_1, \dots, x_n) \rightarrow \text{eval}_j(e_1, \dots, e_n) \llbracket \varphi \rrbracket$$

with φ as described above.

Example 4.2. Consider the following imperative program:

```
while (x > 0 && y > 0) {
  if (x > y) {
    while (x > 0) {
      (x, y) := (x - 1, y + 1);
    }
  } else {
    while (y > 0) {
      (x, y) := (x + 1, y - 1);
    }
  }
}
```

from which the `if` statement can be reached. In this case φ is the conjunction of constraints obtained along that path.

Chapter 4. Translating Imperative Programs

It is translated into the following CERS:

$$\begin{aligned}\text{eval}_1(x, y) &\rightarrow \text{eval}_2(x, y) \llbracket x > 0 \wedge y > 0 \wedge x > y \rrbracket \\ \text{eval}_1(x, y) &\rightarrow \text{eval}_3(x, y) \llbracket x > 0 \wedge y > 0 \wedge x \not> y \rrbracket \\ \text{eval}_2(x, y) &\rightarrow \text{eval}_2(x - 1, y + 1) \llbracket x > 0 \rrbracket \\ \text{eval}_2(x, y) &\rightarrow \text{eval}_1(x, y) \llbracket x \not> 0 \rrbracket \\ \text{eval}_3(x, y) &\rightarrow \text{eval}_3(x + 1, y - 1) \llbracket y > 0 \rrbracket \\ \text{eval}_3(x, y) &\rightarrow \text{eval}_1(x, y) \llbracket y \not> 0 \rrbracket\end{aligned}$$

The outer `while`-loop is the first control point and the inner `while`-loops are the second and third control point, i.e., the technique of Footnote 2 has been used. \triangle

Correctness of the translation is based on the observation that any state transition of the imperative program can be simulated by a rewrite sequence.

Theorem 4.3. *Let P be an imperative program. Then the above translation produces a CERS $(\mathcal{R}_P, \mathcal{S}, \mathcal{E})$ where \mathcal{S} and \mathcal{E} are used to model $Th_{\mathbb{Z}}$ such that P is terminating if \mathcal{R}_P is terminating.*

Notice that the converse of this statement is not true in general, i.e., $(\mathcal{R}_P, \mathcal{S}, \mathcal{E})$ might be non-terminating although P is terminating. This clearly illustrates the limitations of the simple translation introduced in this chapter.

Example 4.4. Consider the following imperative program [145]:

```
while (x >= 0) {
  y := 1;
  while (x > y) {
    y := 2 * y
  }
  x--
}
```


The imperative program is translated into the following CERS:

$$\begin{aligned} \text{eval}_1(x, y) &\rightarrow \text{eval}_2(x, 1) \llbracket x \geq 0 \rrbracket \\ \text{eval}_2(x, y) &\rightarrow \text{eval}_2(x, 2 \cdot y) \llbracket x > y \rrbracket \\ \text{eval}_2(x, y) &\rightarrow \text{eval}_1(x - 1, y) \llbracket x \not> y \rrbracket \end{aligned}$$

The imperative program is terminating since the inner `while`-loop increases y until it exceeds the upper bound x and the outer `while`-loop decreases x . The resulting CERS, however, is not terminating since $\text{eval}_2(1, 0) \xrightarrow{\mathcal{S}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}} \text{eval}_2(1, 2 \cdot 0) \xrightarrow{\mathcal{S}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}} \text{eval}_2(1, 2 \cdot 2 \cdot 0) \xrightarrow{\mathcal{S}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}} \dots \quad \triangle$

The problem in Example 4.4 is that the control flow is not suitably taken into account since it is not determined that y is set to 1 before the inner `while`-loop is entered. Using static program analysis, this (and further information on the program) can be determined automatically.

4.3 Utilizing Static Program Analysis

In some cases, a successful automatic termination proof requires simple invariants on the program variables (such as “a variable is always non-negative”) or simple reasoning of the kind “if variables do not change between control points, then relations that are true for them at the first control point are still true at the second control point”. This kind of information can be obtained automatically using static program analysis tools such as `Interproc`³. The program analysis tool `Interproc` is based on the abstract interpretation framework [52] in combination with the interval [52], polyhedra [53], or octagon [133] domain and automatically performs the kind of reasoning discussed above.

³Freely available from <http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/interproc/>

Chapter 4. Translating Imperative Programs

In examples within this dissertation, straight-line code segments are annotated with the information discovered by the static program analysis. These annotations are indicated in the program by writing a corresponding $\mathcal{Th}_{\mathbb{Z}}$ -constraint of the form $[\psi]$ before that straight-line code segment. The translation of Section 4.2 can utilize this information by replacing the constraint φ obtained from the conditions of **while**-loops and **if**-statements as discussed above by $\varphi \wedge \psi$.

Example 4.5. Using the interval domain [52], the following annotations are obtained for the imperative program from Example 4.4:

```

while (x >= 0) {
    [ x >= 0 ]
    y := 1;
    while (x > y) {
        [ x >= 0 && y > 0 ]
        y := 2 * y
    }
    [ x >= 0 && y > 0 ]
    x--
}

```

Using these annotations, the imperative program is translated into the following CERS:

$$\begin{aligned}
 \text{eval}_1(x, y) &\rightarrow \text{eval}_2(x, 1) \llbracket x \geq 0 \rrbracket \\
 \text{eval}_2(x, y) &\rightarrow \text{eval}_2(x, 2 \cdot y) \llbracket x \geq 0 \wedge y > 0 \wedge x > y \rrbracket \\
 \text{eval}_2(x, y) &\rightarrow \text{eval}_1(x - 1, y) \llbracket x \geq 0 \wedge y > 0 \wedge x \not> y \rrbracket
 \end{aligned}$$

It is shown in Example 7.29 how the annotations are used for a successful termination proof. △

4.4 Summary

This chapter has introduced a simple translation of a class of imperative programs into CERSs operating on built-in integers. This translation is sound for termination purposes (i.e., the imperative program is terminating whenever the CERS obtained from it is terminating) but not complete in general. Often, an automatically performed static program analysis is helpful in cases where the translation is incomplete for termination purposes.

The purpose of this translation is to make the methods for proving termination of CERSs developed in this dissertation applicable for proving termination of imperative programs as well. The class of imperative programs considered in this chapter is restricted. Further research is needed in order to broaden this class to come closer to full-fledged imperative languages such as `C` or `Java`. Nonetheless, the translation presented in this chapter in combination with the methods for proving termination of CERSs presented in subsequent chapters are already sufficient for showing termination of most examples considered in the literature on termination analysis of imperative programs [44, 45, 145, 36, 37, 49, 50]. Notice that an empirical comparison of the methods presented in this dissertation and the methods of [44, 45, 145, 36, 37, 49, 50] is not possible since implementations of those methods are not publicly available.

Chapter 5

The Dependency Pair Framework

The dependency pair method [12] is commonly considered to be the most successful automated method for proving termination of ordinary rewriting. The crucial idea of the dependency pairs method is to compare the left-hand sides of rules with recursive calls to defined symbols on the right-hand sides of rules. The main theorem of the dependency pair method then states that an ordinary TRS is terminating if and only if it is impossible to build infinite chains using these recursive calls. This general idea has been extended from ordinary rewriting to rewriting modulo associativity and commutativity [129, 118], rewriting modulo i.u.v. collapse-free sets of equations [81], and ordinary context-sensitive rewriting [2, 1].

The most recent formulation of this method is the *dependency pair framework* [86], a general framework for termination proving that makes it possible to modularly combine different techniques. Apart from a clear theoretical formulation, the dependency pair framework is also suitable for a modular implementation of automated termination provers such as AProVE [84].

While originally developed for ordinary rewriting, the dependency pair framework has recently been extended to rewriting modulo associativity and commutativ-

ity [156] and to ordinary context-sensitive rewriting [1]. This chapter extends the dependency pair framework to rewriting with RCERSs and thus, in particular, to full or innermost rewriting with CERSs.

For the purpose of termination analysis, it suffices to only consider one further sort in addition to the sort `base` of the built-in theory. Thus, it is assumed in Chapters 5–12 that the only sorts are `base` and a new sort `univ`. Every RCERS can easily be transformed into a new RCERS over these two sorts by identifying all sorts that are different from `base`. Clearly, the original RCERS is terminating if the RCERS with only two sorts is terminating.

5.1 Dependency Pairs

The definition of a dependency pair is essentially the well-known one [12], with the only difference that a dependency pair inherits the constraint of the rule it is created from. As customary, a signature \mathcal{F}^\sharp is introduced, containing the function symbol f^\sharp for each function symbol $f \in \mathcal{F}$, where f^\sharp has the sort declaration $s_1 \times \dots \times s_n \rightarrow \mathbf{top}$ if f has the sort declaration $s_1 \times \dots \times s_n \rightarrow s$ with $s \in \{\mathbf{base}, \mathbf{univ}\}$. Here, `top` is a new sort that is distinct from `base` and `univ`. For the term $t = f(t_1, \dots, t_n)$, the term $f^\sharp(t_1, \dots, t_n)$ is denoted by t^\sharp .

Definition 5.1 (Dependency Pairs). *Let $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be an RCERS. The dependency pairs of \mathcal{R} are $\text{DP}(\mathcal{R}) = \{l^\sharp \rightarrow t^\sharp[\varphi] \mid t \text{ is a subterm of } r \text{ with } \text{root}(t) \in \mathcal{D}(\mathcal{R}) \text{ for some } l \rightarrow r[\varphi] \in \mathcal{R}\}$.*

Example 5.2. Example 3.13 gives rise to the following dependency pairs:

$$\text{app}^\sharp(\text{cons}(x, ys), zs) \rightarrow \text{app}^\sharp(ys, zs) \quad (5.1)$$

$$\text{low}^\sharp(x, y \cup z) \rightarrow \text{low}^\sharp(x, y) \quad (5.2)$$

$$\text{low}^\sharp(x, y \cup z) \rightarrow \text{low}^\sharp(x, z) \quad (5.3)$$

$$\mathbf{high}^\sharp(x, y \cup z) \rightarrow \mathbf{high}^\sharp(x, y) \quad (5.4)$$

$$\mathbf{high}^\sharp(x, y \cup z) \rightarrow \mathbf{high}^\sharp(x, z) \quad (5.5)$$

$$\mathbf{qsort}^\sharp(\{x\} \cup y) \rightarrow \mathbf{app}^\sharp(\mathbf{qsort}(\mathbf{low}(x, y)), \mathbf{cons}(x, \mathbf{qsort}(\mathbf{high}(x, y)))) \quad (5.6)$$

$$\mathbf{qsort}^\sharp(\{x\} \cup y) \rightarrow \mathbf{qsort}^\sharp(\mathbf{low}(x, y)) \quad (5.7)$$

$$\mathbf{qsort}^\sharp(\{x\} \cup y) \rightarrow \mathbf{low}^\sharp(x, y) \quad (5.8)$$

$$\mathbf{qsort}^\sharp(\{x\} \cup y) \rightarrow \mathbf{qsort}^\sharp(\mathbf{high}(x, y)) \quad (5.9)$$

$$\mathbf{qsort}^\sharp(\{x\} \cup y) \rightarrow \mathbf{high}^\sharp(x, y) \quad (5.10)$$

Notice that a single constrained rewrite rule may give rise to more than one dependency pair. \triangle

In order to verify termination of an RCERS, the notion of *chains* is used. Intuitively, a dependency pair corresponds to a recursive call, and a chain represents a possible sequence of calls in a reduction w.r.t. $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$. In the following, it is always assumed that different (occurrences of) dependency pairs are variable-disjoint. Notice that application of a substitutions does not introduce occurrences of function symbols from \mathcal{F}^\sharp into terms from $\mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{Th}}, \mathcal{V})$ since the symbols from \mathcal{F}^\sharp have resulting sort **top**.

Definition 5.3 ((Minimal) $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -Chains). *Let \mathcal{P} be a set of dependency pairs and let $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be an RCERS. A (possibly infinite) sequence of dependency pairs $s_1 \rightarrow t_1 \llbracket \varphi_1 \rrbracket, s_2 \rightarrow t_2 \llbracket \varphi_2 \rrbracket, \dots$ from \mathcal{P} is an $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain iff there exists a \mathcal{Th} -based substitution σ such that $t_i \sigma \xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^* \circ \xrightarrow{> \Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \circ \xrightarrow{\gtrsim \Lambda}_{\mathcal{E}} s_{i+1} \sigma$, the \mathcal{Th} -constraint $\varphi_i \sigma$ is \mathcal{Th} -valid, and $s_i \sigma$ is irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{Q}}$ and $\xrightarrow{> \Lambda}_{\mathcal{E} \setminus \mathcal{S}}$ for all $i \geq 1$. The above $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain is minimal iff $t_i \sigma$ does not start an infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reduction for all $i \geq 1$.*

Here, $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^*$ corresponds to reductions occurring strictly below the root of $t_i \sigma$ (notice that $\text{root}(t_i) \in \mathcal{F}^\sharp$), and $\xrightarrow{> \Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \circ \xrightarrow{\gtrsim \Lambda}_{\mathcal{E}}$ corresponds to normalization and

matching before applying $s_{i+1} \rightarrow t_{i+1} \llbracket \varphi_{i+1} \rrbracket$ at the root position.

Example 5.4. Using the dependency pair (5.2) from Example 5.2 twice, the following chain is obtained:

$$\text{low}^\#(x, y \cup z) \rightarrow \text{low}^\#(x, y), \text{low}^\#(x', y' \cup z') \rightarrow \text{low}^\#(x', y')$$

To see that this is indeed a chain, it suffices to consider the substitution $\sigma = \{x \mapsto 1, y \mapsto \{1\} \cup \{-1\}, z \mapsto \{2\}, x' \mapsto 1, y' \mapsto \{1\}, z' \mapsto \{-1\}\}$. \triangle

Using chains, an exact characterization of termination can be obtained. The intuition for this result is as follows: If $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ is not terminating, then there exists a *minimal non-terminating term*, i.e., a term that starts an infinite reduction such that none of its strict subterms starts an infinite reduction.

Example 5.5. Consider the following ordinary TRS:

$$\begin{aligned} \mathbf{a} &\rightarrow \mathbf{b} \\ \mathbf{f}(x, \mathbf{b}) &\rightarrow \mathbf{g}(x) \\ \mathbf{g}(x) &\rightarrow \mathbf{g}(\mathbf{d}(x)) \end{aligned}$$

Then the term $\mathbf{g}(\mathbf{f}(\mathbf{a}, \mathbf{a}))$ starts the infinite reduction $\mathbf{g}(\mathbf{f}(\mathbf{a}, \mathbf{a})) \rightarrow_{\mathcal{R}} \mathbf{g}(\mathbf{d}(\mathbf{f}(\mathbf{a}, \mathbf{a}))) \rightarrow_{\mathcal{R}} \mathbf{g}(\mathbf{d}(\mathbf{d}(\mathbf{f}(\mathbf{a}, \mathbf{a})))) \rightarrow_{\mathcal{R}} \dots$, but $\mathbf{g}(\mathbf{f}(\mathbf{a}, \mathbf{a}))$ is not a minimal non-terminating term since its strict subterm $\mathbf{f}(\mathbf{a}, \mathbf{a})$ starts the infinite reduction $\mathbf{f}(\mathbf{a}, \mathbf{a}) \rightarrow_{\mathcal{R}} \mathbf{f}(\mathbf{a}, \mathbf{b}) \rightarrow_{\mathcal{R}} \mathbf{g}(\mathbf{a}) \rightarrow_{\mathcal{R}} \mathbf{g}(\mathbf{d}(\mathbf{a})) \rightarrow_{\mathcal{R}} \mathbf{g}(\mathbf{d}(\mathbf{d}(\mathbf{a})))$. Thus, $\mathbf{f}(\mathbf{a}, \mathbf{a})$ is a minimal non-terminating term since its only strict subterm \mathbf{a} does not start an infinite reduction. \triangle

For any minimal non-terminating term, a rewrite rule has to be applied at the root position eventually. Then the right-hand side of the rewrite rule that is used for the reduction at the root position contains a non-variable subterm which is instantiated to a minimal non-terminating term and $\text{DP}(\mathcal{R})$ contains a corresponding dependency pair. The same reasoning can then be applied to the instantiation of that non-variable subterm of the right-hand side.

Theorem 5.6. *Let $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be an RCERS. Then $\xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}$ is terminating if and only if there are no minimal infinite $(DP(\mathcal{R}), \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chains.*

5.2 DP Framework

Given Theorem 5.6, termination of an RCERS can be investigated by considering chains of dependency pairs. For ordinary rewriting, a large number of techniques has been developed to this extent (see, e.g., [86, 88, 95]). These techniques typically cannot show termination by themselves. Instead, they may remove some of the dependency pairs or they may decompose a set of dependency pairs into several independent sets of dependency pairs. Then, a successful termination proof consists of a successive application of such techniques. In order to show soundness of these techniques independently, and in order to freely combine them in a flexible manner in implementations like AProVE [84], the notions of *DP problems* and *DP processors* were introduced for ordinary rewriting in [86], giving rise to the *DP framework*. In this dissertation, these notions are applied to rewriting with RCERSs.

Definition 5.7 (DP Problems). *A DP problem is a tuple $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ such that \mathcal{P} is a finite set of dependency pairs and $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ is an RCERS.*

In order to show that rewriting with $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ is terminating, Theorem 5.6 implies that it suffices to show that there are no infinite minimal $(DP(\mathcal{R}), \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chains, where, $(DP(\mathcal{R}), \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ is the initial DP problem obtained from the RCERS $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$. In order to show that a DP problem does not give rise to infinite chains, it is transformed into a set of DP problems for which this property has to be shown instead. This transformation is done using *DP processors*.

Definition 5.8 ((Sound) DP Processors). *A DP processor is a function Proc that takes a DP problem as input and returns a finite set of DP problems as output.*

Proc is sound¹ iff for all DP problems $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ with an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain there is a DP problem $(\mathcal{P}', \mathcal{Q}', \mathcal{R}', \mathcal{S}', \mathcal{E}') \in \text{Proc}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ with an infinite minimal $(\mathcal{P}', \mathcal{Q}', \mathcal{R}', \mathcal{S}', \mathcal{E}')$ -chain.

Ideally, DP processors have the property that it is easier to show absence of infinite chains for the DP problems that are obtained as output than it is to show absence of infinite chains for the input DP problem. It is in general impossible to characterize this property of DP processors precisely, but the removal of (one or more) dependency pairs from a DP problem results in a DP problem for which it is at least not harder to show absence of infinite chains. Notice that $\text{Proc}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) = \{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$ (or, more generally, $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) \in \text{Proc}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$) is a legal behavior for a DP processor. This can be interpreted as a failure of Proc on its input and indicates that a different DP processor should be applied.

The main motivation for introducing the DP framework is to formally model the recursive nature of termination proving once it has been reduced to showing the absence of infinite chains. This recursive nature gives rise to the concept of *DP trees*.

Definition 5.9 (DP Trees). *For an RCERS $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$, a DP tree for $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ is a tree whose nodes are labelled with DP problems or “yes” such that the root is labelled with the DP problem $(\text{DP}(\mathcal{R}), \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$, all leaves are labelled with “yes”, and for every inner node labelled with the DP problem $(\mathcal{P}', \mathcal{Q}', \mathcal{R}', \mathcal{S}', \mathcal{E}')$, there exists a sound DP processor Proc satisfying one of the following conditions:*

- $\text{Proc}(\mathcal{P}', \mathcal{Q}', \mathcal{R}', \mathcal{S}', \mathcal{E}') = \emptyset$ and the node has just one child, labelled with “yes”.
- $\text{Proc}(\mathcal{P}', \mathcal{Q}', \mathcal{R}', \mathcal{S}', \mathcal{E}') \neq \emptyset$ and the children of the node are labelled with the DP problems in $\text{Proc}(\mathcal{P}', \mathcal{Q}', \mathcal{R}', \mathcal{S}', \mathcal{E}')$.

¹The dual of soundness, i.e., *completeness* [86], is only needed for proving non-termination. Proving non-termination of RCERSs is not considered in this dissertation, but the DP framework for RCERSs can easily be extended for this purpose.

Chapter 5. The Dependency Pair Framework

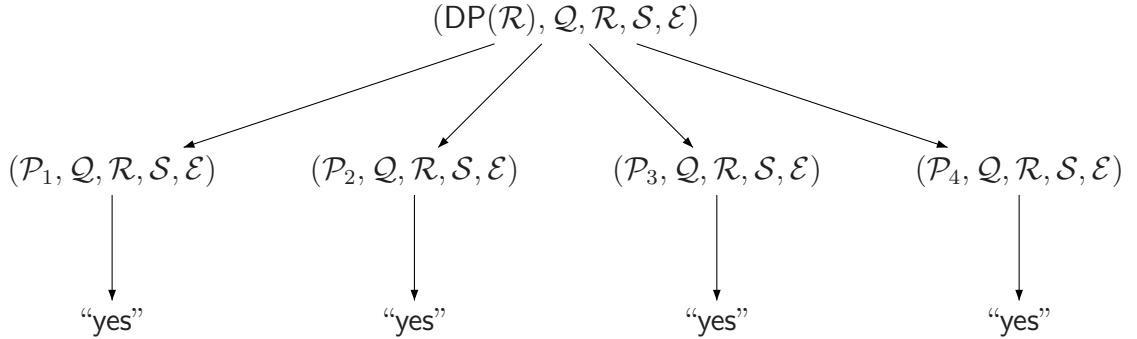
The following is immediate by Definition 5.8 and Theorem 5.6. Thus, the construction of a DP tree suffices for proving termination of an RCERS.

Corollary 5.10. *If there exists a DP tree for an RCERS $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$, then $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\text{Th}\|\mathcal{E}\setminus\mathcal{R}}$ is terminating.*

Example 5.11. This example illustrates the DP framework by giving the high-level structure of a termination proof for the quicksort RCERS from Example 5.2. For the sake of concreteness, let $\mathcal{Q} = \mathcal{R}$ (i.e., innermost termination is investigated) and consider the following DP processors:

1. Let Proc_1 be the DP processor introduced in Section 6.3. It is shown in Example 6.6 that $\text{Proc}_1(\text{DP}(\mathcal{R}), \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) = \{(\mathcal{P}_1, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}), (\mathcal{P}_2, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}), (\mathcal{P}_3, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}), (\mathcal{P}_4, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$ for some subsets $\mathcal{P}_i \subseteq \mathcal{P}$ of dependency pairs.
2. Let Proc_2 be the DP processor introduced in Section 6.6. Example 6.26 shows that $\text{Proc}_2(\mathcal{P}_1, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) = \text{Proc}_2(\mathcal{P}_2, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) = \text{Proc}_2(\mathcal{P}_3, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) = \emptyset$.
3. Let Proc_3 be the DP processor introduced in Section 8.1. As shown in Example 8.2, $\text{Proc}_3(\mathcal{P}_4, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) = \emptyset$.

Using these DP processors, the following DP tree can be constructed:



Therefore, rewriting with $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ is terminating. △

5.3 Summary

The dependency pair method [12] is commonly considered to be the most successful automated method for proving termination of ordinary rewriting, and the contributions of this chapter make it possible to apply the method to the expressive framework of RCERSs as well. For this, a precise characterization of termination of an RCERS using the absence of infinite chains of dependency pairs has been given.

Motivated by a clear theoretical foundation and a practical way to implement the dependency pair method, a general framework for termination proofs that makes it possible to modularly combine different techniques has been introduced, following the spirit of [86]. This framework is used as the basis for the methods presented in the next chapters, where various techniques for the termination analysis of RCERSs are presented in the form of DP processors.

Chapter 6

DP Processors Operating on Dependency Pairs

This chapter introduces various sound DP processors that consider the dependency pairs in \mathcal{P} and the constrained rewrite rules in \mathcal{R} independently of each other when operating on the DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$. Furthermore, most of these DP processors can disregard the constrained rewrite rules in \mathcal{R} altogether. The DP processors of Section 6.1 and Section 6.2 use basic properties of $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ and $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel_{\mathcal{E}} \setminus \mathcal{R}}$ in order to remove dependency pairs from a DP problem. Section 6.3 introduces the dependency graph, which makes it possible to decompose a DP problem into several independent DP problems by determining which dependency pairs may follow each other in chains. Using ideas developed for the dependency graph, Section 6.4 introduces a DP processor that can simplify the right-hand sides of dependency pairs. The DP processor in Section 6.5 can be used to combine dependency pairs that occur after each other in chains. Finally, the DP processor of Section 6.6 uses a specific well-founded relation (the subterm relation modulo \mathcal{E}) in order to remove dependency pairs from a DP problem.

While most of these DP processors are new contributions of this dissertation, the DP processors of Sections 6.3 and 6.6 are based on ideas used for ordinary TRSs [12, 87, 95]. An adaptation for RCERSs is non-trivial due to the presence of the sets \mathcal{S} and \mathcal{E} which need to be taken into consideration.

6.1 Unsatisfiable Constraints

Dependency pairs and rules may be deleted from a DP problem if they have a constraint that is \mathcal{Th} -unsatisfiable. This is sound since no instance of an unsatisfiable constraint is \mathcal{Th} -valid. The removal can also be performed at the level of RCERSs before the dependency pairs are computed.

Theorem 6.1 (DP Processor Based on Unsatisfiable Constraints). *Let Proc be a DP processor with $\text{Proc}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) = \{(\mathcal{P} - \mathcal{P}', \mathcal{Q} - \mathcal{Q}', \mathcal{R} - \mathcal{R}', \mathcal{S}, \mathcal{E})\}$, where*

- $\mathcal{P}' = \{s \rightarrow t[\varphi] \in \mathcal{P} \mid \varphi \text{ is } \mathcal{Th}\text{-unsatisfiable}\},$
- $\mathcal{Q}' = \{l \rightarrow r[\varphi] \in \mathcal{Q} \mid \varphi \text{ is } \mathcal{Th}\text{-unsatisfiable}\},$ and
- $\mathcal{R}' = \{l \rightarrow r[\varphi] \in \mathcal{R} \mid \varphi \text{ is } \mathcal{Th}\text{-unsatisfiable}\}.$

Then Proc is sound.

Example 6.2. This example is used to illustrate several simple DP processors. \mathcal{E} and \mathcal{S} are used to only model $\mathcal{Th}_{\mathbb{Z}}$ and sets built using \emptyset , $\{\cdot\}$, and \cup . The rewrite rules in \mathcal{R} are as follows:

$$f(x, y) \rightarrow f(x, y) \llbracket x > y \wedge y > x \rrbracket \quad (6.1)$$

$$f(x + (0 + (-x)), y) \rightarrow f(x + (0 + (-x)), y) \quad (6.2)$$

$$g(x \cup y) \rightarrow g(\emptyset \cup \emptyset) \quad (6.3)$$

In order to simplify presentation, the technique of Theorem 6.1 is applied to the RCERS $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ with $\mathcal{Q} = \emptyset$ and not on the level of DP problems. Then the RCERS $(\emptyset, \{(6.1), (6.2), (6.3)\}, \mathcal{S}, \mathcal{E})$ is transformed into $(\emptyset, \{(6.2), (6.3)\}, \mathcal{S}, \mathcal{E})$ because the constraint $x > y \wedge y > x$ is unsatisfiable in the integers. \triangle

6.2 Reducible Left-Hand Sides

If dependency pairs or rules have a left-hand side that is reducible by $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$, then these dependency pairs and rules may be deleted. This is sound since the instantiated left-hand side of a dependency pair in a chain or of a rule that is used for rewriting is irreducible by $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ according to Definitions 3.24 and 5.3. The removal can also be performed at the level of RCERSs before the dependency pairs are computed.

Theorem 6.3 (DP Processor Based on Reducible Left-Hand Sides). *Let Proc be a DP processor with $\text{Proc}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) = \{(\mathcal{P} - \mathcal{P}', \mathcal{Q} - \mathcal{Q}', \mathcal{R} - \mathcal{R}', \mathcal{S}, \mathcal{E})\}$, where*

- $\mathcal{P}' = \{s \rightarrow t[\varphi] \in \mathcal{P} \mid s \text{ is reducible by } \xrightarrow{>\Lambda}_{\mathcal{E} \setminus \mathcal{S}}\},$
- $\mathcal{Q}' = \{l \rightarrow r[\varphi] \in \mathcal{Q} \mid l \text{ is reducible by } \xrightarrow{>\Lambda}_{\mathcal{E} \setminus \mathcal{S}}\},$ and
- $\mathcal{R}' = \{l \rightarrow r[\varphi] \in \mathcal{R} \mid l \text{ is reducible by } \xrightarrow{>\Lambda}_{\mathcal{E} \setminus \mathcal{S}}\}.$

Then Proc is sound.

Example 6.4. Continuing Example 6.2, the RCERS $(\emptyset, \{(6.2), (6.3)\}, \mathcal{S}, \mathcal{E})$ is transformed into $(\emptyset, \{(6.3)\}, \mathcal{S}, \mathcal{E})$ since the left-hand side $f(x + (0 + (-x)), y)$ of (6.2) is reducible by $\xrightarrow{>\Lambda}_{\mathcal{E} \setminus \mathcal{S}}$ (for instance using the rule $x + 0 \rightarrow x$). \triangle

6.3 Dependency Graphs

The DP processor introduced in this section decomposes a DP problem into several independent DP problems by determining which dependency pairs from \mathcal{P} may follow each other in a $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain. This gives a restricted kind of modularity in termination proofs and may make it possible to disregard certain dependency pairs if they cannot appear in a chain more than once. The processor relies on the notion of *dependency graphs*, which are also used in the dependency pair method for ordinary rewriting [12].

Definition 6.5 (Dependency Graphs). *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be a DP problem. The nodes of its dependency graph $\text{DG}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ are the dependency pairs in \mathcal{P} and there is an arc from $s_1 \rightarrow t_1[\![\varphi_1]\!] to $s_2 \rightarrow t_2[\![\varphi_2]\!] iff $s_1 \rightarrow t_1[\![\varphi_1]\!]$, $s_2 \rightarrow t_2[\![\varphi_2]\!]$ is a $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain.$$*

Example 6.6. Continuing Example 5.2, recall the following dependency pairs:

$$\text{app}^\sharp(\text{cons}(x, ys), zs) \rightarrow \text{app}^\sharp(ys, zs) \quad (5.1)$$

$$\text{low}^\sharp(x, y \cup z) \rightarrow \text{low}^\sharp(x, y) \quad (5.2)$$

$$\text{low}^\sharp(x, y \cup z) \rightarrow \text{low}^\sharp(x, z) \quad (5.3)$$

$$\text{high}^\sharp(x, y \cup z) \rightarrow \text{high}^\sharp(x, y) \quad (5.4)$$

$$\text{high}^\sharp(x, y \cup z) \rightarrow \text{high}^\sharp(x, z) \quad (5.5)$$

$$\text{qsort}^\sharp(\{x\} \cup y) \rightarrow \text{app}^\sharp(\text{qsort}(\text{low}(x, y)), \text{cons}(x, \text{qsort}(\text{high}(x, y)))) \quad (5.6)$$

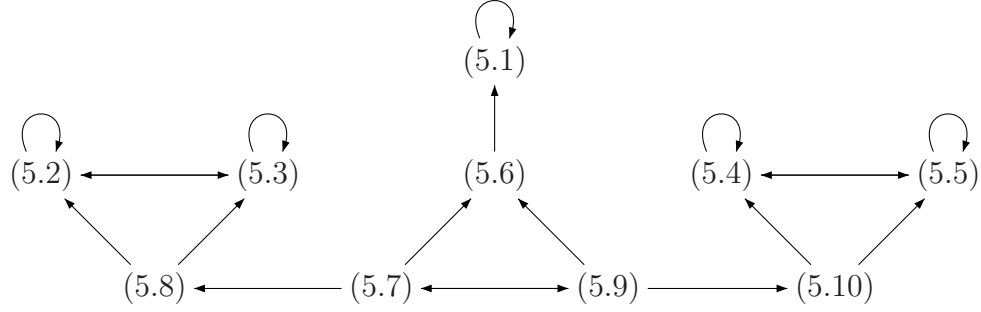
$$\text{qsort}^\sharp(\{x\} \cup y) \rightarrow \text{qsort}^\sharp(\text{low}(x, y)) \quad (5.7)$$

$$\text{qsort}^\sharp(\{x\} \cup y) \rightarrow \text{low}^\sharp(x, y) \quad (5.8)$$

$$\text{qsort}^\sharp(\{x\} \cup y) \rightarrow \text{qsort}^\sharp(\text{high}(x, y)) \quad (5.9)$$

$$\text{qsort}^\sharp(\{x\} \cup y) \rightarrow \text{high}^\sharp(x, y) \quad (5.10)$$

For these dependency pairs,



is the dependency graph. △

In general, $DG(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ cannot be computed exactly since it is undecidable whether two dependency pairs form a chain. Thus, an estimation has to be used instead. A possible estimation is given below.

A non-empty subset $\mathcal{P}' \subseteq \mathcal{P}$ of dependency pairs is a *strongly connected subgraph*¹ of $DG(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ iff for all dependency pairs $s_1 \rightarrow t_1[\varphi_1]$ and $s_2 \rightarrow t_2[\varphi_2]$ from \mathcal{P}' , there exists a path from $s_1 \rightarrow t_1[\varphi_1]$ to $s_2 \rightarrow t_2[\varphi_2]$ that only traverses dependency pairs from \mathcal{P}' . A strongly connected subgraph is a *strongly connected component (SCC)* if it is not a proper subset of any other strongly connected subgraph. Now every infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain has an infinite tail that stays within one strongly connected subgraph of $DG(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$, and it is thus sufficient to prove the absence of infinite chains for each SCC separately.

Theorem 6.7 (DP Processor Based on Dependency Graphs). *Let Proc be a DP processor with $\text{Proc}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) = \{(\mathcal{P}_1, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}), \dots, (\mathcal{P}_n, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, where $\mathcal{P}_1, \dots, \mathcal{P}_n$ are the SCCs in $DG(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$.² Then Proc is sound.*

Example 6.8. The dependency graph from Example 6.6 contains four SCCs, and according to Theorem 6.7, the following DP problems are obtained:

$$(\{(5.1)\}, \mathcal{R}, \mathcal{S}, \mathcal{E}) \tag{6.4}$$

¹Strongly connected subgraphs are often called *cycles* in the dependency pair literature. The name “strongly connected subgraph” stems from graph theory, where “cycle” has a different meaning.

²Notice, in particular, that $\text{Proc}(\emptyset, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) = \emptyset$.

$$(\{(5.2), (5.3)\}, \mathcal{R}, \mathcal{S}, \mathcal{E}) \quad (6.5)$$

$$(\{(5.4), (5.5)\}, \mathcal{R}, \mathcal{S}, \mathcal{E}) \quad (6.6)$$

$$(\{(5.7), (5.9)\}, \mathcal{R}, \mathcal{S}, \mathcal{E}) \quad (6.7)$$

These DP problems can be handled independently of each other. △

As mentioned above, $\text{DG}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ cannot be computed exactly in general since it is undecidable whether two dependency pairs $s_1 \rightarrow t_1 \llbracket \varphi_1 \rrbracket$ and $s_2 \rightarrow t_2 \llbracket \varphi_2 \rrbracket$ form a chain. Thus, an estimation has to be used instead. The idea for the estimation is that subterms of t_1 which might be instantiated to become reducible by $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ in a chain are abstracted by fresh variables. Then, it is checked whether the term obtained from t_1 in this way can be instantiated to reduce to an instantiation of s_2 using $\xrightarrow{\succ \Lambda}_{\mathcal{E} \setminus \mathcal{S}} \circ \xrightarrow{\succ \Lambda}_{\mathcal{E}}$. The function CAP is used to abstract subterms that might become reducible by $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$. Notice that a variable might be instantiated to become reducible by $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ in a chain only if it has sort `univ` and $\mathcal{Q} \not\subseteq \mathcal{R}$ (recall that the substitutions used for building chains are \mathcal{Th} -based and instantiate variables by terms that are irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{Q}}$). For terms whose root symbol is not a variable, CAP is applied recursively to the arguments and it is checked whether a rewrite rule might be applied at the root positions of the term that is obtained by the recursive application of CAP. For ordinary rewriting, a similar function CAP was introduced in [87]. In the following, a substitution μ is *Th-based for* $\mathcal{V}' \subseteq \mathcal{V}$ iff $\mu(x) \in \mathcal{T}(\mathcal{F}_{\mathcal{Th}}, \mathcal{V})$ for all $x \in \mathcal{V}'$ of sort `base`.

Definition 6.9 (Estimated Dependency Graphs). *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be a DP problem. Then its estimated dependency graph $\text{EDG}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ has the dependency pairs in \mathcal{P} as nodes and there is an arc from $s_1 \rightarrow t_1 \llbracket \varphi_1 \rrbracket$ to $s_2 \rightarrow t_2 \llbracket \varphi_2 \rrbracket$ iff there exists a substitution μ that is Th-based for $\mathcal{V}(s_1) \cup \mathcal{V}(s_2)$ such that $\text{CAP}(t_1)\mu \xrightarrow{\succ \Lambda}_{\mathcal{E} \setminus \mathcal{S}} \circ \xrightarrow{\succ \Lambda}_{\mathcal{E}} s_2\mu$, $\varphi_1\mu$ and $\varphi_2\mu$ are Th-valid, and $s_1\mu$ and $s_2\mu$ are irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{Q}}$ and $\xrightarrow{\succ \Lambda}_{\mathcal{E} \setminus \mathcal{S}}$. Here, CAP is defined by*

Chapter 6. DP Processors Operating on Dependency Pairs

1. for variables x of sort **base**, $\text{CAP}(x) = x$,
2. for variables x of sort **univ**, $\text{CAP}(x) = \begin{cases} x & \text{if } \mathcal{Q} \supseteq \mathcal{R} \\ y & \text{otherwise} \end{cases}$
3. $\text{CAP}(f(t_1, \dots, t_n)) = f(\text{CAP}(t_1), \dots, \text{CAP}(t_n))$ if there does not exist a rule $l \rightarrow r[\varphi] \in \mathcal{R}$ such that $f(\text{CAP}(t_1), \dots, \text{CAP}(t_n))\mu \xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}} \circ \overset{\geq \Lambda}{\sim}_{\mathcal{E}} l\mu$ for a substitution μ that is \mathcal{Th} -based for $\mathcal{V}(f(t_1, \dots, t_n)) \cup \mathcal{V}(l)$ where $\varphi\mu$ is \mathcal{Th} -valid and all proper subterms of $l\mu$ are irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{Q}}$, and
4. $\text{CAP}(f(t_1, \dots, t_n)) = y$ otherwise.

Here, y is the next variable in an infinite list y_1, y_2, \dots of fresh variables.

It is also possible to omit the checks for irreducibility by $\xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}}$, irreducibility by $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$, and \mathcal{Th} -validity, and it is possible to replace case 3 in the definition of CAP by a simple check for $f \notin \mathcal{D}(\mathcal{R})$. The following results remain correct for these possibilities. Implementing EDG is still hard in general since it has to be decided whether there exists a substitution μ such that $s\mu \xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}} \circ \overset{\geq \Lambda}{\sim}_{\mathcal{E}} t\mu$ for two terms s, t . A discussion of possible implementations can be found in Section 9.1.

Example 6.10. In order to check whether the estimated dependency graph contains an arc from (5.6) to (5.1) in Example 6.6, it needs to be checked whether there exists a substitution μ that is \mathcal{Th} -based for $\mathcal{V}(\text{qsort}^\sharp(\{x\} \cup y)) \cup \mathcal{V}(\text{app}^\sharp(\text{cons}(x', ys'), zs'))$ such that

$$\begin{aligned} & \text{CAP}(\text{app}^\sharp(\text{qsort}(\text{low}(x, y)), \text{cons}(x, \text{qsort}(\text{high}(x, y))))\mu \\ &= \text{app}^\sharp(y_1, \text{cons}(x, y_2))\mu \\ & \xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}} \circ \overset{\geq \Lambda}{\sim}_{\mathcal{E}} \text{app}^\sharp(\text{cons}(x', ys'), zs')\mu \end{aligned}$$

This can easily be seen to be the case by considering, e.g., the substitution $\mu = \{y_1 \mapsto \text{cons}(x', ys'), zs' \mapsto \text{cons}(x, y_2)\}$. Therefore, the estimated dependency graph contains an arc from (5.6) to (5.1). Repeating similar reasoning to all possible arcs, an

estimated dependency graph that is identical to the (exact) dependency graph from Example 6.6 is obtained. \triangle

It remains to be shown that the estimated dependency graph is indeed an overapproximation of the dependency graph, i.e., that $\text{EDG}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ is a supergraph of $\text{DG}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$.

Theorem 6.11 (Correctness of EDG). *For any DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$, the estimated dependency graph $\text{EDG}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ is a supergraph of the dependency graph $\text{DG}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$.*

6.4 Reducing Right-Hand Sides

The function CAP introduced in the previous section has a further use that is not related to estimated dependency graphs. Using CAP , it is possible to apply $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ to the right-hand side of a dependency pair in a certain way. More precisely, $\text{CAP}(t|_p)$ for a non-root position $p \in \text{Pos}(t)$ can be reduced by $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ for a dependency pair $s \rightarrow t[[\varphi]] \in \mathcal{P}$.

Theorem 6.12 (DP Processor Based on Reducing Right-Hand Sides of Dependency Pairs). *Let Proc be a DP processor with $\text{Proc}(\mathcal{P} \cup \{s \rightarrow t[[\varphi]]\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) =$*

- $\{(\mathcal{P} \cup \{s \rightarrow t'[[\varphi]]\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, *if there exists a non-root position $p \in \text{Pos}(t)$ such that $\text{CAP}(t|_p) \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \hat{t}$ and $t' = t[\hat{t}\tau]_p$, where τ is the substitution with $\text{CAP}(t|_p)\tau = t|_p$, and*
- $\{(\mathcal{P} \cup \{s \rightarrow t[[\varphi]]\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, *otherwise.*

Then Proc is sound.

Example 6.13. Continuing Example 6.4, the RCERS $(\mathcal{Q}, \{(6.3)\}, \mathcal{S}, \mathcal{E})$ with an arbitrary set \mathcal{Q} gives rise to the DP problem $(\{(6.8)\}, \mathcal{Q}, \{(6.3)\}, \mathcal{S}, \mathcal{E})$ with the following dependency pair:

$$\mathbf{g}^\sharp(x \cup y) \rightarrow \mathbf{g}^\sharp(\emptyset \cup \emptyset) \quad (6.8)$$

Applying the DP processor of Theorem 6.12 to the only dependency pair, the DP problem $(\{\mathbf{g}^\sharp(x \cup y) \rightarrow \mathbf{g}^\sharp(\emptyset)\}, \mathcal{Q}, \{(6.3)\}, \mathcal{S}, \mathcal{E})$ is obtained. The dependency graph of this DP problem does not contain any SCC (this is also true for the initial DP problem $(\{(6.8)\}, \mathcal{Q}, \{(6.3)\}, \mathcal{S}, \mathcal{E})$, but it is harder to determine this automatically, cf. Section 9.1). \triangle

6.5 Dependency Pair Narrowing

Under certain conditions it is possible to replace a dependency pair $s \rightarrow t[[\varphi]]$ by a set of new dependency pairs that are formed by combining $s \rightarrow t[[\varphi]]$ with all other dependency pairs that may follow it in chains.³ This way, it might be possible to obtain more information about the possible substitutions used for a chain since, in particular, the constraints of the dependency pairs are suitably combined.

Example 6.14. Consider the following rewrite rules \mathcal{R} , where \mathcal{S} and \mathcal{E} are used to model $\mathcal{Th}_{\mathcal{Z}}$:

$$f(x) \rightarrow f(-x + 1) \llbracket x > 0 \rrbracket$$

$$f(x) \rightarrow f(-x - 1) \llbracket x < 0 \rrbracket$$

³It is also possible to combine $s \rightarrow t[[\varphi]]$ with all dependency pairs that may *precede* it in chains. Notice that dependency pair narrowing has some similarities to the use of conditional constraints in [89]. Dependency pair narrowing conceptually differs from these conditional constraints, however, since conditional constraints are solely used in combination with one particular technique (reduction pairs). Dependency pair narrowing is independent of other techniques and transforms a DP problem. It is thus also similar to the dependency pair transformations from [12, 88].

Rewriting with these rules is terminating, since the absolute value of the argument of f is always decreasing. However, the implementation of the methods presented in this dissertation does not succeed in proving termination if the technique presented in this section is not used.

\mathcal{R} gives rise to the following dependency pairs:

$$f^\sharp(x) \rightarrow f^\sharp(-x + 1) \llbracket x > 0 \rrbracket \quad (6.9)$$

$$f^\sharp(x) \rightarrow f^\sharp(-x - 1) \llbracket x < 0 \rrbracket \quad (6.10)$$

By combining (6.9) with all dependency pairs that may follow it in chains (i.e., with (6.9) and (6.10)), the dependency pair (6.9) is replaced by

$$f^\sharp(x) \rightarrow f^\sharp(-(-x + 1) + 1) \llbracket x > 0 \wedge -x + 1 > 0 \rrbracket \quad (6.11)$$

$$f^\sharp(x) \rightarrow f^\sharp(-(-x + 1) - 1) \llbracket x > 0 \wedge -x + 1 < 0 \rrbracket \quad (6.12)$$

The implementation succeeds on the DP problem $(\{(6.10), (6.11), (6.12), \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}\})$ since the constraint of (6.11) is unsatisfiable and the DP problem $(\{6.12\}, \mathcal{Q}, \mathcal{E}, \mathcal{S}, \mathcal{E})$ obtained from the only SCC in the dependency graph can easily be handled by methods introduced in Chapter 7. \triangle

This idea can be stated in the form of a DP Processor. The conditions imposed on the left- and right-hand sides of the dependency pairs ensure that the combined dependency pairs can be easily computed.

Theorem 6.15 (DP Processor Based on Dependency Pair Narrowing). *Let Proc be a DP processor with $\text{Proc}(\mathcal{P} \cup \{s \rightarrow t \llbracket \varphi \rrbracket\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) =$*

- $\{(\mathcal{P} \cup \mathcal{P}', \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, if
 - $t = f^\sharp(t_1, \dots, t_n)$ with $t_i \in \mathcal{T}(\mathcal{F}_{\mathcal{T}h}, \mathcal{V})$ and $\text{sort}(t_i) = \mathbf{base}$ for all $1 \leq i \leq n$,
 - $s' = f^\sharp(x_1, \dots, x_n)$ for distinct variables x_1, \dots, x_n for all $s' \rightarrow t' \llbracket \varphi' \rrbracket \in \mathcal{P} \cup \{s \rightarrow t \llbracket \varphi \rrbracket\}$ with $\text{root}(s') = f^\sharp$, and

$$- \mathcal{P}' = \{s \rightarrow t'\tau[\varphi \wedge \varphi'\tau] \mid f^\sharp(x_1, \dots, x_n) \rightarrow t'[\varphi'] \in \mathcal{P} \cup \{s \rightarrow t[\varphi]\} \text{ and } \tau = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}\}.$$

- $\{(\mathcal{P} \cup \{s \rightarrow t[\varphi]\}), \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}\}$, otherwise.

Then Proc is sound.

If the right-hand side of the dependency pair $s \rightarrow t[\varphi]$ considered in Theorem 6.15 does not have the form $f(t_1, \dots, t_n)$ with $t_i \in \mathcal{T}(\mathcal{F}_{Th}, \mathcal{V})$ and $\text{sort}(t_i) = \text{base}$ for all $1 \leq i \leq n$, then it might be possible to eliminate certain arguments of f in order to satisfy this condition. The elimination of arguments is done using a simple version of *non-collapsing argument filters* [117].

Definition 6.16 (Non-Collapsing Argument Filters). A non-collapsing argument filter π maps every n -ary function symbol $f^\sharp \in \mathcal{F}^\sharp$ to a (possibly empty) list $[i_1, \dots, i_m]$ with $1 \leq i_1 < \dots < i_m \leq n$. The set $\pi(\mathcal{F}^\sharp)$ consists of all function symbols $f^\sharp \in \mathcal{F}^\sharp$, where f^\sharp now has arity m if $\pi(f) = [i_1, \dots, i_m]$. A non-collapsing argument filter π induces a mapping on terms t^\sharp defined by

$$\pi(t^\sharp) = f^\sharp(t_{i_1}, \dots, t_{i_m}) \quad \text{if } t^\sharp = f^\sharp(t_1, \dots, t_n) \text{ and } \pi(f^\sharp) = [i_1, \dots, i_m]$$

Now non-collapsing argument filters can be used to remove certain arguments in a DP problem. As motivated above, this may make it possible to apply the DP processor of Theorem 6.15 afterwards. A further use of the removal of arguments is presented in Section 7.3.

Theorem 6.17 (DP Processor Based on Non-Collapsing Argument Filters). Let π be a non-collapsing argument filter and let Proc be a DP processor such that $\text{Proc}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) =$

- $\{(\pi(\mathcal{P}), \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, if $\mathcal{V}(\pi(t)) \subseteq \mathcal{V}(\pi(s))$ for all $s \rightarrow t[\varphi] \in \mathcal{P}$. Here, $\pi(\mathcal{P}) = \{\pi(s) \rightarrow \pi(t)[\varphi] \mid s \rightarrow t[\varphi] \in \mathcal{P}\}$.⁴

⁴Notice that $\mathcal{V}(\varphi) \not\subseteq \mathcal{V}(s)$ is possible. This does not cause any complications.

- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, otherwise.

Then Proc is sound.

Example 6.18. With \mathcal{S} and \mathcal{E} as in Example 6.14, consider the following set \mathcal{R} of rewrite rules:

$$f(x, y) \rightarrow f(-x + 1, g(y)) \llbracket x > 0 \rrbracket \quad (6.13)$$

$$f(x, y) \rightarrow f(-x - 1, g(y)) \llbracket x < 0 \rrbracket \quad (6.14)$$

$$g(y) \rightarrow y \quad (6.15)$$

$$g(y) \rightarrow y + 17 \quad (6.16)$$

The dependency pairs obtained from these rewrite rules are

$$f^\sharp(x, y) \rightarrow f^\sharp(-x + 1, g(y)) \llbracket x > 0 \rrbracket \quad (6.17)$$

$$f^\sharp(x, y) \rightarrow f^\sharp(-x - 1, g(y)) \llbracket x < 0 \rrbracket \quad (6.18)$$

Notice that the DP processor of Theorem 6.15 is not applicable since the second arguments of f^\sharp on the right-hand sides contain recursive calls to g . By applying the non-collapsing argument filter $\pi(f^\sharp) = [1]$, these dependency pairs are transformed into the dependency pairs (6.9) and (6.10) from Example 6.14. \triangle

6.6 Subterm Criterion

The subterm criterion [95] is a relatively simple and efficient technique which is nonetheless surprisingly powerful. The technique works particularly well for functions that are defined using primitive recursion. For ordinary rewriting, the subterm criterion applies a *simple projection* which collapses a term $f^\sharp(t_1, \dots, t_n)$ to one of its direct subterms. Given a set \mathcal{P} of dependency pairs and a subset $\mathcal{P}' \subseteq \mathcal{P}$, the

subterm criterion consists of finding a simple projection such that the collapsed right-hand side is a subterm of the collapsed left-hand side for all dependency pairs in \mathcal{P} , where this subterm relation is furthermore strict for all dependency pairs from \mathcal{P}' . Then, the dependency pairs from \mathcal{P}' may be removed from the DP problem since they cannot occur infinitely often in a chain.

Definition 6.19 (Simple Projections). *A simple projection is a mapping π that assigns an argument position i with $1 \leq i \leq n$ to every $f^\# \in \mathcal{F}^\#$ with $\text{arity}(f^\#) = n$. The mapping that assigns the argument $t_{\pi(f^\#)}$ to any term $f^\#(t_1, \dots, t_n)$ is also denoted by π .*

In the context of this dissertation, the subterm relation modulo \mathcal{E} is used. This generalization is quite natural and makes it possible to apply the idea of the subterm criterion to RCERSs.

Definition 6.20 (\mathcal{E} -Subterms). *Let $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be an RCERS and let s, t be terms. Then t is a strict \mathcal{E} -subterm of s , written $s \triangleright_{\mathcal{E}} t$, iff $s \sim_{\mathcal{E}} \circ \triangleright \circ \sim_{\mathcal{E}} t$. The term t is an \mathcal{E} -subterm of s , written $s \triangleright_{\mathcal{E}} t$, iff $s \triangleright_{\mathcal{E}} t$ or $s \sim_{\mathcal{E}} t$.*

Example 6.21. For $\mathcal{E} = \{x \cup y \approx y \cup x, x \cup (y \cup z) \approx (x \cup y) \cup z\}$, $\{0\} \cup (\{1\} \cup \{2\}) \triangleright_{\mathcal{E}} \{2\} \cup \{0\}$ since $\{0\} \cup (\{1\} \cup \{2\}) \sim_{\mathcal{E}} \{1\} \cup (\{2\} \cup \{0\}) \triangleright \{2\} \cup \{0\}$. \triangle

Notice that $\triangleright_{\mathcal{E}}$ is not well-founded in general. Attention is thus restricted to the case where \mathcal{E} is size-preserving. In particular, this requirement is satisfied for the canonizable theories in Figure 3.1 and the canonizable collection data structures in Figure 3.2. The subterm criterion does not inherently depend on size-preservingness, however. Other criteria that ensure well-foundedness of $\triangleright_{\mathcal{E}}$ may be used as well, with size-preservingness being an easily checkable criterion that is sufficient for most cases.

It will be shown in Lemma 6.23 below that the subterm relation modulo \mathcal{E} is *stable*. This property is important since all instantiations of a dependency pair can be considered simultaneously by considering just the dependency pair itself.

Definition 6.22 (Stability). *A relation \bowtie on terms is stable iff $s \bowtie t$ implies $s\sigma \bowtie t\sigma$ for all terms s, t and all substitutions σ .*

The following lemma collects several properties of $\triangleright_{\mathcal{E}}$ and $\triangleright_{\mathcal{E}}$ in the case where \mathcal{E} is size-preserving. Here, only 1 and 2 depend on size-preservingness.

Lemma 6.23. *Let $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be an RCERS such that \mathcal{E} is size-preserving.*

1. *Given terms s, t , it is decidable whether $s \triangleright_{\mathcal{E}} t$ or $s \triangleright_{\mathcal{E}} t$.*
2. *$\triangleright_{\mathcal{E}}$ is well-founded.*
3. *$\triangleright_{\mathcal{E}}$ and $\triangleright_{\mathcal{E}}$ are stable.*
4. *$\triangleright_{\mathcal{E}}$ and $\triangleright_{\mathcal{E}}$ are compatible with $\sim_{\mathcal{E}}$, i.e., $\sim_{\mathcal{E}} \circ \triangleright_{\mathcal{E}} \circ \sim_{\mathcal{E}} \subseteq \triangleright_{\mathcal{E}}$ and $\sim_{\mathcal{E}} \circ \triangleright_{\mathcal{E}} \circ \sim_{\mathcal{E}} \subseteq \triangleright_{\mathcal{E}}$.*

A DP processor based on the subterm criterion is defined as follows. This DP processor has the advantage that it does not need to consider \mathcal{R} and \mathcal{S} in a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$. This makes it possible to handle many DP problems very efficiently.

Theorem 6.24 (DP Processor Based on the Subterm Criterion). *Let π be a simple projection and let Proc be a DP processor with $\text{Proc}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) =$*

- $\{(\mathcal{P} - \mathcal{P}', \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, if \mathcal{E} is size-preserving and $\mathcal{P}' \subseteq \mathcal{P}$ such that
 - $\pi(s) \triangleright_{\mathcal{E}} \pi(t)$ for all $s \rightarrow t[\![\varphi]\!] \in \mathcal{P}'$, and
 - $\pi(s) \triangleright_{\mathcal{E}} \pi(t)$ for all $s \rightarrow t[\![\varphi]\!] \in \mathcal{P} - \mathcal{P}'$.
- $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$, otherwise.

Then Proc is sound.

Recall from Lemma 6.23.1 that $\triangleright_{\mathcal{E}}$ and $\triangleright_{\mathcal{E}}$ are decidable if \mathcal{E} is size-preserving. An implementation thus reduces to finding the simple projection π .

Example 6.25. This example illustrates a mergesort algorithm that takes a set of integers and returns a sorted list of the elements of that set. For this, integers are modeled as in Figure 3.1 and sets are modeled using \emptyset , $\{\cdot\}$, and \cup as in Figure 3.2. The mergesort algorithm can be given as follows:

$$\begin{aligned} \text{merge}(\text{nil}, y) &\rightarrow y \\ \text{merge}(x, \text{nil}) &\rightarrow x \\ \text{merge}(\text{cons}(x, xs), \text{cons}(y, ys)) &\rightarrow \text{cons}(y, \text{merge}(\text{cons}(x, xs), ys)) \llbracket x > y \rrbracket \\ \text{merge}(\text{cons}(x, xs), \text{cons}(y, ys)) &\rightarrow \text{cons}(x, \text{merge}(xs, \text{cons}(y, ys))) \llbracket x \not> y \rrbracket \\ \text{msort}(\emptyset) &\rightarrow \text{nil} \\ \text{msort}(\{x\}) &\rightarrow \text{cons}(x, \text{nil}) \\ \text{msort}(x \cup y) &\rightarrow \text{merge}(\text{msort}(x), \text{msort}(y)) \end{aligned}$$

These rules give rise to the following five dependency pairs:

$$\text{merge}^{\sharp}(\text{cons}(x, xs), \text{cons}(y, ys)) \rightarrow \text{merge}^{\sharp}(\text{cons}(x, xs), ys) \llbracket x > y \rrbracket \quad (6.19)$$

$$\text{merge}^{\sharp}(\text{cons}(x, xs), \text{cons}(y, ys)) \rightarrow \text{merge}^{\sharp}(xs, \text{cons}(y, ys)) \llbracket x \not> y \rrbracket \quad (6.20)$$

$$\text{msort}^{\sharp}(x \cup y) \rightarrow \text{merge}^{\sharp}(\text{msort}(x), \text{msort}(y)) \quad (6.21)$$

$$\text{msort}^{\sharp}(x \cup y) \rightarrow \text{msort}^{\sharp}(x) \quad (6.22)$$

$$\text{msort}^{\sharp}(x \cup y) \rightarrow \text{msort}^{\sharp}(y) \quad (6.23)$$

The (estimated) dependency graph contains two SCCs and it suffices to consider the following DP problems independently:

$$(\{(6.19), (6.20)\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) \quad (6.24)$$

$$(\{(6.22), (6.23)\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) \quad (6.25)$$

For the DP problem (6.24), first apply the DP processor based on the subterm criterion with the simple projection $\pi(\text{merge}^\sharp) = 1$. Then

$$\begin{aligned} \pi(\text{merge}^\sharp(\text{cons}(x, xs), \text{cons}(y, ys))) &= \text{cons}(x, xs) \\ &\triangleright_{\mathcal{E}} \text{cons}(x, xs) = \pi(\text{merge}^\sharp(\text{cons}(x, xs), ys)) \\ \pi(\text{merge}^\sharp(\text{cons}(x, xs), \text{cons}(y, ys))) &= \text{cons}(x, xs) \\ &\triangleright_{\mathcal{E}} xs = \pi(\text{merge}^\sharp(xs, \text{cons}(y, ys))) \end{aligned}$$

and the dependency pair (6.20) may be deleted. The newly obtained DP problem $(\{(6.19)\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ can be handled by the subterm criterion with $\pi(\text{merge}^\sharp) = 2$.

The DP problem (6.25) can also be handled by the subterm criterion, using the simple projection $\pi(\text{msort}^\sharp) = 1$. △

Example 6.26. The DP problem (6.4) from Example 6.10 can be handled by the simple projection $\pi(\text{app}^\sharp) = 1$. Similarly, the simple projection $\pi(\text{low}^\sharp) = \pi(\text{high}^\sharp) = 2$ removes all dependency pairs from the DP problems (6.5) and (6.6). △

6.7 Summary

This chapter has introduced various sound DP processors for the termination analysis of RCERSs. First, it was shown that rewrite rules and dependency pairs may be removed from a DP problem if their constraint is \mathcal{Th} -unsatisfiable or if their left-hand side is reducible by $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$. Next, the concept of dependency graphs was introduced. While nearly all other DP processor introduced in this dissertation only remove rewrite rules and/or dependency pairs, this technique makes it possible to decompose a DP problem into several independent DP problems by determining which dependency pairs may follow each other in chains. The dependency graph cannot be implemented in its full generality, but it is possible to give effectively implementable approximations of it. Then, DP processors that reduce right-hand sides

Chapter 6. DP Processors Operating on Dependency Pairs

of dependency pairs and combine dependency pairs that follow each other in chains were introduced. Finally, it was shown how the subterm relation modulo \mathcal{E} can be used in order to remove dependency pairs from a DP problem. This is particularly useful for functions that are defined using primitive recursion.

The next chapter introduces several further sound DP processors. Like the DP processor based on the subterm criterion, these DP processors are based on well-founded relations. In contrast to the technique based on the subterm criterion, however, these DP processors are also useful for showing termination of functions that are not defined using primitive recursion.

Chapter 7

Reduction Pairs

The dependency pair framework for ordinary term rewriting makes heavy use of reduction pairs (\succsim, \succ) [117] in order to remove dependency pairs from a DP problem. The idea is simple: If \succ is well-founded and all dependency pairs from a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ are decreasing w.r.t. \succsim or \succ , then all dependency pairs that are decreasing w.r.t. \succ cannot occur infinitely often in infinite chains and may thus be deleted. In order to capture the reductions that take place between the instantiated dependency pairs, it is necessary to also require that all rules in \mathcal{R} are decreasing w.r.t. \succsim . For RCERSs, similar conditions need to be imposed on \mathcal{S} and \mathcal{E} as well.

The following sections present three variations on the theme of reduction pairs, starting with the ordinary reduction pairs from [117]. Then, reduction pairs tailored towards $\mathcal{Th}_{\mathbb{N}}$ and $\mathcal{Th}_{\mathbb{Z}}$ are introduced. In all three cases, the techniques can be applied to any RCERS, regardless of \mathcal{Q} .

Apart from the theoretical concepts, this chapter also introduces methods to obtain the aforementioned variations of reduction pairs using polynomial interpretations. A discussion on how to automatically create suitable polynomial interpretations is postponed until Section 9.2.

7.1 Ordinary Reduction Pairs

These are the well-known reduction pairs from [117]. They make strong requirements on the monotonicity of \succsim .

Definition 7.1 (Monotonicity). *A relation \bowtie on terms is monotonic iff $s \bowtie t$ implies $f(s_1, \dots, s_{i-1}, s, s_{i+1}, \dots, s_n) \bowtie f(s_1, \dots, s_{i-1}, t, s_{i+1}, \dots, s_n)$ for all function symbols f , all $1 \leq i \leq \text{arity}(f)$, and all terms $s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_n$.*

Notice that $s \bowtie t$ implies $C[s] \bowtie C[t]$ for all contexts C if \bowtie is monotonic.

Definition 7.2 (Ordinary Reduction Pairs). *Let \succsim be reflexive, transitive, monotonic, and stable. Let \succ be well-founded and stable. Then (\succsim, \succ) is an ordinary reduction pair iff \succ is compatible with \succsim , i.e., iff $\succsim \circ \succ \circ \succsim \subseteq \succ$. The relation $\succsim \cap \succsim^{-1}$ is denoted by \sim .*

As a simple example of compatibility, consider the usual relations $>$ and \geq on integers. Then $>$ is compatible with \geq since $a \geq b > c \geq d$ implies $a > d$ for all integers a, b, c, d . More abstractly, if \succ is compatible with \succsim , then preceding and succeeding occurrences of \succsim can be “absorbed” into \succ .

There is a variety of methods to generate reduction pairs: path orders [54], Knuth-Bendix orders [114], polynomial interpretations [119], matrix interpretations [63], Since it is impossible to give a broad survey of these methods, only polynomial interpretations are briefly discussed here because the reduction pairs tailored towards $\mathcal{Th}_{\mathbb{N}}$ and $\mathcal{Th}_{\mathbb{Z}}$ as introduced in Sections 7.2 and 7.3 are in practice most easily obtained using (more general) polynomial interpretations as well.¹

¹Reduction pairs tailored towards $\mathcal{Th}_{\mathbb{N}}$ and $\mathcal{Th}_{\mathbb{Z}}$ can also be obtained using matrix interpretations by adapting the ideas presented below for polynomial interpretations. This is not discussed further in order to keep the presentation simple. The implementation in AProVE *does* contain the specialized matrix interpretations, but they are rarely (if ever) needed in practice.

Chapter 7. Reduction Pairs

The idea for polynomial interpretations is to map terms to polynomials. Then, these polynomials are compared. In order to map terms to polynomials, it suffices to assign polynomials to all function symbols $f \in \mathcal{F}_{Th} \cup \mathcal{F} \cup \mathcal{F}^\sharp$ such that $\mathcal{P}ol(f) \in \mathbb{N}[x_1, \dots, x_n]$ if $\text{arity}(f) = n$. Now terms are mapped to polynomials by defining $[x]_{\mathcal{P}ol} = x$ for variables $x \in \mathcal{V}$ and $[f(t_1, \dots, t_n)]_{\mathcal{P}ol} = \mathcal{P}ol(f)([t_1]_{\mathcal{P}ol}, \dots, [t_n]_{\mathcal{P}ol})$ for $f \in \mathcal{F}_{Th} \cup \mathcal{F} \cup \mathcal{F}^\sharp$, i.e., the polynomials obtained from the direct subterms t_1, \dots, t_n are combined using the polynomial assigned to the root symbol. Notice that the variables in terms are translated into polynomial variables of the same name.

Example 7.3. If $\mathcal{P}ol(f) = 2x_1 + x_2$ and $\mathcal{P}ol(g) = x_1^2 + 1$ then $[g(f(x, y))]_{\mathcal{P}ol} = [f(x, y)]_{\mathcal{P}ol}^2 + 1 = (2x + y)^2 + 1 = 4x^2 + 4xy + y^2 + 1$. \triangle

In order to compare terms, the polynomials obtained from them are compared. For this, notice that ground terms are mapped to natural numbers.

Definition 7.4 ($\succ_{\mathcal{P}ol}$ and $\succeq_{\mathcal{P}ol}$ for Polynomial Interpretations). *Let $\mathcal{P}ol$ be a polynomial interpretation. Then $\succ_{\mathcal{P}ol}$ is defined by $s \succ_{\mathcal{P}ol} t$ if $[s\sigma]_{\mathcal{P}ol} > [t\sigma]_{\mathcal{P}ol}$ for all ground substitutions $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{Th})$. Similarly, $\succeq_{\mathcal{P}ol}$ is defined by $s \succeq_{\mathcal{P}ol} t$ if $[s\sigma]_{\mathcal{P}ol} \geq [t\sigma]_{\mathcal{P}ol}$ for all ground substitutions $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{Th})$. Thus, $s \sim_{\mathcal{P}ol} t$ if $[s\sigma]_{\mathcal{P}ol} = [t\sigma]_{\mathcal{P}ol}$ for all ground substitutions $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{Th})$.*

For a given polynomial interpretation, $s \succ_{\mathcal{P}ol} t$ is checked as follows. Recall that $s \succ_{\mathcal{P}ol} t$ means $[s\sigma]_{\mathcal{P}ol} > [t\sigma]_{\mathcal{P}ol}$ for all ground substitutions σ . This can be ensured by showing that

$$\forall x_1 \geq 0, \dots, x_n \geq 0. [s]_{\mathcal{P}ol} > [t]_{\mathcal{P}ol}$$

is true in the integers. Here x_1, \dots, x_n are the variables occurring in s and t . While it is in general undecidable whether this formula is true in the integers due to the undecidability of Hilbert's 10th problem, it becomes decidable if $\mathcal{P}ol(f)$ is linear for all $f \in \mathcal{F}_{Th} \cup \mathcal{F} \cup \mathcal{F}^\sharp$ since, in this case, $[s]_{\mathcal{P}ol}$ and $[t]_{\mathcal{P}ol}$ are linear polynomials as

well, thus implying that the above formula is decidable due to Presburger's classical result [146].

Example 7.5. If $\mathcal{P}ol(f) = 2x_1 + x_2$ and $\mathcal{P}ol(g) = 2x_1 + 1$, then $g(f(x, y)) \succ_{\mathcal{P}ol} f(x, y)$. To see this, notice that $[g(f(x, y))]_{\mathcal{P}ol} = 4x + 2y + 1$, $[f(x, y)]_{\mathcal{P}ol} = 2x + y$, and that $\forall x \geq 0, y \geq 0. 4x + 2y + 1 > 2x + y$ is true in the integers. \triangle

As is well-known, these relations indeed give rise to reduction pairs.

Theorem 7.6. *Let $\mathcal{P}ol$ be a polynomial interpretation. Then $(\succsim_{\mathcal{P}ol}, \succ_{\mathcal{P}ol})$ is an ordinary reduction pair.*

Since reduction pairs need to satisfy certain requirements derived from a DP problem, the following notation is introduced. Here, the set \mathcal{R}' will not be used until Section 8.2.

Definition 7.7. *Let (\succsim, \succ) be an ordinary reduction pair, let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be a DP problem, let \mathcal{P}' be a set of dependency pairs, and let \mathcal{R}' be a set of constrained rewrite rules. Then $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \mathcal{R}', \mathcal{R}, \mathcal{S}, \mathcal{E})$ iff*

1. $s \succ t$ for all $s \rightarrow t[[\varphi]] \in \mathcal{P}'$,
2. $s \succsim t$ for all $s \rightarrow t[[\varphi]] \in \mathcal{P} - \mathcal{P}'$,
3. $l \succ r$ for all $l \rightarrow r[[\varphi]] \in \mathcal{R}'$,
4. $l \succsim r$ for all $l \rightarrow r[[\varphi]] \in \mathcal{R} - \mathcal{R}'$,
5. $l \succsim r$ for all $l \rightarrow r \in \mathcal{S}$, and
6. $u \sim v$ for all $u \approx v \in \mathcal{E}$.

Using ordinary reduction pairs, dependency pairs $s \rightarrow t[[\varphi]]$ such that $s \succ t$ can be removed from a DP problem.

Theorem 7.8 (DP Processor Based on Ordinary Reduction Pairs). *Let (\succsim, \succ) be an ordinary reduction pair and let Proc be a DP processor with $\text{Proc}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) =$*

Chapter 7. Reduction Pairs

- $\{(\mathcal{P} - \mathcal{P}', \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, if $\mathcal{P}' \subseteq \mathcal{P}$ such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \emptyset, \mathcal{R}, \mathcal{S}, \mathcal{E})$.
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, otherwise.

Then Proc is sound.

Example 7.9. To illustrate Theorem 7.8 and ordinary reduction pairs based on polynomial interpretations, consider an RCERS over $\mathcal{Th}_{\mathbb{N}}$, i.e., \mathcal{S} and \mathcal{E} are as follows:

$$\begin{aligned} \mathcal{S} : \quad & x + 0 \rightarrow x \\ \mathcal{E} : \quad & x + y \approx y + x \\ & x + (y + z) \approx (x + y) + z \end{aligned}$$

Let \mathcal{R} consist of the rule $f(x + 1) \rightarrow f(x)$ and let \mathcal{Q} be arbitrary. Then the initial DP problem is $(\{f^\sharp(x + 1) \rightarrow f^\sharp(x)\}, \mathcal{R}, \mathcal{Q}, \mathcal{S}, \mathcal{E})$. This DP problem can be handled by the DP processor of Theorem 7.8 using the following polynomial interpretation:

$$\begin{aligned} \mathcal{Pol}(f^\sharp) &= x_1 \\ \mathcal{Pol}(f) &= 0 \\ \mathcal{Pol}(+) &= x_1 + x_2 \\ \mathcal{Pol}(1) &= 1 \\ \mathcal{Pol}(0) &= 0 \end{aligned}$$

Indeed, $(\succsim_{\mathcal{Pol}}, \succ_{\mathcal{Pol}}) \models (\{f^\sharp(x + 1) \rightarrow f^\sharp(x)\}, \{f^\sharp(x + 1) \rightarrow f^\sharp(x)\}, \emptyset, \mathcal{R}, \mathcal{S}, \mathcal{E})$. △

7.2 $\mathcal{Th}_{\mathbb{N}}$ -Reduction Pairs

For $\mathcal{Th}_{\mathbb{N}}$, the requirement that \succsim needs to be monotonic for all possible contexts can be relaxed since, given a set of dependency pairs, reductions with $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ may only take place in certain argument positions of a symbol $f^\sharp \in \mathcal{F}^\sharp$. This follows

Chapter 7. Reduction Pairs

directly from the requirement that the substitution σ used for building a chain is $\mathcal{Th}_{\mathbb{N}}$ -based. In particular, this relaxation of monotonicity makes it possible to use polynomial interpretations with negative coefficients for the function symbols in $\mathcal{F}^{\#}$. This is often needed for a successful termination proof, see Example 7.19 below. Similar reasoning applies for $\mathcal{Th}_{\mathbb{Z}}$ as well, see Section 7.3.

For function symbols $f \notin \mathcal{F}^{\#}$ nothing changes, i.e., monotonicity for contexts over $\mathcal{F} \cup \mathcal{F}_{\mathcal{Th}_{\mathbb{N}}}$ is still required for a $\mathcal{Th}_{\mathbb{N}}$ -reduction pair.²

Definition 7.10 (\mathcal{F} -Monotonic Relations). *A relation \bowtie is \mathcal{F} -monotonic iff $s \bowtie t$ implies $f(s_1, \dots, s_{i-1}, s, s_{i+1}, \dots, s_n) \bowtie f(s_1, \dots, s_{i-1}, t, s_{i+1}, \dots, s_n)$ for all $f \in \mathcal{F} \cup \mathcal{F}_{\mathcal{Th}_{\mathbb{N}}}$, all $1 \leq i \leq \text{arity}(f)$, and all $s, t, s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_n \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{Th}_{\mathbb{N}}}, \mathcal{V})$.*

As motivated above, monotonicity w.r.t. a context that has a symbol $f^{\#} \in \mathcal{F}^{\#}$ at its root is only required for certain argument positions of $f^{\#}$.

Definition 7.11 ($f^{\#}$ -Monotonic Relations). *Let $f^{\#} \in \mathcal{F}^{\#}$ and $1 \leq i \leq \text{arity}(f^{\#})$. A relation \bowtie is $f^{\#}$ -monotonic at position i iff $s \bowtie t$ implies $f^{\#}(s_1, \dots, s_{i-1}, s, s_{i+1}, \dots, s_n) \bowtie f^{\#}(s_1, \dots, s_{i-1}, t, s_{i+1}, \dots, s_n)$ for all $s, t, s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_n \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{Th}_{\mathbb{N}}}, \mathcal{V})$.*

Notice that \bowtie is monotonic in the sense of Definition 7.1 if and only if \bowtie is \mathcal{F} -monotonic and $f^{\#}$ -monotonic at position i for all $f^{\#} \in \mathcal{F}^{\#}$ and all $1 \leq i \leq \text{arity}(f^{\#})$.

When considering the DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$, the relation \succsim needs to be $f^{\#}$ -monotonic at position i only if \mathcal{P} contains a dependency pair of the form $s \rightarrow f^{\#}(t_1, \dots, t_i, \dots, t_n) \llbracket \varphi \rrbracket$ where $\text{sort}(t_i) = \text{univ}$ or $t_i \notin \mathcal{T}(\mathcal{F}_{\mathcal{Th}_{\mathbb{N}}}, \mathcal{V})$. The reason for this is that \succsim needs to be monotonic only in those argument positions of $\mathcal{F}^{\#}$ where a $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reduction may potentially take place. Notice that no instantiation of t_i can be reduced using $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ in a $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain if $\text{sort}(t_i) = \text{base}$ and

²Using ideas from [89, 74], it might be possible to relax this requirement. For simplicity, this is not considered in this dissertation and is left for future work.

Chapter 7. Reduction Pairs

$t_i \in \mathcal{T}(\mathcal{F}_{\mathcal{Th}_{\mathbb{N}}}, \mathcal{V})$ since the substitution σ used for the chain is $\mathcal{Th}_{\mathbb{N}}$ -based. A similar observation has already been made in the case of innermost termination of ordinary rewriting considered in [12]. In this dissertation, however, this refinement becomes applicable in the non-innermost case as well.

Definition 7.12 (Reducible Positions). *Let \mathcal{P} be a set of dependency pairs and let $f^\# \in \mathcal{F}^\#$. Then the set of reducible positions of $f^\#$ for \mathcal{P} is given by $\text{RedPos}(f^\#, \mathcal{P}) = \{i \mid \text{there exists } s \rightarrow f^\#(t_1, \dots, t_i, \dots, t_n) \llbracket \varphi \rrbracket \in \mathcal{P} \text{ such that } \text{sort}(t_i) = \mathbf{univ} \text{ or } t_i \notin \mathcal{T}(\mathcal{F}_{\mathcal{Th}}, \mathcal{V})\}$.*

Notice that an instance of a term t_i with $\text{sort}(t_i) = \mathbf{base}$ and $t_i \in \mathcal{T}(\mathcal{F}_{\mathcal{Th}_{\mathbb{N}}}, \mathcal{V})$ may still be reduced using the $\mathcal{Th}_{\mathbb{N}}$ -rules from \mathcal{S} . In order to ensure that these reductions result in terms that are equivalent w.r.t. $\gtrsim \cap \gtrsim^{-1}$, first define $\mathcal{S}_s = \{l \rightarrow r \in \mathcal{S} \mid \text{sort}(l) = s\}$, where $s \in \{\mathbf{base}, \mathbf{univ}\}$. It is then required that $\gtrsim \cap \gtrsim^{-1}$ is $f^\#$ -monotonic at position i for all $i \notin \text{RedPos}(f^\#, \mathcal{P})$ and that $l \gtrsim \cap \gtrsim^{-1} r$ for all $l \rightarrow r \in \mathcal{S}_{\mathbf{base}}$, i.e., the rules from $\mathcal{S}_{\mathbf{base}}$ need to be treated like the equations from \mathcal{E} . Then, $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}} t$ for $s, t \in \mathcal{T}(\mathcal{F}_{\mathcal{Th}_{\mathbb{N}}}, \mathcal{V})$ with $\text{sort}(s) = \text{sort}(t) = \mathbf{base}$ implies $s \gtrsim \cap \gtrsim^{-1} t$.

The notion of $\mathcal{Th}_{\mathbb{N}}$ -reduction pairs generalizes ordinary reduction pairs and depends on the DP problem under consideration. It is similar to the notion of *generalized reduction pairs* [89, 74] in the sense that full monotonicity is not required.

Definition 7.13 ($\mathcal{Th}_{\mathbb{N}}$ -Reduction Pairs). *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be a DP problem and let \gtrsim and \succ be relations on terms such that*

1. \gtrsim is reflexive, transitive, and \mathcal{F} -monotonic,
2. for all $f^\# \in \mathcal{F}^\#$,
 - \gtrsim is $f^\#$ -monotonic at position i for all $i \in \text{RedPos}(f^\#, \mathcal{P})$,
 - $\gtrsim \cap \gtrsim^{-1}$ is $f^\#$ -monotonic at position i for all $i \notin \text{RedPos}(f^\#, \mathcal{P})$, and

Chapter 7. Reduction Pairs

3. \succ is well-founded.

Then (\succsim, \succ) is a $\mathcal{Th}_{\mathbb{N}}$ -reduction pair for \mathcal{P} iff \succ is compatible with \succsim , i.e., iff $\succsim \circ \succ \circ \succsim \subseteq \succ$. The relation $\succsim \cap \succsim^{-1}$ is denoted by \sim .

Notice that neither \succsim nor \succ are required to be stable. Indeed, stability for all substitutions is not needed since this property is only required for certain substitutions that can be used in $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chains. These substitutions are indirectly given by the constraints of the dependency pairs and rules that are to be oriented.

Definition 7.14 (*$\mathcal{Th}_{\mathbb{N}}$ -Reduction Pairs on Constrained Terms*). *Let (\succsim, \succ) be a $\mathcal{Th}_{\mathbb{N}}$ -reduction pair. Let s, t be terms and let φ be a $\mathcal{Th}_{\mathbb{N}}$ -constraint. Then $s[\![\varphi]\!] \succsim t[\![\varphi]\!]$ iff $s\sigma \succsim t\sigma$ for all $\mathcal{Th}_{\mathbb{N}}$ -based substitutions σ such that $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{N}}$ -valid. Similarly, $s[\![\varphi]\!] \succ t[\![\varphi]\!]$ iff $s\sigma \succ t\sigma$ for all $\mathcal{Th}_{\mathbb{N}}$ -based substitutions σ such that $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{N}}$ -valid.*

The easiest way to generate $\mathcal{Th}_{\mathbb{N}}$ -reduction pairs is based on polynomial interpretations. In contrast to Section 7.1, however, some coefficients of the polynomials may now be negative. This increased flexibility is often needed for successful termination proofs. The class of polynomial interpretations considered here is similar to the class considered in [89].

A $\mathcal{Th}_{\mathbb{N}}$ -polynomial interpretation \mathcal{Pol} fixes a constant $c_{\mathcal{Pol}} \in \mathbb{Z}$ and maps

1. the symbols in $\mathcal{F}_{\mathcal{Th}_{\mathbb{N}}}$ to polynomials over \mathbb{N} in the natural way, i.e., $\mathcal{Pol}(0) = 0$, $\mathcal{Pol}(1) = 1$, and $\mathcal{Pol}(+) = x_1 + x_2$,
2. the symbols in \mathcal{F} to polynomials over \mathbb{N} such that $\mathcal{Pol}(f) \in \mathbb{N}[x_1, \dots, x_n]$ if $\text{arity}(f) = n$, and
3. the symbols in \mathcal{F}^{\sharp} to polynomials over \mathbb{Z} such that $\mathcal{Pol}(f^{\sharp}) \in \mathbb{Z}[x_1, \dots, x_n]$ if $\text{arity}(f^{\sharp}) = n$.

The reason for fixing the polynomials for the symbols from $\mathcal{F}_{\mathcal{Th}_{\mathbb{N}}}$ this way is that a term from $\mathcal{T}(\mathcal{F}_{\mathcal{Th}_{\mathbb{N}}})$ is then mapped to the natural number it represents. As will

Chapter 7. Reduction Pairs

become apparent later, this makes it possible to directly use the $\mathcal{Th}_{\mathbb{N}}$ -constraint when comparing two constrained terms.

$\mathcal{Th}_{\mathbb{N}}$ -polynomial interpretations generate relations on terms as follows. This is similar to Definition 7.4. Here, the condition $[s\sigma]_{\mathcal{P}ol} \geq c_{\mathcal{P}ol}$ in the definition of $\succ_{\mathcal{P}ol}$ is needed for well-foundedness.

Definition 7.15 ($\succ_{\mathcal{P}ol}$ and $\succeq_{\mathcal{P}ol}$ for $\mathcal{Th}_{\mathbb{N}}$ -Polynomial Interpretations). *Let $\mathcal{P}ol$ be a $\mathcal{Th}_{\mathbb{N}}$ -polynomial interpretation. Then $\succ_{\mathcal{P}ol}$ is defined by $s \succ_{\mathcal{P}ol} t$ iff $[s\sigma]_{\mathcal{P}ol} \geq c_{\mathcal{P}ol}$ and $[s\sigma]_{\mathcal{P}ol} > [t\sigma]_{\mathcal{P}ol}$ for all ground substitutions $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{Th}_{\mathbb{N}}})$. Similarly, $\succeq_{\mathcal{P}ol}$ is defined by $s \succeq_{\mathcal{P}ol} t$ iff $[s\sigma]_{\mathcal{P}ol} \geq [t\sigma]_{\mathcal{P}ol}$ for all ground substitutions $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{Th}_{\mathbb{N}}})$. Thus, $s \sim_{\mathcal{P}ol} t$ iff $[s\sigma]_{\mathcal{P}ol} = [t\sigma]_{\mathcal{P}ol}$ for all ground substitutions $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{Th}_{\mathbb{N}}})$.*

For a given $\mathcal{Th}_{\mathbb{N}}$ -polynomial interpretation, $s[\![\varphi]\!] \succ_{\mathcal{P}ol} t[\![\varphi]\!]$ is checked as follows. Recall that $s[\![\varphi]\!] \succ_{\mathcal{P}ol} t[\![\varphi]\!]$ means $s\sigma \succ_{\mathcal{P}ol} t\sigma$ for all $\mathcal{Th}_{\mathbb{N}}$ -based substitutions σ that make $\varphi\sigma$ $\mathcal{Th}_{\mathbb{N}}$ -valid. This, in turn, means $[s\sigma\sigma']_{\mathcal{P}ol} \geq c_{\mathcal{P}ol}$ and $[s\sigma\sigma']_{\mathcal{P}ol} > [t\sigma\sigma']_{\mathcal{P}ol}$ for all ground substitutions σ' . This can be achieved by showing that the following formulas are true in the integers:

$$\begin{aligned} \forall x_1 \geq 0, \dots, x_n \geq 0. \varphi \Rightarrow [s]_{\mathcal{P}ol} \geq c_{\mathcal{P}ol} \\ \forall x_1 \geq 0, \dots, x_n \geq 0. \varphi \Rightarrow [s]_{\mathcal{P}ol} > [t]_{\mathcal{P}ol} \end{aligned}$$

Here x_1, \dots, x_n are the variables occurring in s and t . Notice that it suffices to consider instantiations of x_1, \dots, x_n by natural numbers since all function symbols from $\mathcal{F} \cup \mathcal{F}_{\mathcal{Th}}$ are mapped to polynomials over \mathbb{N} and substitutions do not introduce function symbols from $\mathcal{F}^{\#}$. As for ordinary polynomial interpretations, it is decidable whether these formulas are true in the integers if $\mathcal{P}ol(f)$ is linear for all $f \in \mathcal{F}_{\mathcal{Th}} \cup \mathcal{F}^{\#}$. This again follows from the decidability of Presburger arithmetic [146].

Now $\succeq_{\mathcal{P}ol}$ and $\succ_{\mathcal{P}ol}$ indeed give rise to $\mathcal{Th}_{\mathbb{N}}$ -reduction pairs where $f^{\#}$ -monotonicity at position i is achieved by requiring that $\mathcal{P}ol(f^{\#})$ is weakly increasing in x_i .

Chapter 7. Reduction Pairs

Theorem 7.16. *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be a DP problem and let Pol be a $\text{Th}_{\mathbb{N}}$ -polynomial interpretation. Then $(\succsim_{\text{Pol}}, \succ_{\text{Pol}})$ is a $\text{Th}_{\mathbb{N}}$ -reduction pair for \mathcal{P} if $\text{Pol}(f^{\sharp})$ is weakly increasing in all x_i with $i \in \text{RedPos}(f^{\sharp}, \mathcal{P})$.*

Again, the following notation is introduced.

Definition 7.17. *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be a DP problem, let (\succsim, \succ) be a $\text{Th}_{\mathbb{N}}$ -reduction pair for \mathcal{P} , let \mathcal{P}' be a set of dependency pairs, and let \mathcal{R}' be a set of constrained rewrite rules. Then $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \mathcal{R}', \mathcal{R}, \mathcal{S}, \mathcal{E})$ iff $\text{Th}_{\mathbb{N}}$ is built-in and*

1. $s[[\varphi]] \succ t[[\varphi]]$ for all $s \rightarrow t[[\varphi]] \in \mathcal{P}'$,
2. $s[[\varphi]] \succsim t[[\varphi]]$ for all $s \rightarrow t[[\varphi]] \in \mathcal{P} - \mathcal{P}'$,
3. $l[[\varphi]] \succ r[[\varphi]]$ for all $l \rightarrow r[[\varphi]] \in \mathcal{R}'$,
4. $l[[\varphi]] \succsim r[[\varphi]]$ for all $l \rightarrow r[[\varphi]] \in \mathcal{R} - \mathcal{R}'$,
5. $l \succsim r$ for all $l \rightarrow r \in \mathcal{S}_{\text{univ}}$,
6. $l \sim r$ for all $l \rightarrow r \in \mathcal{S}_{\text{base}}$, and
7. $u \sim v$ for all $u \approx v \in \mathcal{E}$.

Using $\text{Th}_{\mathbb{N}}$ -reduction pairs, dependency pairs $s \rightarrow t[[\varphi]]$ such that $s[[\varphi]] \succ t[[\varphi]]$ can again be removed.

Theorem 7.18 (DP Processor Based on $\text{Th}_{\mathbb{N}}$ -Reduction Pairs). *Let Proc be a DP processor with $\text{Proc}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) =$*

- $\{(\mathcal{P} - \mathcal{P}', \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, if (\succsim, \succ) is a $\text{Th}_{\mathbb{N}}$ -reduction pair for \mathcal{P} and $\mathcal{P}' \subseteq \mathcal{P}$ such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \emptyset, \mathcal{R}, \mathcal{S}, \mathcal{E})$.
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, otherwise.

Then Proc is sound.

Example 7.19. In contrast to ordinary reduction pairs, $\mathcal{Th}_{\mathbb{N}}$ -reduction pairs can also be used to show termination that is due to a bounded increase, i.e., if an argument is counted upwards until a fixed bound is reached. To illustrate this, consider an RCERS over $\mathcal{Th}_{\mathbb{N}}$, i.e., \mathcal{S} and \mathcal{E} are as follows:

$$\begin{aligned} \mathcal{S} : \quad & x + 0 \rightarrow x \\ \mathcal{E} : \quad & x + y \approx y + x \\ & x + (y + z) \approx (x + y) + z \end{aligned}$$

Let \mathcal{R} consist of the rule $f(x, y) \rightarrow f(x + 1, y) \llbracket y > x \rrbracket$ and let \mathcal{Q} be arbitrary. The initial DP problem $(\{f^\sharp(x, y) \rightarrow f^\sharp(x + 1, y) \llbracket y > x \rrbracket\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ can be handled using the following $\mathcal{Th}_{\mathbb{N}}$ -polynomial interpretation with $c_{\text{Pol}} = 0$:

$$\begin{aligned} \text{Pol}(f^\sharp) &= x_2 - x_1 \\ \text{Pol}(f) &= 0 \\ \text{Pol}(+) &= x_1 + x_2 \\ \text{Pol}(1) &= 1 \\ \text{Pol}(0) &= 0 \end{aligned}$$

For this interpretation, $(\succsim_{\text{Pol}}, \succ_{\text{Pol}}) \models (\{f^\sharp(x, y) \rightarrow f^\sharp(x + 1, y) \llbracket y > x \rrbracket\}, \{f^\sharp(x, y) \rightarrow f^\sharp(x + 1, y) \llbracket y > x \rrbracket\}, \emptyset, \mathcal{R}, \mathcal{S}, \mathcal{E})$ because

$$\begin{aligned} \forall x \geq 0, y \geq 0. y > x &\Rightarrow y - x \geq 0 \\ \forall x \geq 0, y \geq 0. y > x &\Rightarrow y - x > y - (x + 1) \end{aligned}$$

are true in the integers. △

7.3 $\mathcal{Th}_{\mathbb{Z}}$ -Reduction Pairs

For $\mathcal{Th}_{\mathbb{Z}}$, the requirement that \succsim needs to be monotonic for all possible contexts can also be relaxed, albeit in a slightly different way than for $\mathcal{Th}_{\mathbb{N}}$. In order to introduce

Chapter 7. Reduction Pairs

$\mathcal{Th}_{\mathbb{Z}}$ -polynomial interpretations in analogy to the $\mathcal{Th}_{\mathbb{N}}$ -polynomial interpretations in Section 7.2, it becomes necessary to interpret the function symbol $- \in \mathcal{F}_{\mathcal{Th}_{\mathbb{Z}}}$ by the polynomial $\mathcal{Pol}(-) = -x_1$. But this clearly destroys \mathcal{F} -monotonicity. It thus becomes necessary to impose restrictions under which these kinds of polynomial interpretations may be applied.

If all arguments of right-hand sides of \mathcal{P} are terms from $\mathcal{T}(\mathcal{F}_{\mathcal{Th}_{\mathbb{Z}}}, \mathcal{V})$, then no reductions w.r.t. $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ can take place between instantiated dependency pairs in a chain since chains are built using $\mathcal{Th}_{\mathbb{Z}}$ -based substitutions (recall the discussion on this in Section 7.2). Thus, if $\mathcal{RedPos}(f^{\sharp}, \mathcal{P}) = \emptyset$ for all $f^{\sharp} \in \mathcal{F}^{\sharp}$, then the reduction pair does not need to be monotonic in the argument of $- \in \mathcal{F}_{\mathcal{Th}_{\mathbb{Z}}}$.

If this requirement is not fulfilled, then it can be ensured that no reduction w.r.t. $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ can take place below the function symbol $-$ if \mathcal{F} does not contain any function symbol with resulting sort **base**. In this case, it still needs to be required that function symbols from $\mathcal{F} \cup \mathcal{F}^{\sharp}$ are monotonic in all argument positions of sort **univ** since reductions may take place in these arguments.

Definition 7.20 (**univ-Monotonic Relations**). *A relation \bowtie is univ-monotonic iff $s \bowtie t$ implies $f(s_1, \dots, s_{i-1}, s, s_{i+1}, \dots, s_n) \bowtie f(s_1, \dots, s_{i-1}, t, s_{i+1}, \dots, s_n)$ for all $f \in \mathcal{F} \cup \mathcal{F}^{\sharp}$, all $1 \leq i \leq \text{arity}(f)$ such that the i^{th} argument of f has sort **univ**, and all $s, t, s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_n \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{Th}_{\mathbb{Z}}}, \mathcal{V})$.*

For argument positions of sort **base** the following, slightly different notion of monotonicity is required. This is essentially identical to the f^{\sharp} -monotonicity of $\succsim \cap \succsim^{-1}$ required for $\mathcal{Th}_{\mathbb{N}}$ -reduction pairs and again ensures that reductions with $\mathcal{S}_{\text{base}}$ result in terms that are equivalent w.r.t. $\succsim \cap \succsim^{-1}$.

Definition 7.21 (**base-Monotonic Relations**). *Let \bowtie be a relation and let $\widehat{\bowtie} = \bowtie \cap \bowtie^{-1}$. Then \bowtie is base-monotonic iff $s \widehat{\bowtie} t$ implies $f(s_1, \dots, s_{i-1}, s, s_{i+1}, \dots, s_n) \widehat{\bowtie} f(s_1, \dots, s_{i-1}, t, s_{i+1}, \dots, s_n)$ for all $f \in \mathcal{F} \cup \mathcal{F}_{\mathcal{Th}_{\mathbb{Z}}} \cup \mathcal{F}^{\sharp}$, all $1 \leq i \leq \text{arity}(f)$ such that*

Chapter 7. Reduction Pairs

the i^{th} argument of f has sort **base**, and all $s, t, s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_n \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{Th}_{\mathbb{Z}}}, \mathcal{V})$.

As for $\mathcal{Th}_{\mathbb{N}}$ -reduction pairs, the notion of $\mathcal{Th}_{\mathbb{Z}}$ -reduction pairs depends on the DP problem under consideration.

Definition 7.22 ($\mathcal{Th}_{\mathbb{Z}}$ -Reduction Pairs). *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be a DP problem and let \succsim and \succ be relations on terms such that \succsim is reflexive, transitive, univ-monotonic, and base-monotonic and \succ is well-founded. Then (\succsim, \succ) is a \mathcal{Th} -reduction pair for \mathcal{P} iff \succ is compatible with \succsim , i.e., iff $\succsim \circ \succ \circ \succsim \subseteq \succ$. The relation $\succsim \cap \succsim^{-1}$ is denoted by \sim .*

Notice that \succsim and \succ are again not required to be stable. $\mathcal{Th}_{\mathbb{Z}}$ -reduction pairs can be used for constrained terms just like this is done for $\mathcal{Th}_{\mathbb{N}}$ -reduction pairs.

The easiest way to generate $\mathcal{Th}_{\mathbb{Z}}$ -reduction pairs is again based on polynomial interpretations. A $\mathcal{Th}_{\mathbb{Z}}$ -polynomial interpretation \mathcal{Pol} fixes a constant $c_{\mathcal{Pol}} \in \mathbb{Z}$ and maps

1. the symbols in $\mathcal{F}_{\mathcal{Th}_{\mathbb{Z}}}$ to polynomials over \mathbb{Z} in the natural way, i.e., $\mathcal{Pol}(0) = 0$, $\mathcal{Pol}(1) = 1$, $\mathcal{Pol}(-) = -x_1$ and $\mathcal{Pol}(+) = x_1 + x_2$,
2. the symbols in \mathcal{F} to polynomials over \mathbb{N} such that $\mathcal{Pol}(f) \in \mathbb{N}[x_1, \dots, x_n]$ if $\text{arity}(f) = n$, and
3. the symbols in \mathcal{F}^{\sharp} to polynomials over \mathbb{Z} such that $\mathcal{Pol}(f^{\sharp}) \in \mathbb{Z}[x_1, \dots, x_n]$ if $\text{arity}(f^{\sharp}) = n$.

As for $\mathcal{Th}_{\mathbb{N}}$ -polynomial interpretations, the reason for fixing the polynomials for the symbols from $\mathcal{F}_{\mathcal{Th}_{\mathbb{Z}}}$ this way is that a term from $\mathcal{T}(\mathcal{F}_{\mathcal{Th}_{\mathbb{Z}}})$ is then mapped to the integer it represents. Again, this will make it possible to directly use the $\mathcal{Th}_{\mathbb{Z}}$ -constraint when comparing two constrained terms.

The following is identical to $\mathcal{Th}_{\mathbb{N}}$ -polynomial interpretations, cf. Definition 7.15.

Chapter 7. Reduction Pairs

Definition 7.23 ($\succ_{\mathcal{P}ol}$ and $\succsim_{\mathcal{P}ol}$ for $\mathcal{Th}_{\mathbb{Z}}$ -Polynomial Interpretations). *Let $\mathcal{P}ol$ be a $\mathcal{Th}_{\mathbb{Z}}$ -polynomial interpretation. Then $\succ_{\mathcal{P}ol}$ is defined by $s \succ_{\mathcal{P}ol} t$ iff $[s\sigma]_{\mathcal{P}ol} \geq c_{\mathcal{P}ol}$ and $[s\sigma]_{\mathcal{P}ol} > [t\sigma]_{\mathcal{P}ol}$ for all ground substitutions $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{Th}_{\mathbb{Z}}})$. Similarly, $\succsim_{\mathcal{P}ol}$ is defined by $s \succsim_{\mathcal{P}ol} t$ iff $[s\sigma]_{\mathcal{P}ol} \geq [t\sigma]_{\mathcal{P}ol}$ for all ground substitutions $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{Th}_{\mathbb{Z}}})$. Thus, $s \sim_{\mathcal{P}ol} t$ iff $[s\sigma]_{\mathcal{P}ol} = [t\sigma]_{\mathcal{P}ol}$ for all ground substitutions $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{Th}_{\mathbb{Z}}})$.*

Checking $s[[\varphi]] \succ_{\mathcal{P}ol} t[[\varphi]]$ is done similarly to $\mathcal{Th}_{\mathbb{N}}$ -polynomial interpretations. Recall that it suffices to show that $[s\sigma\sigma']_{\mathcal{P}ol} \geq c_{\mathcal{P}ol}$ and $[s\sigma\sigma']_{\mathcal{P}ol} > [t\sigma\sigma']_{\mathcal{P}ol}$ for all $\mathcal{Th}_{\mathbb{Z}}$ -based substitutions σ such that $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid and for all ground substitutions σ' . This can be achieved by showing that the following formulas are true in the integers:

$$\begin{aligned} \forall x_1, \dots, x_n. \varphi \Rightarrow [s]_{\mathcal{P}ol} &\geq c_{\mathcal{P}ol} \\ \forall x_1, \dots, x_n. \varphi \Rightarrow [s]_{\mathcal{P}ol} &> [t]_{\mathcal{P}ol} \end{aligned}$$

Here, x_1, \dots, x_n are the variables occurring in s and t . In contrast to $\mathcal{Th}_{\mathbb{N}}$ -polynomial interpretations, the variables have to be instantiated by negative numbers as well since $\mathcal{P}ol(-) = -x_1$. As for ordinary polynomial interpretations and $\mathcal{Th}_{\mathbb{N}}$ -polynomial interpretations, it is decidable whether these formulas are true in the integers if $\mathcal{P}ol(f)$ is linear for all $f \in \mathcal{F}_{\mathcal{Th}} \cup \mathcal{F} \cup \mathcal{F}^{\#}$ since this again follows from the decidability of Presburger arithmetic [146].

The above requirements might be impossible to show if one of the x_i has sort **univ** since then the possible values it can take are not restricted by the $\mathcal{Th}_{\mathbb{Z}}$ -constraint φ . By restricting the $\mathcal{Th}_{\mathbb{Z}}$ -polynomial interpretation $\mathcal{P}ol$ such that for each $f \in \mathcal{F}$ with resulting sort **univ**, the polynomial $\mathcal{P}ol(f)$ may only depend on a variable x_i if the i^{th} argument of f has sort **univ**, a requirement that can often be shown easier is obtained. Now showing $s[[\varphi]] \succ_{\mathcal{P}ol} t[[\varphi]]$ can be achieved by showing that the following

Chapter 7. Reduction Pairs

formulas are true in the integers:

$$\begin{aligned} & \forall x_1, \dots, x_k. \forall y_1 \geq 0, \dots, y_l \geq 0. \varphi \Rightarrow [s]_{\mathcal{P}ol} \geq c_{\mathcal{P}ol} \\ & \forall x_1, \dots, x_k. \forall y_1 \geq 0, \dots, y_l \geq 0. \varphi \Rightarrow [s]_{\mathcal{P}ol} > [t]_{\mathcal{P}ol} \end{aligned}$$

Here, x_1, \dots, x_k are the variables of sort **base** in s and t and y_1, \dots, y_l are the variables of sort **univ** in s and t . As before, it is still decidable whether these formulas are true in the integers if $\mathcal{P}ol(f)$ is linear for all $f \in \mathcal{F}_{Th} \cup \mathcal{F} \cup \mathcal{F}^\sharp$.

Example 7.24. Assume that \mathcal{F} contains $g : \mathbf{base} \times \mathbf{univ} \rightarrow \mathbf{univ}$ and $f : \mathbf{univ} \rightarrow \mathbf{univ}$. Then, consider the $Th_{\mathbb{Z}}$ -polynomial interpretation with $c_{\mathcal{P}ol} = 0$, $\mathcal{P}ol(f) = x_1$, and $\mathcal{P}ol(g) = x_2 + 1$. Thus, $[f(g(x, y))]_{\mathcal{P}ol} = y + 1$ and $[f(y)] = y$. Notice that for each function symbol $f \in \mathcal{F}$ with resulting sort **univ**, the polynomial $\mathcal{P}ol(f)$ only depends on variables corresponding to arguments of sort **univ**. Using the refinement discussed above, $f(g(x, y)) \succ_{\mathcal{P}ol} f(y)$ since the following formulas are true in the integers:

$$\begin{aligned} & \forall y \geq 0. y + 1 \geq 0 \\ & \forall y \geq 0. y + 1 > y \end{aligned}$$

In particular, the first formula requires the assumption that y is non-negative. Without the refinement, $f(g(x, y)) \succ_{\mathcal{P}ol} f(y)$ cannot be established since

$$\forall y. y + 1 \geq 0$$

is *not* true in the integers. △

The following lemma justifies that it suffices to instantiate the variables of sort **univ** only by non-negative numbers if this refinement is used.

Lemma 7.25. *Let $\mathcal{P}ol$ be a $Th_{\mathbb{Z}}$ -polynomial interpretation such that for each $f \in \mathcal{F}$ with resulting sort **univ**, the polynomial $\mathcal{P}ol(f)$ only depends on a variable x_i if the i^{th} argument of f has sort **univ**. If $s \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{Th_{\mathbb{Z}}})$ such that s has sort **univ**, then $[s]_{\mathcal{P}ol} \geq 0$.*

Chapter 7. Reduction Pairs

As for $\mathcal{Th}_{\mathbb{N}}$ -polynomial interpretations, the relations $\succsim_{\mathcal{P}ol}$ and $\succ_{\mathcal{P}ol}$ indeed give rise to $\mathcal{Th}_{\mathbb{Z}}$ -reduction pairs. The monotonicity requirements are again achieved by requiring $\mathcal{P}ol(f^\sharp)$ to be weakly increasing in all x_i that correspond to an argument of sort `univ`.

Theorem 7.26. *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be a DP problem and let $\mathcal{P}ol$ be a $\mathcal{Th}_{\mathbb{Z}}$ -polynomial interpretation. Then $(\succsim_{\mathcal{P}ol}, \succ_{\mathcal{P}ol})$ is a $\mathcal{Th}_{\mathbb{Z}}$ -reduction pair for \mathcal{P} if $\mathcal{P}ol(f^\sharp)$ is weakly increasing in all x_i where the i^{th} argument of f^\sharp has sort `univ`.*

Finally, the following notation is also introduced for $\mathcal{Th}_{\mathbb{Z}}$ -reduction pairs.

Definition 7.27. *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be a DP problem, let (\succsim, \succ) be a $\mathcal{Th}_{\mathbb{Z}}$ -reduction pair for \mathcal{P} , let \mathcal{P}' be a set of dependency pairs, and let \mathcal{R}' be a set of constrained rewrite rules. Then $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \mathcal{R}', \mathcal{R}, \mathcal{S}, \mathcal{E})$ is defined to hold iff $\mathcal{Th}_{\mathbb{Z}}$ is built-in, either $\text{RedPos}(f^\sharp, \mathcal{P}) = \emptyset$ for all $f^\sharp \in \mathcal{F}^\sharp$ or \mathcal{F} does not contain any function symbol with resulting sort `base`, and*

1. $s[[\varphi]] \succ t[[\varphi]]$ for all $s \rightarrow t[[\varphi]] \in \mathcal{P}'$,
2. $s[[\varphi]] \succsim t[[\varphi]]$ for all $s \rightarrow t[[\varphi]] \in \mathcal{P} - \mathcal{P}'$,
3. $l[[\varphi]] \succ r[[\varphi]]$ for all $l \rightarrow r[[\varphi]] \in \mathcal{R}'$,
4. $l[[\varphi]] \succsim r[[\varphi]]$ for all $l \rightarrow r[[\varphi]] \in \mathcal{R} - \mathcal{R}'$,
5. $l \succsim r$ for all $l \rightarrow r \in \mathcal{S}_{\text{univ}}$,
6. $l \sim r$ for all $l \rightarrow r \in \mathcal{S}_{\text{base}}$, and
7. $u \sim v$ for all $u \approx v \in \mathcal{E}$.

As before, dependency pairs $s \rightarrow t[[\varphi]]$ such that $s[[\varphi]] \succ t[[\varphi]]$ can be removed from a DP problem.

Theorem 7.28 (DP Processor Based on $\mathcal{Th}_{\mathbb{Z}}$ -Reduction Pairs). *Let Proc be a DP processor with $\text{Proc}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) =$*

Chapter 7. Reduction Pairs

- $\{(\mathcal{P} - \mathcal{P}', \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, if (\succsim, \succ) is a $Th_{\mathbb{Z}}$ -reduction pair for \mathcal{P} and $\mathcal{P}' \subseteq \mathcal{P}$ such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \emptyset, \mathcal{R}, \mathcal{S}, \mathcal{E})$.
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, otherwise.

Then Proc is sound.

Example 7.29. Using $Th_{\mathbb{Z}}$ -polynomial interpretations, termination of the CERS (and thus the imperative program) from Example 4.5 can be established. Recall the following rules and equations:

$$\begin{array}{l}
 \mathcal{S} : \quad x + 0 \rightarrow x \\
 \quad \quad - - x \rightarrow x \\
 \quad \quad -0 \rightarrow 0 \\
 \quad \quad -(x + y) \rightarrow (-x) + (-y) \\
 \quad \quad x + (-x) \rightarrow 0 \\
 \quad \quad (x + (-x)) + y \rightarrow 0 + y \\
 \mathcal{E} : \quad x + y \approx y + x \\
 \quad \quad x + (y + z) \approx (x + y) + z \\
 \mathcal{R} : \quad \text{eval}_1(x, y) \rightarrow \text{eval}_2(x, 1) \llbracket x \geq 0 \rrbracket \\
 \quad \quad \text{eval}_2(x, y) \rightarrow \text{eval}_2(x, 2 \cdot y) \llbracket x \geq 0 \wedge y > 0 \wedge x > y \rrbracket \\
 \quad \quad \text{eval}_2(x, y) \rightarrow \text{eval}_1(x - 1, y) \llbracket x \geq 0 \wedge y > 0 \wedge x \not> y \rrbracket
 \end{array}$$

There are three dependency pairs:

$$\text{eval}_1^\sharp(x, y) \rightarrow \text{eval}_2^\sharp(x, 1) \llbracket x \geq 0 \rrbracket \quad (7.1)$$

$$\text{eval}_2^\sharp(x, y) \rightarrow \text{eval}_2^\sharp(x, 2 \cdot y) \llbracket x \geq 0 \wedge y > 0 \wedge x > y \rrbracket \quad (7.2)$$

$$\text{eval}_2(x, y)^\sharp \rightarrow \text{eval}_1^\sharp(x - 1, y) \llbracket x \geq 0 \wedge y > 0 \wedge x \not> y \rrbracket \quad (7.3)$$

For an arbitrary \mathcal{Q} , the DP problem $(\{(7.1), (7.2), (7.3)\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ is transformed into the DP problem $(\{(7.1), (7.2)\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ using a $Th_{\mathbb{Z}}$ -polynomial interpretation with $\text{Pol}(\text{eval}_1^\sharp) = \text{Pol}(\text{eval}_2^\sharp) = x_1$, $\text{Pol}(\text{eval}_1) = \text{Pol}(\text{eval}_2) = 0$, and $c_{\text{Pol}} = 0$

Chapter 7. Reduction Pairs

because

$$\begin{aligned} \forall x, y. x \geq 0 &\Rightarrow x \geq x \\ \forall x, y. x \geq 0 \wedge y > 0 \wedge x > y &\Rightarrow x \geq x \\ \forall x, y. x \geq 0 \wedge y > 0 \wedge x \not> y &\Rightarrow x \geq 0 \\ \forall x, y. x \geq 0 \wedge y > 0 \wedge x \not> y &\Rightarrow x > x - 1 \end{aligned}$$

are true in the integers. The newly obtained DP problem is transformed into the DP problem $(\{(7.2)\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ since (7.1) is not in the SCC of the problem's dependency graph. Finally, the DP problem $(\{(7.2)\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ can be handled using a $\mathcal{Th}_{\mathbb{Z}}$ -polynomial interpretation with $\mathcal{Pol}(\text{eval}_2^\#) = x_1 - x_2$, $\mathcal{Pol}(\text{eval}_1) = \mathcal{Pol}(\text{eval}_2) = 0$, and $c_{\mathcal{Pol}} = 0$ since

$$\begin{aligned} \forall x, y. x \geq 0 \wedge y > 0 \wedge x > y &\Rightarrow x - y \geq 0 \\ \forall x, y. x \geq 0 \wedge y > 0 \wedge x > y &\Rightarrow x - y > x - 2 \cdot y \end{aligned}$$

are true in the integers. △

Notice that the applicability conditions for $\mathcal{Th}_{\mathbb{Z}}$ -reduction pairs might not always be satisfied, i.e., \mathcal{F} may contain function symbols with resulting sort **base** and $\mathcal{RedPos}(f^\#, \mathcal{P}) \neq \emptyset$ for some $f^\# \in \mathcal{F}^\#$. In this case it might be possible to eliminate certain argument positions of $f^\#$ using the DP processor from Theorem 6.17 such that $\mathcal{RedPos}(f^\#, \mathcal{P})$ becomes empty.

Example 7.30. The RCERS in this example uses non-determinism in order compute a random number. More precisely, $\text{random}(x)$ computes a random number in the interval $[0..x]$ for $x \geq 0$. The sets \mathcal{E} and \mathcal{S} are used to model $\mathcal{Th}_{\mathbb{Z}}$, $\mathcal{Q} = \emptyset$, and \mathcal{R} consists of the following rules:

$$\begin{aligned} \text{random}(x) &\rightarrow \text{rand}(x, 0) \llbracket x \geq 0 \rrbracket \\ \text{rand}(x, y) &\rightarrow y \llbracket x \simeq 0 \rrbracket \end{aligned}$$

$$\begin{aligned} \text{rand}(x, y) &\rightarrow \text{rand}(x - 1, \text{id_inc}(y)) \llbracket x > 0 \rrbracket \\ \text{id_inc}(x) &\rightarrow x \\ \text{id_inc}(x) &\rightarrow x + 1 \end{aligned}$$

Application of the dependency graph results in the DP problem $(\{\text{rand}^\sharp(x, y) \rightarrow \text{rand}^\sharp(x - 1, \text{id_inc}(y)) \llbracket x > 0 \rrbracket\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$. Then $\mathcal{Th}_{\mathbb{Z}}$ -reduction pairs cannot be applied since $\mathcal{RedPos}(\text{rand}^\sharp, \mathcal{P}) = \{2\}$ and id_inc has resulting sort **base**.

Using the non-collapsing argument filtering $\pi(\text{rand}^\sharp) = [1]$, the DP problem $(\{\text{rand}^\sharp(x) \rightarrow \text{rand}^\sharp(x - 1) \llbracket x > 0 \rrbracket\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ is obtained by the DP processor from Theorem 6.17. By reasoning formalized below in Theorem 8.18, the rules in \mathcal{R} do not need to be considered when applying Theorem 7.28 and the $\mathcal{Th}_{\mathbb{Z}}$ -polynomial interpretation with $c_{\mathcal{P}ol} = 0$ and $\mathcal{P}ol(\text{rand}^\sharp) = x_1$ can be used to show termination of the RCERS. \triangle

7.4 Summary

This chapter has presented three variations on the theme of reduction pairs which are based on well-founded relations. After recalling the ordinary reduction pairs from [117], reduction pairs tailored towards $\mathcal{Th}_{\mathbb{N}}$ and $\mathcal{Th}_{\mathbb{Z}}$ were introduced and it has been shown how these new variations of reduction pairs can be obtained using polynomial interpretations with integer coefficients. The use of negative coefficients makes it possible to successfully prove termination in many cases where this is not possible using ordinary polynomial interpretations with coefficients from \mathbb{N} . This is in particular true for examples where termination is due to a bounded increase of an argument until it reaches a fixed upper bound. The automatic generation of such polynomial interpretations will be discussed in Section 9.2.

While the use of reduction pairs as presented in this chapter is already sufficient

Chapter 7. Reduction Pairs

for showing termination of many examples, it has the drawback that all of \mathcal{R} , \mathcal{S} , and \mathcal{E} need to be considered when operating on the DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$. This is problematic since it makes the automatic generation of reduction pairs harder or impossible. Furthermore, it may harm performance of an implementation. Improved methods that make use of reduction pairs are presented in Chapter 8. These methods have the advantage that it becomes possible to consider only (syntactically determined) subsets of \mathcal{R} , \mathcal{S} , and \mathcal{E} , thus circumventing the drawbacks of the methods presented in this chapter.

Chapter 8

Usable Rules and Function Dependencies

Chapters 6 and 7 have already presented several sound DP processors. However, these DP processors are not yet sufficient for automatically proving termination of many natural examples, including the running quicksort example. This chapter introduces further, more powerful DP processors by improving the removal of dependency pairs from a DP problem using reduction pairs as introduced in Chapter 7. As already mentioned in Section 7.4, the DP processors introduced in Chapter 7 have the drawback that all of \mathcal{R} , \mathcal{S} , and \mathcal{E} need to be considered when operating on the DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$. This requirement makes it often hard or impossible to automatically find suitable reduction pairs. The techniques presented in this chapter make it possible to only consider (syntactically determined) subsets of \mathcal{R} , \mathcal{S} , and \mathcal{E} . This increases both efficiency and power of an implementation.

If $\mathcal{Q} \supseteq \mathcal{R}$ (i.e., in the innermost case), then it is possible to make use of the concept of *usable rules*, which is well-known from the dependency pair method for ordinary innermost rewriting [12]. This makes it possible to disregard certain rewrite rules

from \mathcal{R} that cannot be used for building chains. Extending the DP processor based on reduction pairs and usable rules to the case where $\mathcal{Q} \not\supseteq \mathcal{R}$ is quite challenging since the soundness proof of the former DP processor inherently depends on the assumption that $\mathcal{Q} \supseteq \mathcal{R}$. Nonetheless, Section 8.3 presents an extension to the case where it is not assumed that $\mathcal{Q} \supseteq \mathcal{R}$. Somewhat surprisingly, this extension results in a DP processor that is, in practice, slightly *more* powerful than the DP processor based on usable rules.

8.1 Usable Rules

If a reduction pair (\succsim, \succ) is used in the case where $\mathcal{Q} \supseteq \mathcal{R}$, then it is not necessary to require that all rules from \mathcal{R} are decreasing w.r.t. \succsim in Theorems 7.8, 7.18, and 7.28. Instead, it suffices to require this for the *usable rules*. These rules are a superset of the rules that may be used in a chain. This is similar to the situation for ordinary innermost rewriting [12]. Recall that variables are instantiated by terms that are irreducible by $\xrightarrow{S}_{Th \parallel \mathcal{E} \setminus \mathcal{Q}}$ in chains. Thus, the set of usable rules that may be applied to an instantiation of a variable is empty. For a term $t = f(t_1, \dots, t_n)$, all rules defining the function symbol f may be applicable, and furthermore any rule that may be applied to one of the t_i is usable. Then, this reasoning needs to be iterated for the right-hand sides of the usable rules determined so far.

Definition 8.1 (Usable Rules). *Let \mathcal{R} be a set of constrained rewrite rules. For any $f \in \mathcal{F}$ let $\mathcal{R}(f) = \{l \rightarrow r \llbracket \varphi \rrbracket \in \mathcal{R} \mid \text{root}(l) = f\}$. For any term define*

- $\mathcal{U}_{\mathcal{R}}(x) = \emptyset$ for $x \in \mathcal{V}$, and
- $\mathcal{U}_{\mathcal{R}}(f(t_1, \dots, t_n)) = \mathcal{R}(f) \cup \bigcup_{l \rightarrow r \llbracket \varphi \rrbracket \in \mathcal{R}(f)} \mathcal{U}_{\mathcal{R}'}(r) \cup \bigcup_{i=1}^n \mathcal{U}_{\mathcal{R}'}(t_i)$, where $\mathcal{R}' = \mathcal{R} - \mathcal{R}(f)$.

For a set \mathcal{P} of dependency pairs let $\mathcal{U}_{\mathcal{R}}(\mathcal{P}) = \bigcup_{s \rightarrow t \llbracket \varphi \rrbracket \in \mathcal{P}} \mathcal{U}_{\mathcal{R}}(t)$.

Example 8.2. Continuing the running quicksort example, Example 6.26, the only DP problem that still needs to be handled is $(\{5.7, 5.9\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ with the following dependency pairs:

$$\text{qsort}^\sharp(\{x\} \cup y) \rightarrow \text{qsort}^\sharp(\text{low}(x, y)) \quad (5.7)$$

$$\text{qsort}^\sharp(\{x\} \cup y) \rightarrow \text{qsort}^\sharp(\text{high}(x, y)) \quad (5.9)$$

The usable rules $\mathcal{U}_{\mathcal{R}}(\{5.7, 5.9\})$ consist of the rules for **low** and **high**. △

For innermost termination of ordinary rewriting, slightly stronger versions of usable rules have been developed [88, 87]. These definitions could be adapted for RCERSs, but in order to keep the presentation simple this dissertation restricts itself to the most basic version of Definition 8.1.

The next lemma states that $\mathcal{U}_{\mathcal{R}}(\mathcal{P})$ indeed contains all rules that are applicable in a $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain. A $\mathcal{T}h$ -based substitution σ is called *normal* iff $\sigma(x)$ is irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{T}h \parallel \mathcal{E} \setminus \mathcal{Q}}$ for all $x \in \mathcal{V}$. Notice that a term which is irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{T}h \parallel \mathcal{E} \setminus \mathcal{Q}}$ cannot be reduced using a rule from \mathcal{R} either if $\mathcal{Q} \supseteq \mathcal{R}$.

Lemma 8.3. *Let $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be an RCERS with $\mathcal{Q} \supseteq \mathcal{R}$ and let σ a normal substitution. Then for all s, t :*

1. *If $s\sigma \xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{T}h \parallel \mathcal{E} \setminus \mathcal{R}} t$ then $s\sigma \xrightarrow{\mathcal{S}}_{\mathcal{T}h \parallel \mathcal{E} \setminus \mathcal{U}_{\mathcal{R}}(s)} t$. Moreover, there exist a term u and a normal substitution μ such that $t = u\mu$ and $\mathcal{U}_{\mathcal{R}}(u) \subseteq \mathcal{U}_{\mathcal{R}}(s)$.*
2. *If $s\sigma \xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{T}h \parallel \mathcal{E} \setminus \mathcal{R}}^* \circ \xrightarrow{>\Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \circ \xrightarrow{\geq \Lambda}_{\mathcal{E}} t$ then $s\sigma \xrightarrow{\mathcal{S}}_{\mathcal{T}h \parallel \mathcal{E} \setminus \mathcal{U}_{\mathcal{R}}(s)}^* \circ \xrightarrow{>\Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \circ \xrightarrow{\geq \Lambda}_{\mathcal{E}} t$.*

Using reduction pairs, dependency pairs that are strictly decreasing can be removed from a DP problem as in Chapter 7. In contrast to the earlier DP processors, it now suffices to consider only the usable rules.

Theorem 8.4 (DP Processor Based on usable Rules). *Let Proc be a DP processor with $\text{Proc}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) =$*

- $\{(\mathcal{P} - \mathcal{P}', \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, if $\mathcal{Q} \supseteq \mathcal{R}$, $\mathcal{P}' \subseteq \mathcal{P}$, and either
 - there exists an ordinary reduction pair (\succsim, \succ) such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \emptyset, \mathcal{U}_{\mathcal{R}}(\mathcal{P}), \mathcal{S}, \mathcal{E})$, or
 - there exists a $Th_{\mathbb{N}}$ -reduction pair (\succsim, \succ) for \mathcal{P} such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \emptyset, \mathcal{U}_{\mathcal{R}}(\mathcal{P}), \mathcal{S}, \mathcal{E})$, or
 - there exists a $Th_{\mathbb{Z}}$ -reduction pair (\succsim, \succ) for \mathcal{P} such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \emptyset, \mathcal{U}_{\mathcal{R}}(\mathcal{P}), \mathcal{S}, \mathcal{E})$.
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, otherwise.

Then Proc is sound.

Example 8.5. Continuing Example 8.2, the requirement $(\succsim_{\mathcal{P}ol}, \succ_{\mathcal{P}ol}) \models (\{5.7, 5.9\}, \{5.7, 5.9\}, \emptyset, \mathcal{U}_{\mathcal{R}}(\{5.7, 5.9\}), \mathcal{S}, \mathcal{E})$ is satisfied for the following (ordinary) polynomial interpretation:

$$\begin{aligned}
 \mathcal{P}ol(\text{qsort}^\#) &= x_1 \\
 \mathcal{P}ol(\cup) &= x_1 + x_2 + 1 \\
 \mathcal{P}ol(\emptyset) &= 0 \\
 \mathcal{P}ol(\{\cdot\}) &= 0 \\
 \mathcal{P}ol(\text{low}) &= x_2 \\
 \mathcal{P}ol(\text{high}) &= x_2
 \end{aligned}$$

This concludes the termination proof of quicksort in the case where $\mathcal{Q} = \mathcal{R}$. △

8.2 Removal of Rules

The DP processors introduced in Chapter 7 and Theorem 8.4 can only be used to remove dependency pairs from a termination problem, while the set \mathcal{R} of rewrite rules

stays unchanged. But it might be desirable to remove rules from \mathcal{R} as well since this can simplify a termination proof substantially because rules that have been removed once do not need to be considered again later on. Additionally, the set of defined symbols might decrease if rules from \mathcal{R} are removed, which in turn might cause the removal of arcs in the (estimated) dependency graph. In this section, a DP processor for the removal of rules from \mathcal{R} is introduced. Like the DP processors in Chapter 7 and Theorem 8.4, it makes use of reduction pairs, but for soundness it needs to be required that the reduction pair is *monotonic*. Then, application of a rule $l \rightarrow r[[\varphi]]$ with $l[[\varphi]] \succ r[[\varphi]]$ in a reduction $s \xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}} t$ implies $s \succ t$.

Definition 8.6 (Monotonic Reduction Pairs). *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be a DP problem.*

1. *An ordinary reduction pair (\succsim, \succ) is monotonic iff \succ is monotonic.*
2. *A $Th_{\mathbb{N}}$ -reduction (\succsim, \succ) for \mathcal{P} is monotonic iff \succ is \mathcal{F} -monotonic and $f^{\#}$ -monotonic at position i for all $f^{\#} \in \mathcal{F}^{\#}$ and all $i \in \text{RedPos}(f^{\#}, \mathcal{P})$.*
3. *A $Th_{\mathbb{Z}}$ -reduction (\succsim, \succ) for \mathcal{P} is monotonic iff \succ is **univ**-monotonic.*

In order to obtain monotonic reduction pairs from polynomial interpretations, the polynomials need to satisfy the following requirements.

Theorem 8.7. *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be a DP problem.*

1. *An ordinary reduction pair $(\succsim_{\text{Pol}}, \succ_{\text{Pol}})$ generated using a polynomial interpretation is monotonic if $\text{Pol}(f)$ is increasing in all x_i for all $f \in \mathcal{F} \cup \mathcal{F}_{Th} \cup \mathcal{F}^{\#}$.*
2. *A $Th_{\mathbb{N}}$ -reduction pair for \mathcal{P} generated using a $Th_{\mathbb{N}}$ -polynomial interpretation is monotonic if $\text{Pol}(f)$ is increasing in all x_i for any $f \in \mathcal{F} \cup \mathcal{F}_{Th_{\mathbb{N}}}$ and $\text{Pol}(f^{\#})$ is increasing in all x_i with $i \in \text{RedPos}(f^{\#}, \mathcal{P})$ for any $f^{\#} \in \mathcal{F}^{\#}$.*
3. *A $Th_{\mathbb{Z}}$ -reduction pair for \mathcal{P} generated using a $Th_{\mathbb{Z}}$ -polynomial interpretation is monotonic if $\text{Pol}(f)$ is increasing in all x_i where the i^{th} argument of f has sort **univ** for any $f \in \mathcal{F} \cup \mathcal{F}_{Th} \cup \mathcal{F}^{\#}$.*

The following DP processor is similar to the DP processor of Theorem 8.4 but requires a monotonic reduction pair. In contrast to the earlier DP processor, rules from \mathcal{R} may now be removed as well if they are decreasing w.r.t. \succ .

Theorem 8.8 (DP Processor Based on Removal of Rules). *Let Proc be a DP processor with $\text{Proc}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) =$*

- $\{(\mathcal{P} - \mathcal{P}', \mathcal{R} - \mathcal{R}', \mathcal{S}, \mathcal{E})\}$, if $\mathcal{Q} \supseteq \mathcal{R}$, $\mathcal{P}' \subseteq \mathcal{P}$, $\mathcal{R}' \subseteq \mathcal{U}_{\mathcal{R}}(\mathcal{P})$, and either
 - there exists a monotonic ordinary reduction pair (\succsim, \succ) such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \mathcal{R}', \mathcal{U}_{\mathcal{R}}(\mathcal{P}), \mathcal{S}, \mathcal{E})$, or
 - there exists a monotonic $\text{Th}_{\mathbb{N}}$ -reduction pair (\succsim, \succ) for \mathcal{P} such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \mathcal{R}', \mathcal{U}_{\mathcal{R}}(\mathcal{P}), \mathcal{S}, \mathcal{E})$, or
 - there exists a monotonic $\text{Th}_{\mathbb{Z}}$ -reduction pair (\succsim, \succ) for \mathcal{P} such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \mathcal{R}', \mathcal{U}_{\mathcal{R}}(\mathcal{P}), \mathcal{S}, \mathcal{E})$.
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, otherwise.

Then Proc is sound.

Of course, a similar DP processor that considers all of \mathcal{R} instead of just the usable rule $\mathcal{U}_{\mathcal{R}}(\mathcal{P})$ would also be sound.

Example 8.9. Consider the following CERS which computes the size of a set:

$$\begin{array}{lcl}
 \mathcal{S} : & x + 0 & \rightarrow x \\
 & - - x & \rightarrow x \\
 & -0 & \rightarrow 0 \\
 & -(x + y) & \rightarrow (-x) + (-y) \\
 & x + (-x) & \rightarrow 0 \\
 & (x + (-x)) + y & \rightarrow 0 + y \\
 & \text{ins}(x, \text{ins}(x, ys)) & \rightarrow \text{ins}(x, ys)
 \end{array}$$

$$\begin{array}{l}
 \mathcal{E} : \quad x + y \approx y + x \\
 \quad \quad x + (y + z) \approx (x + y) + z \\
 \quad \quad \text{ins}(x, \text{ins}(y, zs)) \approx \text{ins}(y, \text{ins}(x, zs)) \\
 \mathcal{R} : \quad \text{remove}(\emptyset) \rightarrow \emptyset \\
 \quad \quad \text{remove}(\text{ins}(x, ys)) \rightarrow ys \\
 \quad \quad |\emptyset| \rightarrow 0 \\
 \quad \quad |\text{ins}(x, ys)| \rightarrow |\text{remove}(\text{ins}(x, ys))| + 1
 \end{array}$$

If $\mathcal{Q} = \mathcal{R}$, the estimated dependency graph $\text{EDG}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ contains one SCC and the DP problem $(\{|\text{ins}(x, ys)|^\# \rightarrow |\text{remove}(\text{ins}(x, ys))|^\#\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ needs to be handled. This cannot be done using an ordinary polynomial interpretation since the rewrite rule $\text{remove}(\text{ins}(x, ys)) \rightarrow ys$ forces the polynomial $\mathcal{P}ol(\text{remove})$ to depend on the variable x_1 . But then the dependency pair $|\text{ins}(x, ys)|^\# \rightarrow |\text{remove}(\text{ins}(x, ys))|^\#$ cannot be oriented strictly and a termination proof fails. Instead, it is possible to apply a monotonic ordinary reduction pair based on

$$\begin{array}{l}
 \mathcal{P}ol(+) = x_1 + x_2 \\
 \mathcal{P}ol(-) = x_1 \\
 \mathcal{P}ol(0) = 0 \\
 \mathcal{P}ol(| \cdot |^\#) = x_1 \\
 \mathcal{P}ol(\emptyset) = 0 \\
 \mathcal{P}ol(\text{ins}) = x_1 + x_2 + 1 \\
 \mathcal{P}ol(\text{remove}) = x_1
 \end{array}$$

Then $(\succsim_{\mathcal{P}ol}, \succ_{\mathcal{P}ol}) \models (\emptyset, \mathcal{P}, \mathcal{R}', \mathcal{R}, \mathcal{S}, \mathcal{E})$ is satisfied, where $\mathcal{R}' = \{\text{remove}(\text{ins}(x, ys)) \rightarrow ys\}$. After removing \mathcal{R}' from the DP problem, the resulting DP problem can easily be handled by the (non-monotonic) ordinary reduction pair based on the following polynomial interpretation:

$$\mathcal{P}ol(+) = x_1 + x_2$$

$$\begin{aligned}
 \mathcal{P}ol(-) &= x_1 \\
 \mathcal{P}ol(0) &= 0 \\
 \mathcal{P}ol(| \cdot |^\#) &= x_1 \\
 \mathcal{P}ol(\emptyset) &= 0 \\
 \mathcal{P}ol(\text{ins}) &= x_2 + 1 \\
 \mathcal{P}ol(\text{remove}) &= 0
 \end{aligned}$$

Using this polynomial interpretation, the (only) dependency pair is removed. \triangle

8.3 Function Dependencies

Recall that the DP processors from Chapter 7 have to consider all of \mathcal{R} , \mathcal{S} , and \mathcal{E} and that it suffices to consider the *usable rules* in the innermost case, i.e., if $\mathcal{Q} \supseteq \mathcal{R}$. It is well-known from the DP framework for ordinary rewriting that considering all of \mathcal{R} in the non-innermost case is a severe restriction that causes many termination proofs to fail [88, 95].

This section provides a proof that it also suffices to consider the usable rules in the general case. However, since the proof of Lemma 8.3 (and thus, Theorem 8.4) inherently depends on the innermost case and cannot be extended to the unrestricted case, the proof of this result is quite different. Indeed, the reductions between instantiated dependency pairs may use rewrite rules that are not usable in the innermost case since in general there are no assumptions on the substitution σ used for building chains other than the restriction that σ is *Th*-based.

The idea of the result in this section is to show that each $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain that uses a substitution σ can be transformed into a sequence that only uses subsets $\mathcal{R}' \subseteq \mathcal{R}$, $\mathcal{S}' \subseteq \mathcal{S}$, and $\mathcal{E}' \subseteq \mathcal{E}$ by considering a different substitution $\mathcal{I}(\sigma)$. This sequence will not necessarily be a $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain, but this property is not

needed for soundness. The set \mathcal{R}' turns out to coincide with the usable rules as defined in Definition 8.1, i.e., the technique developed in this section is indeed at least as powerful as the technique obtained with usable rules in practice. The only drawback is that the reduction pairs need to orient four additional rewrite rules for two fresh function symbols. This drawback is not severe in practice, however, since these rewrite rules can be easily oriented using polynomial interpretations.

Somewhat surprisingly, the DP processor developed in this section is, in practice, slightly *more* powerful than the DP processor of Section 8.1 that is based on usable rules. Therefore, the DP processor of this section should also be used in the innermost case since it subsumes the DP processor of Theorem 8.4. Recall that Theorem 8.4 needs to consider only the subset of usable rules from \mathcal{R} , but all rules from \mathcal{S} and all equations from \mathcal{E} . The DP processor of this section, in contrast, has to consider the same subset of \mathcal{R} , but additionally only subsets of \mathcal{S} and \mathcal{E} as well. The only restriction of the DP processor in this section is that it is only applicable if \mathcal{E} is size-preserving. Recall that all examples of data structures given in Chapter 3 satisfy this property.

The subsets $\mathcal{R}' \subseteq \mathcal{R}$, $\mathcal{S}' \subseteq \mathcal{S}$, and $\mathcal{E}' \subseteq \mathcal{E}$ that need to be considered are based on the dependencies between function symbols. Intuitively, a function symbol f depends on a function symbol h if h is below f in the recursion hierarchy defined by the rules and equations. Similar definitions to the one below are also used in [167, 88, 95, 64, 156].

Definition 8.10 (Function Dependencies). *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be a DP problem where \mathcal{E} is size-preserving. For two symbols $f, h \in \mathcal{F}$ let $f \blacktriangleright_{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})} h$ iff $f = h$ or there exists a symbol g with $g \blacktriangleright_{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})} h$ and either*

1. *a rule $l \rightarrow r \llbracket \varphi \rrbracket \in \mathcal{R}$ such that $\text{root}(l) = f$ and $g \in \mathcal{F}(r)$, or*
2. *a rule $l \rightarrow r \in \mathcal{S}$ such that $\text{root}(l) = f$ and $g \in \mathcal{F}(r)$, or*

3. an equation $u \approx v$ (or $v \approx u$) in \mathcal{E} such that $\text{root}(u) = f$ and $g \in \mathcal{F}(u \approx v)$.

In the following, let

$$\Delta(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) = \mathcal{F}_{Th} \cup \bigcup_{s \rightarrow t \llbracket \varphi \rrbracket \in \mathcal{P}} \{g \mid f \blacktriangleright_{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})} g \text{ for an } f \in \mathcal{F}(t) - \mathcal{F}^\#\}$$

Notice that this definition only makes sense if \mathcal{E} is collapse-free.

Example 8.11. Consider the following rules and equations:

$$\begin{aligned} f(s(x)) &\rightarrow g(x) \\ g(s(x)) &\rightarrow h(c(x)) \\ c(x) &\approx s(x) \end{aligned}$$

Using a simplified notation and omitting the reflexivity of the relation, $f \blacktriangleright g \blacktriangleright h$, $f \blacktriangleright g \blacktriangleright c$, $c \blacktriangleright s$, and $s \blacktriangleright c$. △

Subsets $\Delta \subseteq \mathcal{F} \cup \mathcal{F}_{Th}$ give rise to subsets of \mathcal{R} , \mathcal{S} , and \mathcal{E} in the obvious way.

Definition 8.12 ($\mathcal{R}(\Delta)$, $\mathcal{S}(\Delta)$, and $\mathcal{E}(\Delta)$). *Let $\Delta \subseteq \mathcal{F} \cup \mathcal{F}_{Th}$. Then $\mathcal{R}(\Delta) = \{l \rightarrow r \llbracket \varphi \rrbracket \in \mathcal{R} \mid \text{root}(l) \in \Delta\}$, $\mathcal{S}(\Delta) = \{l \rightarrow r \in \mathcal{S} \mid \text{root}(l) \in \Delta\}$, and $\mathcal{E}(\Delta) = \{u \approx v \in \mathcal{E} \mid \text{root}(u) \in \Delta \text{ or } \text{root}(v) \in \Delta\}$.*

In the following, a mapping \mathcal{I} from terminating terms¹ to terms that possibly contain fresh function symbols Π_{univ} and Π_{base} is defined. This mapping is similar to mappings defined in [88, 95, 156] but differs in how terms t with $\text{root}(t) \notin \Delta$ are handled. Furthermore, *two* fresh function symbols are needed, one for sort **univ** and one for sort **base**. The idea for this mapping is that every reduction of t that uses \mathcal{R} , \mathcal{S} , and \mathcal{E} can be “simulated” by a reduction of $\mathcal{I}(t)$ that only uses $\mathcal{R}(\Delta)$, $\mathcal{S}(\Delta)$, and $\mathcal{E}(\Delta)$. Within this section, it is assumed that $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ is a DP problem such that \mathcal{E} is size-preserving. Furthermore, let $\Delta = \Delta(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$.

¹A term t is *terminating* iff there are no infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}}$ -reductions starting with t .

Definition 8.13 (\mathcal{I}). For any terminating term $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{Th}, \mathcal{V})$ define $\mathcal{I}(t)$ by

$$\begin{aligned} \mathcal{I}(x) &= x && \text{if } x \in \mathcal{V} \\ \mathcal{I}(f(t_1, \dots, t_n)) &= f(\mathcal{I}(t_1), \dots, \mathcal{I}(t_n)) && \text{if } f \in \Delta \\ \mathcal{I}(t) &= \text{Comp}_{\text{sort}(t)}(\mathcal{R}ed_{\mathcal{S}}(t) \cup \mathcal{R}ed_{\mathcal{R}}(t) \cup \mathcal{E}q_{\mathcal{E}}(t)) && \text{otherwise} \end{aligned}$$

where the sets $\mathcal{R}ed_{\mathcal{S}}(t)$, $\mathcal{R}ed_{\mathcal{R}}(t)$, and $\mathcal{E}q_{\mathcal{E}}(t)$ are defined as

$$\begin{aligned} \mathcal{R}ed_{\mathcal{S}}(t) &= \{\mathcal{I}(t') \mid t \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \circ \sim_{\mathcal{E}} t'\} \\ \mathcal{R}ed_{\mathcal{R}}(t) &= \{\mathcal{I}(t') \mid t \xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}} \circ \sim_{\mathcal{E}} t'\} \\ \mathcal{E}q_{\mathcal{E}}(t) &= \{g(\mathcal{I}(t_1), \dots, \mathcal{I}(t_n)) \mid t \sim_{\mathcal{E}} g(t_1, \dots, t_n)\} \end{aligned}$$

For $s \in \{\text{univ}, \text{base}\}$, let $\text{Comp}_s(\{t\} \uplus M) = \Pi_s(t, \text{Comp}_s(M))$ and $\text{Comp}_s(\emptyset) = \perp_s$ where Π_s is a fresh function symbols with sort declaration $s \times s \rightarrow s$ and \perp_s is a fresh variable of sort s . In order to make this definition unambiguous, it is assumed that t is the minimal element of $\{t\} \uplus M$ w.r.t. some total well-founded order $>_{\mathcal{T}}$ on terms. For a terminating substitution² σ , define the substitution $\mathcal{I}(\sigma)$ by $\mathcal{I}(\sigma)(x) = \mathcal{I}(\sigma(x))$.

It is not immediately obvious that \mathcal{I} is indeed well-defined, i.e., that $\mathcal{I}(t)$ is a finite term whenever t is terminating. This can be shown to be the case, however.

Lemma 8.14. *If t is terminating, then $\mathcal{I}(t)$ is a finite term.*

In order to simulate reductions using \mathcal{R} , \mathcal{S} , and \mathcal{E} by reductions that only use $\mathcal{R}(\Delta)$, $\mathcal{S}(\Delta)$, and $\mathcal{E}(\Delta)$, the following rewrite rules for the fresh function symbols Π_{univ} and Π_{base} are needed.

Definition 8.15 (\mathcal{R}_{Π}). For the fresh function symbols Π_{univ} and Π_{base} from Definition 8.13, let $\mathcal{R}_{\Pi} = \{\Pi_s(x, y) \rightarrow x, \Pi_s(x, y) \rightarrow y \mid s \in \{\text{univ}, \text{base}\}\}$.

²A substitution σ is *terminating* iff $\sigma(x)$ is terminating for all $x \in \mathcal{V}$.

Chapter 8. Usable Rules and Function Dependencies

The following is a well-known simple property of \mathcal{R}_Π . It states that any element t of M can be extracted from the term $\text{Comp}_{\text{sort}(t)}(M)$.

Lemma 8.16. *Let M be a set of terms such that all elements of M have the same sort. If $t \in M$ then $\text{Comp}_{\text{sort}(t)}(M) \rightarrow_{\mathcal{R}_\Pi}^+ t$.*

Next, several properties of the mapping \mathcal{I} are shown that relate reductions using \mathcal{R} , \mathcal{S} , and \mathcal{E} to reductions using $\mathcal{R}(\Delta)$, $\mathcal{S}(\Delta)$, and $\mathcal{E}(\Delta)$.

Lemma 8.17. *Let $s, t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{Th}, \mathcal{V})$ and let σ be a Th -based substitution such that s , t , and $s\sigma$ are terminating.*

1. If $s \in \mathcal{T}(\Delta, \mathcal{V})$ then $\mathcal{I}(s\sigma) = s\mathcal{I}(\sigma)$.
2. $\mathcal{I}(s\sigma) \rightarrow_{\mathcal{R}_\Pi}^* s\mathcal{I}(\sigma)$.
3. If $s \sim_{\mathcal{E}} t$ then $\mathcal{I}(s) \sim_{\mathcal{E}(\Delta)} \mathcal{I}(t)$.
4. If $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}}^* t$ then $\mathcal{I}(s) \rightsquigarrow_1^* \mathcal{I}(t)$,
where $\rightsquigarrow_1 = \sim_{\mathcal{E}(\Delta)} \circ \rightarrow_{\mathcal{R}_\Pi}^* \circ \rightarrow_{\mathcal{S}(\Delta)} \cup \rightarrow_{\mathcal{R}_\Pi}^+$.
5. If $s \xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}}^* t$ then $\mathcal{I}(s) \rightsquigarrow_2^* \mathcal{I}(t)$,
where $\rightsquigarrow_2 = \rightsquigarrow_1^* \circ \sim_{\mathcal{E}(\Delta)} \circ \rightarrow_{\mathcal{R}_\Pi}^* \circ \rightarrow_{\mathcal{R}(\Delta)} \cup \rightarrow_{\mathcal{R}_\Pi}^+$ such that the $\rightarrow_{\mathcal{R}(\Delta)}$ -step uses a Th -based substitution that makes the instantiated constraint of the used rule Th -valid.
6. If $s \in \mathcal{T}(\Delta, \mathcal{V})$ and $s\sigma \xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}}^* \rightarrow_{\mathcal{E} \setminus \mathcal{S}}^! \circ \sim_{\mathcal{E}} t\sigma$, then $s\mathcal{I}(\sigma) \rightsquigarrow_2^* \circ \rightsquigarrow_1^* \circ \sim_{\mathcal{E}(\Delta)} \circ \rightarrow_{\mathcal{R}_\Pi}^* t\mathcal{I}(\sigma)$.

Using Lemma 8.17.6, soundness of the following DP processor can be shown. Notice that the rules from \mathcal{R}_Π can be easily oriented using reduction pairs based on polynomial interpretations by letting $\mathcal{P}ol(\Pi_{\text{univ}}) = \mathcal{P}ol(\Pi_{\text{base}}) = x_1^2 + x_2^2 + 1$.

Theorem 8.18 (DP Processor Based on Function Dependencies). *Let Proc be a DP processor with $\text{Proc}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) =$*

Chapter 8. Usable Rules and Function Dependencies

- $\{(\mathcal{P} - \mathcal{P}', \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, if \mathcal{E} is size-preserving, $\Delta = \Delta(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$, $\mathcal{P}' \subseteq \mathcal{P}$, and either
 - there exists an ordinary reduction pair (\succsim, \succ) such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \emptyset, \mathcal{R}(\Delta) \cup \mathcal{R}_{\Pi}, \mathcal{S}(\Delta), \mathcal{E}(\Delta))$, or
 - there exists a $Th_{\mathbb{N}}$ -reduction pair (\succsim, \succ) for \mathcal{P} such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \emptyset, \mathcal{R}(\Delta) \cup \mathcal{R}_{\Pi}, \mathcal{S}(\Delta), \mathcal{E}(\Delta))$, or
 - there exists a $Th_{\mathbb{Z}}$ -reduction pair (\succsim, \succ) for \mathcal{P} such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \emptyset, \mathcal{R}(\Delta) \cup \mathcal{R}_{\Pi, \text{univ}}, \mathcal{S}(\Delta), \mathcal{E}(\Delta))$.
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, otherwise.

Then Proc is sound.

Example 8.19. This example of the sieve of Eratosthenes is used to illustrate the ideas presented in this section:

$$\begin{array}{l}
 \mathcal{S} : \quad x + 0 \rightarrow x \\
 \quad \quad - - x \rightarrow x \\
 \quad \quad -0 \rightarrow 0 \\
 \quad \quad -(x + y) \rightarrow (-x) + (-y) \\
 \quad \quad x + (-x) \rightarrow 0 \\
 \quad \quad (x + (-x)) + y \rightarrow 0 + y \\
 \mathcal{E} : \quad x + y \approx y + x \\
 \quad \quad x + (y + z) \approx (x + y) + z \\
 \mathcal{R} : \quad \text{primes}(x) \rightarrow \text{sieve}(\text{nats}(2, x)) \\
 \quad \quad \text{nats}(x, y) \rightarrow \text{nil} \llbracket x > y \rrbracket \\
 \quad \quad \text{nats}(x, y) \rightarrow \text{cons}(x, \text{nats}(x + 1, y)) \llbracket x \not> y \rrbracket
 \end{array}$$

Chapter 8. Usable Rules and Function Dependencies

$$\begin{aligned}
\text{sieve}(\text{nil}) &\rightarrow \text{nil} \\
\text{sieve}(\text{cons}(x, ys)) &\rightarrow \text{cons}(x, \text{sieve}(\text{filter}(x, ys))) \\
\text{filter}(x, \text{nil}) &\rightarrow \text{nil} \\
\text{filter}(x, \text{cons}(y, zs)) &\rightarrow \text{if}(\text{isdiv}(x, y), x, y, zs) \\
\text{if}(\text{true}, x, y, zs) &\rightarrow \text{filter}(x, zs) \\
\text{if}(\text{false}, x, y, zs) &\rightarrow \text{cons}(y, \text{filter}(x, zs)) \\
\text{isdiv}(x, 0) &\rightarrow \text{true} \llbracket x > 0 \rrbracket \\
\text{isdiv}(x, y) &\rightarrow \text{false} \llbracket x > y \wedge y > 0 \rrbracket \\
\text{isdiv}(x, y) &\rightarrow \text{isdiv}(x, y - x) \llbracket y \geq x \wedge x > 0 \rrbracket
\end{aligned}$$

Then, the following dependency pairs are obtained:

$$\text{primes}^\sharp(x) \rightarrow \text{sieve}^\sharp(\text{nats}(2, x)) \quad (8.1)$$

$$\text{primes}^\sharp(x) \rightarrow \text{nats}^\sharp(2, x) \quad (8.2)$$

$$\text{nats}^\sharp(x, y) \rightarrow \text{nats}^\sharp(x + 1, y) \llbracket x \not> y \rrbracket \quad (8.3)$$

$$\text{sieve}^\sharp(\text{cons}(x, ys)) \rightarrow \text{sieve}^\sharp(\text{filter}(x, ys)) \quad (8.4)$$

$$\text{sieve}^\sharp(\text{cons}(x, ys)) \rightarrow \text{filter}^\sharp(x, ys) \quad (8.5)$$

$$\text{filter}^\sharp(x, \text{cons}(y, zs)) \rightarrow \text{if}^\sharp(\text{isdiv}(x, y), x, y, zs) \quad (8.6)$$

$$\text{filter}^\sharp(x, \text{cons}(y, zs)) \rightarrow \text{isdiv}^\sharp(x, y) \quad (8.7)$$

$$\text{if}^\sharp(\text{true}, x, y, zs) \rightarrow \text{filter}^\sharp(x, zs) \quad (8.8)$$

$$\text{if}^\sharp(\text{false}, x, y, zs) \rightarrow \text{filter}^\sharp(x, zs) \quad (8.9)$$

$$\text{isdiv}^\sharp(x, y) \rightarrow \text{isdiv}^\sharp(x, y - x) \llbracket y \geq x \wedge x > 0 \rrbracket \quad (8.10)$$

The estimated dependency graph contains four SCCs, and according to Theorem 6.7, the following DP problems are obtained:

$$(\{(8.3)\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) \quad (8.11)$$

$$(\{(8.4)\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) \quad (8.12)$$

$$(\{(8.6), (8.8), (8.9)\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) \quad (8.13)$$

$$(\{(8.10)\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) \tag{8.14}$$

The DP problem (8.13) can be handled by the subterm criterion of Theorem 6.24 with the simple projection $\pi(\text{filter}^\sharp) = 2$ and $\pi(\text{if}^\sharp) = 4$. Then, the dependency pair (8.6) is removed and the dependency pairs (8.8) and (8.9) do not form an SCC.

Notice that $\Delta(8.11) = \Delta(8.14) = \mathcal{F}_{\mathcal{T}h}$ and thus $\mathcal{R}(\Delta) = \emptyset$ for these DP problems. They can be handled using a $\mathcal{T}h_{\mathbb{Z}}$ -polynomial interpretation with $c_{\mathcal{P}ol} = 0$, $\mathcal{P}ol(\text{isdiv}^\sharp) = x_1 + x_2$, and $\mathcal{P}ol(\text{nats}^\sharp) = x_2 - x_1$.

For (8.12), $\Delta = \mathcal{F}_{\mathcal{T}h} \cup \{\text{filter}, \text{if}, \text{isdiv}, \text{cons}, \text{nil}, \text{true}, \text{false}\}$, and $\mathcal{R}(\Delta)$ thus consists of the filter-, if-, and isdiv-rules. Apply Theorem 8.18 and the following $\mathcal{T}h_{\mathbb{Z}}$ -polynomial interpretations with $c_{\mathcal{P}ol} = 0$:

$$\begin{aligned} \mathcal{P}ol(\text{sieve}^\sharp) &= x_1 \\ \mathcal{P}ol(\text{filter}) &= x_2 \\ \mathcal{P}ol(\text{if}) &= x_4 + 1 \\ \mathcal{P}ol(\text{isdiv}) &= 0 \\ \mathcal{P}ol(\text{cons}) &= x_2 + 1 \\ \mathcal{P}ol(\text{nil}) &= 0 \\ \mathcal{P}ol(\text{true}) &= 0 \\ \mathcal{P}ol(\text{false}) &= 0 \end{aligned}$$

Then, all dependency pairs are removed from the DP problem (8.12). △

8.4 Removal of Rules Revisited

The DP processor from Section 8.2 that is based on monotonic reduction pairs and may remove rules from \mathcal{R} in addition to dependency pairs extends to the unrestricted

case as well. If \mathcal{E} is size-preserving, the sets $\mathcal{R}(\Delta)$, $\mathcal{S}(\Delta)$, and $\mathcal{E}(\Delta)$ from the previous section can be used. Notice that \mathcal{R}_Π now has to be oriented using \succ , i.e., the rules in \mathcal{R}_Π need to be strictly decreasing. This is always the case if the above-mentioned polynomials $\mathcal{P}ol(\Pi_{\text{univ}}) = \mathcal{P}ol(\Pi_{\text{base}}) = x_1^2 + x_2^2 + 1$ are used.

Theorem 8.20 (DP Processor Based on Removal of Rules and Function Dependencies). *Let Proc be a DP processor with $\text{Proc}(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}) =$*

- $\{(\mathcal{P} - \mathcal{P}', \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, if \mathcal{E} is size-preserving, $\Delta = \Delta(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$, $\mathcal{P}' \subseteq \mathcal{P}$, $\mathcal{R}' \subseteq \mathcal{R}(\Delta)$, and either
 - there exists a monotonic ordinary reduction pair (\succsim, \succ) such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \mathcal{R}' \cup \mathcal{R}_\Pi, \mathcal{R}(\Delta), \mathcal{S}(\Delta), \mathcal{E}(\Delta))$, or
 - there exists a monotonic $\text{Th}_{\mathbb{N}}$ -reduction pair (\succsim, \succ) for \mathcal{P} such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \mathcal{R}' \cup \mathcal{R}_\Pi, \mathcal{R}(\Delta), \mathcal{S}(\Delta), \mathcal{E}(\Delta))$, or
 - there exists a monotonic $\text{Th}_{\mathbb{Z}}$ -reduction pair (\succsim, \succ) for \mathcal{P} such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \mathcal{R}' \cup \mathcal{R}_{\Pi, \text{univ}}, \mathcal{R}(\Delta), \mathcal{S}(\Delta), \mathcal{E}(\Delta))$.
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, otherwise.

Then Proc is sound.

If \mathcal{E} is not size-preserving, then a similar DP processor can be obtained that has to consider all of \mathcal{R} , \mathcal{S} , and \mathcal{E} , but can disregard \mathcal{R}_Π .

8.5 Summary

This chapter has presented techniques that extend the techniques from Chapter 7. Similar to the techniques from Chapter 7, these new techniques are based on reduction pairs.

Chapter 8. Usable Rules and Function Dependencies

First, the concept of usable rules has been introduced for the innermost case. Here, the usable rules are a superset of the rules that may be used in a chain, and it has been shown that it is possible to disregard the non-usable rules from \mathcal{R} since they cannot be used for building chains. Next, it was shown that reduction pairs can also be used to remove usable rules from a DP problem, provided that the reduction pair is monotonic.

Extending the idea of usable rules to the non-innermost case is quite complex since the soundness proof of the usable rule extension inherently depends on the assumption that $\mathcal{Q} \supseteq \mathcal{R}$. Nonetheless, this chapter has presented a technique that is, in practice, slightly *stronger* than the usable rule extension. This technique is not restricted to the innermost case and still makes it possible to consider only syntactically determined subsets of the rules and equations.

Chapter 9

Implementation

The techniques presented so far have been implemented in the automated termination checker AProVE [84] for $\mathcal{Th}_{\mathbb{Z}}$ and the general case (i.e., $\mathcal{Q} \supseteq \mathcal{R}$ is not assumed). While most of the implementation is straightforward, an implementation of the estimated dependency graph EDG is non-trivial. The same is true for the automated generation of $\mathcal{Th}_{\mathbb{Z}}$ -reduction pairs based on $\mathcal{Th}_{\mathbb{Z}}$ -polynomial interpretations.

9.1 Implementing EDG

It is not clear whether Definition 6.9 can be implemented in general, even if the checks for irreducibility by $\xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}}$ and \mathcal{Th} -validity are omitted, since it needs to be determined whether there exists a substitution μ satisfying $s\mu \xrightarrow{\geq \Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} t\mu$ for two terms s and t . In the following, two approximations for checking this are presented.

First Approximation: Checking whether there exists a substitution μ satisfying $s\mu \xrightarrow{\geq \Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} t\mu$ can be approximated by checking whether s and t are $\mathcal{E} \cup \mathcal{S}$ -unifiable. This approximation can be applied if $\mathcal{E} \cup \mathcal{S}$ -unifiability is decidable and a

Chapter 9. Implementation

decision procedure has been implemented.

Lemma 9.1. *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CERS and let s and t be terms. If there exists a substitution μ such that $s\mu \xrightarrow{\geq \Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \circ \overset{\geq \Lambda}{\sim}_{\mathcal{E}} t\mu$, then s and t are $\mathcal{E} \cup \mathcal{S}$ -unifiable.*

Thus, if s and t are not $\mathcal{E} \cup \mathcal{S}$ -unifiable, then there does not exist a substitution μ such that $s\mu \xrightarrow{\geq \Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \circ \overset{\geq \Lambda}{\sim}_{\mathcal{E}} t\mu$. The first approximation is particularly useful if no collection data structures are used and $\mathcal{E} \cup \mathcal{S}$ -unifiability reduces to $\mathcal{Th}_{\mathbb{Z}}$ -unifiability. If the terms under consideration have the form $f(s_1, \dots, s_n)$ with $\text{sort}(s_i) = \text{base}$ and $s_i \in \mathcal{T}(\mathcal{F}_{\mathcal{Th}_{\mathbb{Z}}}, \mathcal{V})$ for all $1 \leq i \leq n$, then $\mathcal{Th}_{\mathbb{Z}}$ -unifiability is decidable since it reduces to determining whether a system of linear equations is solvable in \mathbb{Z} . This can be decided using SMT-solvers¹ such as Yices [62].

Example 9.2. Let $s_1 = f(x + y + 3, y)$ and $t_1 = f(-x - 4y - 7, x)$. In order to determine whether s_1 and t_1 are $\mathcal{Th}_{\mathbb{Z}}$ -unifiable, it suffices to check whether the system $x + y + 3 = -x - 4y - 7$, $y = x$ is solvable in \mathbb{Z} (which is the case).

Now let $s_2 = f(x + y, x - y)$ and $t_2 = f(0, 1)$. Since s_2 and t_2 are not $\mathcal{Th}_{\mathbb{Z}}$ -unifiable because the system $x + y = 0$, $x - y = 1$ is not solvable, there does not exist a substitution μ such that $s_2\mu \xrightarrow{\geq \Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \circ \overset{\geq \Lambda}{\sim}_{\mathcal{E}} t_2\mu$. △

Furthermore, unifiability is also decidable for the list-like representation of compact lists, multisets, and sets (i.e., the representation using `nil` and `cons` or \emptyset and `ins`, see Chapter 3) [58]. These unification algorithms are, however, not currently implemented in AProVE. Notice that it is also necessary to implement a combination framework for unification algorithms [152, 32, 18] if a combination of integers with one or more collection data structures is used in a CERS.

Second Approximation: The approximation from Lemma 9.1 does not take the directionality of \mathcal{S} into account. The second approximation takes the directionality

¹SMT stands for *satisfiability modulo theories*, see, e.g., [154, 23].

Chapter 9. Implementation

of \mathcal{S} into account but is only applicable if \mathcal{E} is collapse-free. The idea for this approximation is to replace subterms of s that might become reducible by $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ by fresh variables and then check for \mathcal{E} -unifiability. This is of course very similar to the idea that is used in the definition of EDG. This second approximation can be applied if \mathcal{E} -unifiability is decidable which is, e.g., the case if \mathcal{E} only specifies that certain function symbols are associative and commutative.

Lemma 9.3. *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CERS such that \mathcal{E} is collapse-free and let s and t be terms. If there exists a substitution μ such that $s\mu \xrightarrow{>\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \circ \overset{>\Lambda}{\sim}_{\mathcal{E}} t\mu$, then $\text{CAP}_{\mathcal{S}}(s)$ and t are \mathcal{E} -unifiable. Here, $\text{CAP}_{\mathcal{S}}$ is defined by*

1. $\text{CAP}_{\mathcal{S}}(x) = y$ for variables x ,
2. $\text{CAP}_{\mathcal{S}}(f(t_1, \dots, t_n)) = f(\text{CAP}_{\mathcal{S}}(t_1), \dots, \text{CAP}_{\mathcal{S}}(t_n))$ if there does not exist a rule $l \rightarrow r \in \mathcal{S}$ such that $f(\text{CAP}_{\mathcal{S}}(t_1), \dots, \text{CAP}_{\mathcal{S}}(t_n))$ and l are \mathcal{E} -unifiable, and
3. $\text{CAP}_{\mathcal{S}}(f(t_1, \dots, t_n)) = y$ otherwise.

Here, y is the next variable in an infinite list y_1, y_2, \dots of fresh variables.

Thus, if $\text{CAP}_{\mathcal{S}}(s)$ and t are not \mathcal{E} -unifiable, then there does not exist a substitution μ such that $s\mu \xrightarrow{>\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \circ \overset{>\Lambda}{\sim}_{\mathcal{E}} t\mu$. As in Definition 6.9, case 2 can be replaced by a check for $f \notin \mathcal{D}(\mathcal{S}, \mathcal{E})$, where $\mathcal{D}(\mathcal{S}, \mathcal{E})$ is the smallest set with $\mathcal{D}(\mathcal{S}, \mathcal{E}) = \{\text{root}(l) \mid l \rightarrow r \in \mathcal{S}\} \cup \{\text{root}(v) \mid u \approx v \in \mathcal{E} \text{ or } v \approx u \in \mathcal{E}, \text{root}(u) \in \mathcal{D}(\mathcal{S}, \mathcal{E})\}$ (these are the “defined symbols” of \mathcal{S}).

Example 9.4. Let $\mathcal{E} = \{x \cup y \approx y \cup x, x \cup (y \cup z) \approx (x \cup y) \cup z\}$ and $\mathcal{S} = \{x \cup \emptyset \rightarrow x\}$. Let $s_1 = f(\mathbf{g}(x), y \cup z)$ and $t_1 = f(\mathbf{h}(0), w)$. Then $\text{CAP}_{\mathcal{S}}(s_1) = f(\mathbf{g}(z_1), z_2)$ since $\text{CAP}_{\mathcal{S}}(x \cup y) = z_3 \cup z_4$ is \mathcal{E} -unifiable with the left-hand side $x \cup \emptyset$ of the rule from \mathcal{S} . Since $f(\mathbf{g}(z_1), z_2)$ and $f(\mathbf{h}(0), w)$ are not \mathcal{E} -unifiable, there does not exist a substitution μ such that $s_1\mu \xrightarrow{>\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \circ \overset{>\Lambda}{\sim}_{\mathcal{E}} t_1\mu$.

Now let $s_2 = f(x, \mathbf{g}(x \cup y))$ and $t_2 = f(\mathbf{h}(y), x)$. Then $\text{CAP}_{\mathcal{S}}(s_2) = f(z_1, \mathbf{g}(z_2))$

and the terms $f(z_1, g(z_2))$ and t_2 are \mathcal{E} -unifiable even though there does not exist a substitution μ such that $s_2\mu \xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}} t_2\mu$. This demonstrates the limitations of the approximation from Lemma 9.3. \triangle

The second approximation is helpful even if \mathcal{E} -unifiability is not decidable since \mathcal{E} -unifiability for collapse-free equations can be approximated by syntactic unifiability using an approach similar to Lemma 9.3 [64].

9.2 Generation of $\mathcal{Th}_{\mathbb{Z}}$ -Polynomial Interpretations

The following discussion concentrates on the generation of linear $\mathcal{Th}_{\mathbb{Z}}$ -polynomial interpretations. This is also the only case that is currently supported by the implementation in AProVE. The ideas presented in this section extend to the non-linear case as well, however. The generation of polynomial interpretations in the presence of conditions as considered in the related work [75] faces problems that are very similar to the ones encountered in the generation of $\mathcal{Th}_{\mathbb{Z}}$ -polynomial interpretations. Indeed, the method developed in [75] is partially based on the method presented in this section. In particular, the inference rule (A) from [75] has been derived from the inference rule **Express** stated below.

As for regular polynomial interpretations [48], the automated search starts with a parametric linear $\mathcal{Th}_{\mathbb{Z}}$ -polynomial interpretation, i.e., a linear $\mathcal{Th}_{\mathbb{Z}}$ -polynomial interpretation where the coefficients in the polynomials are parameters that have to be determined. Thus, $\mathcal{Pol}(f) = f_1x_1 + \dots + f_nx_n + c_f$ for each function symbol f of arity n , where the f_i and c_f are parameters.

Recall that the goal of finding a $\mathcal{Th}_{\mathbb{Z}}$ -polynomial interpretation is to instantiate the parameters in such a way that $s[\varphi] \bowtie t[\varphi]$ for certain terms s, t and $\mathcal{Th}_{\mathbb{Z}}$ -constraints φ , where $\bowtie \in \{\succ_{\mathcal{Pol}}, \sim_{\mathcal{Pol}}\}$.

Chapter 9. Implementation

In order to simplify presentation, it is assumed in this section that the constraints of all constrained rewrite rules are conjunctions of atomic $\mathcal{Th}_{\mathbb{Z}}$ -constraints. This can always be achieved by a conversion into disjunctive normal form (DNF) and the introduction of one constraint rewrite rule for each dual clause in this DNF.² This clearly results in an equivalent set of rewrite rules.

For the sake of concreteness, it is assumed that the $\mathcal{Th}_{\mathbb{Z}}$ -polynomial interpretation \mathcal{Pol} is restricted such that for each $f \in \mathcal{F}$, the polynomial $\mathcal{Pol}(f)$ may only depend on a variable x_i if the i^{th} argument of f has sort **univ** (see the discussion on this in Section 7.3 and recall that this assumption implies that it suffices to instantiate the variables of sort **univ** only by non-negative numbers when comparing polynomials).³ This is achieved by fixing the coefficient f_i of x_i to be 0 whenever the i^{th} argument of f has sort **base**. Furthermore, it is assumed that the value $c_{\mathcal{Pol}}$ is fixed to be 0. Alternatively, $c_{\mathcal{Pol}}$ can be treated as an additional parameter that needs to be determined.

As shown in Section 7.3, it suffices to instantiate the parameters such that

1. whenever $s[\varphi] \succ_{\mathcal{Pol}} t[\varphi]$ needs to be satisfied, then the following formulas are true in the integers:

$$\begin{aligned} & \forall x_1, \dots, x_k. \forall y_1 \geq 0, \dots, y_l \geq 0. \varphi \Rightarrow [s]_{\mathcal{Pol}} \geq 0 \\ & \forall x_1, \dots, x_k. \forall y_1 \geq 0, \dots, y_l \geq 0. \varphi \Rightarrow [s]_{\mathcal{Pol}} - [t]_{\mathcal{Pol}} > 0 \end{aligned}$$

2. whenever $s[\varphi] \succeq_{\mathcal{Pol}} t[\varphi]$ needs to be satisfied, then

$$\forall x_1, \dots, x_k. \forall y_1 \geq 0, \dots, y_l \geq 0. \varphi \Rightarrow [s]_{\mathcal{Pol}} - [t]_{\mathcal{Pol}} \geq 0$$

is true in the integers, and

²Recall that a formula in DNF is a disjunction of conjunctions. The conjunctions are called *dual clauses*. Thus, in $(x \wedge y) \vee z$, the dual clauses are $x \wedge y$ and z .

³The case where this assumption is not imposed is slightly simpler. The implementation in AProVE supports both possibilities, and there are CERSs that can be shown terminating with one of the possibilities but not with the other.

Chapter 9. Implementation

3. whenever $s[\![\varphi]\!] \sim_{\mathcal{P}ol} t[\![\varphi]\!]$ needs to be satisfied, then

$$\forall x_1, \dots, x_k. \forall y_1 \geq 0, \dots, y_l \geq 0. \varphi \Rightarrow [s]_{\mathcal{P}ol} - [t]_{\mathcal{P}ol} = 0$$

is true in the integers.

Here, x_1, \dots, x_k are the variables of sort **base** in s and t and y_1, \dots, y_l are the variables of sort **univ** in s and t .

$[s]_{\mathcal{P}ol}$ and $[s]_{\mathcal{P}ol} - [t]_{\mathcal{P}ol}$ are linear polynomials whose coefficients are polynomials over the parameters. Polynomials of this shape are called *linear parametric polynomials*. Notice that the polynomials over the parameters that are the coefficients of linear parametric polynomials may be non-linear due to nested function symbols in s and t .

Example 9.5. Consider a parametric linear $\mathcal{Th}_{\mathbb{Z}}$ -polynomial interpretation with $\mathcal{P}ol(\mathbf{f}) = f_1x_1 + f_2x_2 + c_f$ and $\mathcal{P}ol(\mathbf{g}) = g_1x_1 + c_g$. Then $[\mathbf{f}(\mathbf{g}(x), \mathbf{g}(y))]_{\mathcal{P}ol} = f_1(g_1x + c_g) + f_2(g_1y + c_g) + c_f = f_1g_1x + f_2g_1y + f_1c_g + f_2c_g + c_f. \quad \triangle$

In the following, case 2 from above is discussed in detail. The cases 1 and 3 can be handled similarly. Thus, the problem that needs to be solved is as follows: Given a $\mathcal{Th}_{\mathbb{Z}}$ -constraint φ and a linear parametric polynomial p , determine values for the parameters such that

$$\forall x_1, \dots, x_k. \forall y_1 \geq 0, \dots, y_l \geq 0. \varphi \Rightarrow p \geq 0$$

is true in the integers after the parameters have been instantiated. For this, sufficient conditions on the parameters are derived and it is checked whether these conditions are satisfiable. Furthermore, any satisfying assignment to the conditions on the parameters gives rise to a suitable polynomial interpretation.

The conditions on the parameters are obtained as follows:

1. φ is transformed into a conjunction of atomic $\mathcal{Th}_{\mathbb{Z}}$ -constraints of the form $\sum_{i=1}^n a_i x_i + c \geq 0$ where $a_1, \dots, a_n, c \in \mathbb{Z}$.

Chapter 9. Implementation

2. The $\mathcal{Th}_{\mathbb{Z}}$ -constraints from step 1 are used to derive upper and/or lower bounds on the variables in p .
3. The bounds from step 2 are used to derive conditions on the parameters.

Step 1: Transformation of φ . This is straightforward: $s \simeq t$ is transformed into $s - t \geq 0 \wedge t - s \geq 0$, $s \geq t$ is transformed into $s - t \geq 0$, and $s > t$ is transformed into $s - t - 1 \geq 0$.

Step 2: Deriving upper and/or lower bounds. The $\mathcal{Th}_{\mathbb{Z}}$ -constraints obtained after step 1 may already contain upper and/or lower bounds on the variables, where a lower bound has the form $x + c \geq 0$ and an upper bound has the form $-x + c \geq 0$ for some $c \in \mathbb{Z}$. Otherwise, it might be possible to obtain such bounds as follows.

An atomic constraint of the form $ax + c \geq 0$ with $a \neq 0, 1, -1$ that contains only one variable gives a bound on that variable that can be obtained by dividing by $|a|$ and taking the integer floor of the divided constant⁴. For example, the $\mathcal{Th}_{\mathbb{Z}}$ -constraint $2x + 3 \geq 0$ is transformed into $x + 1 \geq 0$, and $-3x - 2 \geq 0$ is transformed into $-x - 1 \geq 0$.

An atomic $\mathcal{Th}_{\mathbb{Z}}$ -constraint with more than one variable can be used to express a variable x occurring with coefficient 1 in terms of the other variables and a fresh slack variable w with $w \geq 0$. This makes it possible to eliminate x from the polynomial p and at the same time gives the lower bound 0 on the slack variable w . For example, $x - 2y \geq 0$ can be used to eliminate the variable x by replacing it with $2y + w$. Similar reasoning applies if the variable x occurs with coefficient -1 .

These ideas are formalized in the transformation rules (to be read from top to bottom) from Figure 9.1 that operate on triples $\langle \varphi_1, \varphi_2, q \rangle$, where φ_1 and φ_2 are sets

⁴For any real number x , the integer floor is defined by $\lfloor x \rfloor := \max \{n \in \mathbb{Z} \mid n \leq x\}$.

| | | |
|--|--|--|
| $\text{Strengthen } \frac{\langle \varphi_1, \varphi_2 \uplus \{a_i x_i + c \geq 0\}, q \rangle}{\langle \varphi_1 \cup \left\{ \frac{a_i}{ a_i } x_i + \lfloor \frac{c}{ a_i } \rfloor \geq 0 \right\}, \varphi_2, q \rangle} \quad \text{if } a_i \neq 0$ | | |
| <table style="width: 100%; border: none;"> <tr> <td style="vertical-align: middle; padding-right: 10px;"> $\text{Express } \frac{\langle \varphi_1, \varphi_2 \uplus \{ \sum_{i=1}^n a_i x_i + c \geq 0 \}, q \rangle}{\langle \varphi_1 \cup \{w \geq 0\}, \varphi_2 \sigma, q \sigma \rangle}$ </td> <td style="vertical-align: middle; padding-left: 10px;"> if $a_j = 1$, there exists a $j' \neq j$ with $a_{j'} \neq 0$, and σ replaces x_j by $-\sum_{i \neq j} a_j a_i x_i - a_j c + a_j w$ </td> </tr> </table> | $\text{Express } \frac{\langle \varphi_1, \varphi_2 \uplus \{ \sum_{i=1}^n a_i x_i + c \geq 0 \}, q \rangle}{\langle \varphi_1 \cup \{w \geq 0\}, \varphi_2 \sigma, q \sigma \rangle}$ | if $ a_j = 1$, there exists a $j' \neq j$ with $a_{j'} \neq 0$, and σ replaces x_j by $-\sum_{i \neq j} a_j a_i x_i - a_j c + a_j w$ |
| $\text{Express } \frac{\langle \varphi_1, \varphi_2 \uplus \{ \sum_{i=1}^n a_i x_i + c \geq 0 \}, q \rangle}{\langle \varphi_1 \cup \{w \geq 0\}, \varphi_2 \sigma, q \sigma \rangle}$ | if $ a_j = 1$, there exists a $j' \neq j$ with $a_{j'} \neq 0$, and σ replaces x_j by $-\sum_{i \neq j} a_j a_i x_i - a_j c + a_j w$ | |

Figure 9.1: Transformation rules for the generation of polynomial interpretations.

of atomic $\mathcal{Th}_{\mathbb{Z}}$ -constraints and q is a linear parametric polynomial. Here, φ_1 only contains $\mathcal{Th}_{\mathbb{Z}}$ -constraints of the form $\pm x_i + c \geq 0$ giving upper and/or lower bounds on the variable x_i and φ_2 contains arbitrary atomic $\mathcal{Th}_{\mathbb{Z}}$ -constraints. The initial triple is $\langle \emptyset, \varphi, p \rangle$.

Step 3: Deriving conditions on the parameters. After finishing step 2, a final triple $\langle \varphi_1, \varphi_2, q \rangle$ is obtained. If φ_1 contains more than one bound on a variable x_i , then it suffices to consider the maximal lower bound and the minimal upper bound. The bounds in φ_1 are used in combination with reasoning similar to the absolute positiveness test⁵ [96] in order to obtain conditions on the parameters that make $q = \sum_{i=1}^k p_i x_i + \sum_{j=1}^l p'_j y_j + p_0$ non-negative for all instantiations of the x_i satisfying φ_1 and all $y_j \geq 0$.

If φ_1 contains a lower bound of the form $x_j + c \geq 0$ for the variable x_j , then notice that $q = \sum_{i=1}^k p_i x_i + \sum_{i=1}^l p'_i y_i + p_0$ can also be written as $q = \sum_{i \neq j} p_i x_i + p_j(x_j + c) + \sum_{i=1}^l p'_i y_i + p_0 - p_j c$. Since $x_j + c \geq 0$ is assumed, the absolute positiveness test requires $p_j \geq 0$ as a condition on p_j .⁶ Similarly, if $-x_j + c \geq 0$ occurs in φ_1 , then q

⁵For $p = p_1 x_1 + \dots + p_n x_n + c$ such that x_1, \dots, x_n may only be instantiated by natural numbers, the absolute positiveness test concludes that p is non-negative if $p_1, \dots, p_n, c \geq 0$.

⁶Alternatively, reasoning similar to the rule **Express** can be used, i.e., if φ_1 contains $x_j + c \geq 0$, then x_j could be replaced by $-c + w$, where $w \geq 0$. Both methods produce the same conditions on the parameters.

Chapter 9. Implementation

can be written as $q = \sum_{i \neq j} p_i x_i - p_j(-x_j + c) + \sum_{i=1}^l p'_i y_i + p_0 + p_j c$ and $-p_j \geq 0$ is obtained as a condition on p_j . If φ_1 does not contain any upper or lower bound on a variable x_j , then $p_j = 0$ is obtained by the absolute positiveness test. For any of the variables y_i it is required that $p'_i \geq 0$ is true since y_i is known to be instantiated by a non-negative integer.

After all variables of q have been processed in this fashion, it additionally needs to be required that the constant term of the final polynomial is non-negative as well. Notice that this constant term is not just p_0 in general since the above transformation modifies the constant term.

Example 9.6. If $\varphi_1 = \{x + 1 \geq 0, -y - 1 \geq 0\}$ and $q = (a + b)x + by + c$ where x and y range over the integers, then q can also be written as $q = (a + b)(x + 1) - b(-y - 1) + c - (a + b) - b$ and the absolute positiveness test requires $a + b \geq 0$, $-b \geq 0$, and $c - a - 2b \geq 0$ as conditions on the parameters. \triangle

Summarizing this method, the algorithm from Figure 9.2 is used in order to obtain conditions D on the parameters. Here, $\text{sign}(\psi) = 1$ if ψ is of the form $x_i + c \geq 0$ and $\text{sign}(\psi) = -1$ if ψ is of the form $-x_i + c \geq 0$.

Once conditions on the parameters have been established, existing methods for solving non-linear Diophantine constraints [48, 73] can be used. Since it is in general undecidable whether non-linear Diophantine constraints have a solution, these methods require a finite range $[0..m]$ for some $m \geq 0$ and determine whether the non-linear Diophantine constraints have a solution within this range. Recall, however, that some parameters may be instantiated by negative numbers. Replacing these parameters f_i in the non-linear Diophantine constraints by $f_i - \lfloor \frac{m}{2} \rfloor$ effectively shifts the search space for f_i to the finite range $[-\lfloor \frac{m}{2} \rfloor, m - \lfloor \frac{m}{2} \rfloor]$.

Example 9.7. This example illustrates the method discussed above on a simple example. Assume that $\varphi = x - y - z \geq 0$ and that the parameters a, b, c, d are to be

```

D := true
r := p0
for 1 ≤ i ≤ k do
  take constraint ψ of the form ±xi + c ≥ 0 from φ1
  if none such ψ exists then
    D := D ∧ pi ≈ 0
  else
    D := D ∧ sign(ψ) · pi ≥ 0
    r := r − sign(ψ) · c · pi
  end if
end for
D := D ∧ r ≥ 0
for 1 ≤ i ≤ l do
  D := D ∧ p'i ≥ 0
end for

```

Figure 9.2: Obtaining conditions on the parameters.

instantiated such that

$$\forall x, y, z. \varphi \Rightarrow ax + by + cz + d \geq 0$$

$$\forall x, y, z. \varphi \Rightarrow -b - c > 0$$

are true in the integers (these formulas are obtained from attempting to satisfy $f(x, y, z)[x \geq y + z] \succ_{pol} f(x, y + 1, z + 1)[x \geq y + z]$).

Starting with the initial triple $\langle \emptyset, \varphi, ax + by + cz + d \rangle$ for the first formula, a transformation sequence that exhaustively applies the transformation rules from Figure 9.1 is as follows:

$$\text{Express} \frac{\emptyset, \{x - y - z \geq 0\}, ax + by + cz + d}{\{w \geq 0\}, \emptyset, (a + b)y + (a + c)z + aw + d}$$

Here, the Express-step uses $\sigma = \{x \mapsto y + z + w\}$. The absolute positiveness test now returns $a + b \simeq 0 \wedge a + c \simeq 0 \wedge a \geq 0 \wedge d \geq 0$ as conditions on the parameters.

Chapter 9. Implementation

For the second formula, $-b - c > 0$ is immediately obtained as a condition on the parameters. The final conditions on the parameters are thus

$$a + b \simeq 0 \wedge a + c \simeq 0 \wedge a \geq 0 \wedge d \geq 0 \wedge -b - c > 0$$

These conditions are satisfiable and $a = 1, b = -1, c = -1, d = 0$ is a possible solution (the polynomial interpretation obtained from these parameters is $\mathcal{Pol}(f) = x_1 - x_2 - x_3$ and indeed satisfies $f(x, y, z) \llbracket x \geq y + z \rrbracket \succ_{\mathcal{Pol}} f(x, y + 1, z + 1) \llbracket x \geq y + z \rrbracket$). \triangle

Automatically finding strictly decreasing dependency pairs or rules. For the DP processors based on reduction pairs, it also needs to be ensured that the set \mathcal{P}' of dependency pairs that are decreasing w.r.t. $\succ_{\mathcal{Pol}}$ is non-empty. Furthermore, this set should be efficiently determined automatically. Let $s_i \rightarrow t_i \llbracket \varphi_i \rrbracket$ be the dependency pairs of a DP problem for $1 \leq i \leq n$. Then a non-empty set \mathcal{P}' (if it exists) can easily be found automatically by requiring the condition

$$\bigwedge_{i=1}^n D_1^i \wedge \bigvee_{i=1}^n D_2^i$$

on the parameters. Here, D_1^i are the conditions on the parameters obtained for making $s_i \llbracket \varphi_i \rrbracket \succeq_{\mathcal{Pol}} t_i \llbracket \varphi_i \rrbracket$ true, and D_2^i are the conditions on the parameters obtained for making $s_i \llbracket \varphi_i \rrbracket \succ_{\mathcal{Pol}} t_i \llbracket \varphi_i \rrbracket$ true. Thus, the above requirement on the parameters can be read as “all dependency pairs are weakly decreasing and at least one dependency pair is strictly decreasing”. This immediately extends to the DP processors based on the removal of rules where it is also sufficient if the set \mathcal{R}' of rewrite rules that are decreasing w.r.t. $\succ_{\mathcal{Pol}}$ is non-empty.

9.3 Summary

In order to demonstrate that the techniques presented in this dissertation are not only of theoretical interest but can be applied automatically in an efficient way, the

Chapter 9. Implementation

methods have been implemented in the automated termination checker **AProVE** [84] for $\mathcal{Th}_{\mathbb{Z}}$ and the general case. Most of the techniques presented in Chapters 6–8 can be implemented straightforwardly, but an implementation of the estimated dependency graph **EDG** is non-trivial since it depends on an extension of \mathcal{E} -unification. To this extent, it was shown how this problem can be approximated effectively.

While the automatic generation of ordinary polynomial interpretations is well understood [48, 73], the automatic generation of $\mathcal{Th}_{\mathbb{Z}}$ -polynomial interpretations needs a non-trivial extension of these methods. The extension presented in this chapter has been fully implemented in **AProVE** and has proved to be very efficient and successful, cf. Chapter 15.

Chapter 10

Conditional Rewriting

The constraints used in CERSs make it possible to restrict the rewrite relation by requiring that the matching substitution makes the constraint of the rule that is to be applied \mathcal{Th} -valid. While this makes it possible to naturally model algorithms in many cases, it is restricted to properties expressible in the (decidable) theory \mathcal{Th} . More general conditions that may also use user-defined functions are possible in the framework of *conditional rewriting*. Here, a rewrite step is only allowed if the conditions of the rule that is to be applied can be established by recursively rewriting them.

Example 10.1. This example shows a quicksort algorithm that takes a set and returns a list. It is a modification of an example from [26] that is widely used in the literature on conditional rewriting. \mathcal{S} and \mathcal{E} are used to model integers and sets as in Chapter 3:

$$\begin{aligned} \mathcal{S} : \quad x + 0 &\rightarrow x \\ &-- x \rightarrow x \\ &- 0 \rightarrow 0 \end{aligned}$$

Chapter 10. Conditional Rewriting

$$\begin{aligned}
-(x + y) &\rightarrow (-x) + (-y) \\
x + (-x) &\rightarrow 0 \\
(x + (-x)) + y &\rightarrow 0 + y \\
x \cup \emptyset &\rightarrow x \\
x \cup x &\rightarrow x \\
(x \cup x) \cup y &\rightarrow x \cup y \\
\mathbf{ins}(x, \mathbf{ins}(x, ys)) &\rightarrow \mathbf{ins}(x, ys) \\
\mathcal{E} : \quad x + y &\approx y + x \\
x + (y + z) &\approx (x + y) + z \\
x \cup y &\approx y \cup x \\
x \cup (y \cup z) &\approx (x \cup y) \cup z \\
\mathbf{ins}(x, \mathbf{ins}(y, zs)) &\approx \mathbf{ins}(y, \mathbf{ins}(x, zs))
\end{aligned}$$

Quicksort is specified by the following conditional constrained rewrite rules:

$$\begin{aligned}
\mathbf{app}(\mathbf{nil}, zs) &\rightarrow zs \\
\mathbf{app}(\mathbf{cons}(x, ys), zs) &\rightarrow \mathbf{cons}(x, \mathbf{app}(ys, zs)) \\
\mathbf{split}(x, \emptyset) &\rightarrow \langle \emptyset, \emptyset \rangle \\
\mathbf{split}(x, zs) \rightarrow^* \langle zl, zh \rangle \mid \mathbf{split}(x, \mathbf{ins}(y, zs)) &\rightarrow \langle \mathbf{ins}(y, zl), zh \rangle \llbracket x > y \rrbracket \\
\mathbf{split}(x, zs) \rightarrow^* \langle zl, zh \rangle \mid \mathbf{split}(x, \mathbf{ins}(y, zs)) &\rightarrow \langle zl, \mathbf{ins}(y, zh) \rangle \llbracket x \not> y \rrbracket \\
\mathbf{qsort}(\emptyset) &\rightarrow \mathbf{nil} \\
\mathbf{split}(x, ys) \rightarrow^* \langle yl, yh \rangle \mid \mathbf{qsort}(\mathbf{ins}(x, ys)) &\rightarrow \mathbf{app}(\mathbf{qsort}(yl), \mathbf{cons}(x, \mathbf{qsort}(yh)))
\end{aligned}$$

Here, $\mathbf{split}(x, ys)$ returns a pair of sets $\langle yl, yh \rangle$ where yl contains all $y \in ys$ such that $x > y$ and yh contains all $y \in ys$ such that $x \not> y$. Intuitively, the condition $\mathbf{split}(x, ys) \rightarrow^* \langle yl, yh \rangle$ of the second \mathbf{qsort} -rule means that $\mathbf{split}(x, ys)$ first needs to be rewritten recursively until it matches $\langle yl, yh \rangle$ (thus giving a binding to these variables) before $\mathbf{qsort}(\mathbf{ins}(x, ys))$ may be reduced using that rule. \triangle

This chapter introduces conditional CERSs, which generalize the CERSs intro-

duced in Chapter 3. Then, termination of conditional CERSs is discussed. For this, well-foundedness of the rewrite relation is not sufficient since it also needs to be ensured that evaluation of the conditions is terminating. These properties are ensured by the notion of *operational termination* [127, 59]. The main result of this chapter shows that operational termination of a conditional CERS can be reduced to (regular) termination of an unconditional CERS by a simple syntactic transformation. This way, the methods developed in this dissertation for showing termination of unconditional CERSs can also be applied for showing operational termination of conditional CERSs. The syntactic transformation is based on a similar transformation used for ordinary conditional rewriting [136, 79].

10.1 Conditional CERSs

Modeling built-in theories and collection data structures is done as in Chapter 3 using a set \mathcal{S} of rewrite rules and a set \mathcal{E} of equations. In contrast to Chapter 3, defined functions are now specified using conditional constrained rewrite rules. As formalized below, both the conditions and the \mathcal{Th} -constraints guard when a rewrite step may be performed. As discussed above, the constraints are still evaluated by a decision procedure for \mathcal{Th} -validity and thus use the built-in theory \mathcal{Th} . The conditions, on the other hand, are evaluated by recursively rewriting them, thus taking their semantics *not* from the theory \mathcal{Th} , but from the CERS itself. This distinction between constraints and conditions is also made at the syntactic level.

Definition 10.2 (Conditional Constrained Rewrite Rules). Conditional constrained rewrite rules *have the form*

$$s_1 \rightarrow^* t_1, \dots, s_n \rightarrow^* t_n \mid l \rightarrow r[[\varphi]]$$

where

Chapter 10. Conditional Rewriting

1. $l, r \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{Th}, \mathcal{V})$ such that $\text{root}(l) \in \mathcal{F} - \mathcal{F}(\mathcal{E} \cup \mathcal{S})$,
2. $s_i, t_i \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{Th}, \mathcal{V})$,
3. φ is a *Th*-constraint with $\mathcal{V}(\varphi) \subseteq \mathcal{V}(l)$,
4. $\mathcal{V}(r) \subseteq \mathcal{V}(l) \cup \bigcup_{j=1}^n \mathcal{V}(t_j)$, and
5. $\mathcal{V}(s_i) \subseteq \mathcal{V}(l) \cup \bigcup_{j=1}^{i-1} \mathcal{V}(t_j)$ for all $1 \leq i \leq n$.¹

As before, the constraint \top will be omitted when stating a rule. Notice that the rules from Example 10.1 satisfy these requirements.

As discussed above, the semantical difference between conditions and constraints in a rule is operational. Conditions need to be evaluated by recursively rewriting them, while the evaluation of constraints is done using a decision procedure for *Th*-validity and does *not* involve any rewriting. This distinction is formalized in Definition 10.4 below.

Conditional constrained equational rewrite systems generalize CERSs by using conditional constrained rewrite rules.

Definition 10.3 (Conditional Constrained Equational Rewrite Systems (CCERS)). *A conditional constrained equational rewrite system (CCERS) has the form $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ for a finite set \mathcal{R} of conditional constrained rewrite rules and, a finite set \mathcal{E} of equations, and a finite set \mathcal{S} of rewrite rules such that*

1. \mathcal{S} is right-linear, i.e., each variable occurs at most once in r for all $l \rightarrow r \in \mathcal{S}$,
2. $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ is \mathcal{E} -convergent, and
3. $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ is strongly \mathcal{E} -coherent.

If \mathcal{R} is unconditional (i.e., $n = 0$ for all conditional constrained rewrite rules $s_1 \rightarrow^* t_1, \dots, s_n \rightarrow^* t_n \mid l \rightarrow r[\varphi]$ from \mathcal{R}), the CERSs from Chapter 3 are obtained.

¹Using the notation of [137], the last two conditions yield deterministic type 3 rules.

Chapter 10. Conditional Rewriting

The rewrite relation induced by a CCERS generalizes Definition 3.15. In addition to the requirements of Definition 3.15, it is now furthermore required that the conditions of the rewrite rule can be established before a reduction may be performed.

Definition 10.4 (Conditional Rewrite Relation). *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CCERS. Then $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ is the least relation satisfying $s \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} t$ iff there exist a conditional constraint rewrite rule $s_1 \rightarrow^* t_1, \dots, s_n \rightarrow^* t_n \mid l \rightarrow r \llbracket \varphi \rrbracket$ in \mathcal{R} , a position $p \in \text{Pos}(s)$, and a \mathcal{Th} -based substitution σ such that*

1. $s|_p \xrightarrow{>\Lambda!}_{\mathcal{E} \setminus \mathcal{S}} l\sigma \circ \gtrsim_{\mathcal{E}}^{\Lambda} l\sigma$,
2. $\varphi\sigma$ is \mathcal{Th} -valid,
3. $s_i\sigma \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^* t_i\sigma \circ \sim_{\mathcal{E}} t_i\sigma$ for all $1 \leq i \leq n$, and
4. $t = s[r\sigma]_p$.

Example 10.5. Continuing Example 10.1, this example illustrates $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ for CCERSs. Consider $t = \text{qsort}(\text{ins}(1, \text{ins}(3, \text{ins}(1, \emptyset))))$ and the \mathcal{Th} -based substitution $\sigma = \{x \mapsto 3, ys \mapsto \text{ins}(1, \emptyset), yl \mapsto \text{ins}(1, \emptyset), yh \mapsto \emptyset\}$. Then it is easy to see that $t \xrightarrow{>\Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \text{qsort}(\text{ins}(1, \text{ins}(3, \emptyset))) \gtrsim_{\mathcal{E}}^{\Lambda} \text{qsort}(\text{ins}(x, ys))\sigma$ and thus $t \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} \text{app}(\text{qsort}(\text{ins}(1, \emptyset)), \text{cons}(3, \text{qsort}(\emptyset)))$ using the third rule for qsort , provided that $\text{split}(3, \text{ins}(1, \emptyset)) \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^* \sim_{\mathcal{E}} \langle \text{ins}(1, \emptyset), \emptyset \rangle$ can be established. In order to verify this, the second split -rule is used. For this, it needs to be checked that the instantiated constraint $3 > 1$ is \mathcal{Th} -valid and that $\text{split}(3, \emptyset) \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^* \sim_{\mathcal{E}} \langle \emptyset, \emptyset \rangle$ can be established. This can be done using the first split -rule. \triangle

Notice that the substitution σ has to instantiate all variables occurring in the conditional rewrite rule and not only the variables occurring in l . From an operational point of view, the substitution σ is constructed as follows. The variables occurring in l are instantiated as in the unconditional case by \mathcal{E} -matching, giving rise to a substitution σ_0 . Since $\mathcal{V}(s_1) \subseteq \mathcal{V}(l)$, the term $s_1\sigma_0$ is fully instantiated and may be reduced until a term s'_1 is reached such that the variables in $\mathcal{V}(t_1) - \mathcal{V}(l)$ may

be instantiated using a substitution σ_1 such that $s'_1 \sim_{\mathcal{E}} t_1 \sigma_0 \sigma_1$. The substitution σ_1 can again be found using \mathcal{E} -matching and determines the instantiations of the fresh variables in $\mathcal{V}(t_1) - \mathcal{V}(l)$. Similar reasoning is then applied to the remaining conditions, resulting in the final substitution $\sigma := \sigma_0 \sigma_1 \cdots \sigma_n$.

An alternative characterization of the rewrite relation of a CCERS can be obtained by an inductive construction, similar to how this is done for ordinary conditional rewriting [137]. For this, a series of unconditional CERSs is defined inductively, and the rewrite relation of the CCERS is then the union of the rewrite relations of these unconditional CERSs:

$$\begin{aligned} \mathcal{R}_0 &= \emptyset \\ \mathcal{R}_{i+1} &= \{l\sigma \rightarrow r\sigma \llbracket \varphi \sigma \rrbracket \mid s_1 \rightarrow^* t_1, \dots, s_n \rightarrow^* t_n \mid l \rightarrow r \llbracket \varphi \rrbracket \in \mathcal{R} \text{ and} \\ &\quad s_j \sigma \xrightarrow{\mathcal{S}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}_i} t_j \sigma \text{ for all } 1 \leq j \leq n\} \end{aligned}$$

Then $\xrightarrow{\mathcal{S}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}} = \bigcup_{i=0}^{\infty} \xrightarrow{\mathcal{S}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}_i}$.

The important property of Lemma 3.26.1 is still true for CCERSs.

Lemma 10.6. *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CCERS. Then $\sim_{\mathcal{E}} \circ \xrightarrow{\mathcal{S}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}} \subseteq \xrightarrow{\mathcal{S}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}} \circ \sim_{\mathcal{E}}$. Furthermore, the $\xrightarrow{\mathcal{S}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}}$ -steps can be performed using the same conditional constrained rewrite rule and Th -based substitution.*

10.2 Termination and Operational Termination

It is well-known that it is not sufficient for a well-behaved notion of termination of conditional rewriting that the rewrite relation is well-founded. In order to get a decidable rewrite relation (under the assumption that \mathcal{E} is size-preserving, see Lemma 3.20), it additionally has to be ensured that evaluation of the conditions is terminating.

Example 10.7. Consider the following ordinary conditional TRS:

$$f(a) \rightarrow^* b \mid a \rightarrow b$$

Then the rewrite relation of this conditional TRS is terminating. In fact, the relation $\rightarrow_{\mathcal{R}}$ is empty since there is no unconditional rewrite rule. But an implementation of rewriting will typically not terminate when trying to reduce the term a since, in order to reduce the term a , it will try to reduce the term $f(a)$, which again requires to reduce the subterm a . \triangle

As argued in [127, 59], the notion of *operational termination* is a natural choice for the combination of these properties that better captures the behavior of actual implementations of rewriting than other commonly used notions like *effective termination* [131]. Furthermore, operational termination turns out to be equivalent to *quasi-decreasingness* [136] (also called *left-right decreasingness* [79]) for ordinary conditional TRSs, see [127]. Since it is unclear whether quasi-decreasingness can be easily extended to rewriting with equations², this chapter considers the notion of operational termination as a natural definition.

For operational termination, the recursive nature of conditional rewriting is reflected by an inference system which aims at proving $s \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R}} t$ or $s \xrightarrow{\mathcal{S}}^*_{Th\|\mathcal{E}\setminus\mathcal{R}} t$. Then operational termination is characterized by the absence of infinite proof trees for this inference system. Notice that the set of inference rules differs from [127] by combining their inference rules **Cong** and **Repl** into one inference rule.

Definition 10.8 (Proof Trees). *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CCERS. The set of (finite) proof trees for $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ and the head of a proof tree are inductively defined as follows:*

²The definition of quasi-decreasingness assumes that the subterm relation is well-founded. But for rewriting with equations, the subterm relation modulo \mathcal{E} is in general not well-founded.

Chapter 10. Conditional Rewriting

1. An open goal G , where G is either $s \rightarrow t$ or $s \rightarrow^* t$ for some terms s, t , is a proof tree. In this case $\text{head}(G) = G$ is the head of the proof tree.
2. A derivation tree

$$T = \frac{T_1 \quad \cdots \quad T_n}{G} \Delta$$

is a proof tree, where G is as in the first case, Δ is one of the derivation rules in Figure 10.1, and T_1, \dots, T_n are proof trees such that

$$\frac{\text{head}(T_1) \quad \cdots \quad \text{head}(T_n)}{G}$$

is an instance of Δ . In this case, $\text{head}(T) = G$.

A proof tree is closed iff it does not contain any open goals.

Example 10.9. Consider the CCERS for quicksort from Examples 10.1 and 10.5 again. Then $\text{qsort}(\text{ins}(1, \text{ins}(3, \text{ins}(1, \emptyset)))) \rightarrow \text{app}(\text{qsort}(\text{ins}(1, \emptyset)), \text{cons}(3, \text{qsort}(\emptyset)))$ is an open goal and

$$\frac{\frac{\frac{\text{split}(3, \emptyset) \rightarrow \langle \emptyset, \emptyset \rangle \quad \text{Repl} \quad \frac{\langle \emptyset, \emptyset \rangle \rightarrow^* \langle \emptyset, \emptyset \rangle \quad \text{Refl}}{\langle \emptyset, \emptyset \rangle \rightarrow^* \langle \emptyset, \emptyset \rangle} \quad \text{Tran}}{\text{split}(3, \emptyset) \rightarrow^* \langle \emptyset, \emptyset \rangle} \quad \text{Repl} \quad \frac{\text{split}(3, \text{ins}(1, \emptyset)) \rightarrow \langle \text{ins}(1, \emptyset), \emptyset \rangle \quad \text{Refl} \quad \frac{\langle \text{ins}(1, \emptyset), \emptyset \rangle \rightarrow^* \langle \text{ins}(1, \emptyset), \emptyset \rangle \quad \text{Refl}}{\langle \text{ins}(1, \emptyset), \emptyset \rangle \rightarrow^* \langle \text{ins}(1, \emptyset), \emptyset \rangle} \quad \text{Tran}}{\text{split}(3, \text{ins}(1, \emptyset)) \rightarrow^* \langle \text{ins}(1, \emptyset), \emptyset \rangle} \quad \text{Repl}}{\text{qsort}(\text{ins}(1, \text{ins}(3, \text{ins}(1, \emptyset)))) \rightarrow \text{app}(\text{qsort}(\text{ins}(1, \emptyset)), \text{cons}(3, \text{qsort}(\emptyset)))} \quad \text{Repl}$$

is a closed proof tree with this goal as its head. △

Now an infinite proof tree is defined to be a sequence of proof trees such that each member of this sequence can be obtained from its immediate predecessor by expanding one or more open goals.

| |
|--|
| <p>Refl $\frac{}{s \rightarrow^* t}$</p> <p>if $s \sim_{\mathcal{E}} t$</p> |
| <p>Tran $\frac{s \rightarrow t \quad t \rightarrow^* u}{s \rightarrow^* u}$</p> |
| <p>Repl $\frac{s_1\sigma \rightarrow^* t_1\sigma \quad \cdots \quad s_n\sigma \rightarrow^* t_n\sigma}{s \rightarrow t}$</p> <p>if $s_1 \rightarrow^* t_1, \dots, s_n \rightarrow^* t_n \mid l \rightarrow r[[\varphi]] \in \mathcal{R},$ $p \in \mathcal{Pos}(s),$ σ is \mathcal{Th}-based, $s _p \xrightarrow{\geq \Lambda} \mathcal{E} \setminus \mathcal{S} \circ \gtrsim_{\mathcal{E}} l\sigma,$ $\varphi\sigma$ is \mathcal{Th}-valid, and $t = s[r\sigma]_p$</p> |

Figure 10.1: Derivation rules for the generation of proof trees.

Definition 10.10 (Prefixes of Proof Trees, Infinite Proof Trees). *A proof tree T is a prefix of a proof tree T' , written $T \subset T'$, if there exist one or more open goals G_1, \dots, G_n in T such that T' is obtained from T by replacing each G_i by a derivation tree T_i with $\text{head}(T_i) = G_i$. An infinite proof tree is an infinite sequence $\{T_i\}_{i \geq 0}$ of finite proof trees such that $T_i \subset T_{i+1}$ for all $i \geq 0$.*

Proof trees do not impose any restriction on the order in which leaves are expanded. While Definition 10.4 does not impose an order in which the conditions are to be evaluated, the discussion following Definition 10.4 shows that an implementation of conditional rewriting needs to evaluate the conditions from left to right. This

behavior can be reflected in proof trees as well.

Definition 10.11 (Well-Formed Proof Trees). *A proof tree T is well-formed if it is either an open goal, a closed proof tree, or a derivation tree of the form*

$$\frac{T_1 \quad \cdots \quad T_n \Delta}{G}$$

where T_j is a well-formed proof tree for all $1 \leq j \leq n$ and there is an $i \leq n$ such that T_i is not closed, T_j is closed for all $j < i$, and T_k is an open goal for all $k > i$. An infinite proof tree is well-formed iff it consists of well-formed proof trees.

As mentioned above, *operational termination* is characterized by the absence of infinite well-formed proof trees.

Definition 10.12 (Operational Termination). *A CCERS $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is operationally terminating iff it does not admit infinite well-formed proof trees.*

It is easy to show that the notions of termination and operational termination coincide for CERSs.

Lemma 10.13. *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CERS. Then $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is operationally terminating iff $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is terminating.*

10.3 Elimination of Conditions

In order to show operational termination of a CCERS $(\mathcal{R}, \mathcal{S}, \mathcal{E})$, it is transformed into a CERS $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ such that operational termination of $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ implies operational termination of $(\mathcal{R}, \mathcal{S}, \mathcal{E})$. By Lemma 10.13, $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ is operationally terminating if it is terminating, and thus a termination proof of $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ constitutes a proof of operational termination of $(\mathcal{R}, \mathcal{S}, \mathcal{E})$. The transformation generalizes

the well-known one for ordinary conditional rewriting [136, 79] to rewriting with equations, normalization, and constraints. An extension of the classical transformation to context-sensitive rewriting with equations was proposed in [59]. The presentation in this section is influenced by [59].

Definition 10.14 (Transformation \mathcal{U}). *Let $\rho : s_1 \rightarrow^* t_2, \dots, s_n \rightarrow^* t_n \mid l \rightarrow r \llbracket \varphi \rrbracket$ be a conditional constrained rewrite rule. Then $\mathcal{U}(\rho)$ is defined by:*

1. *if $n = 0$ then $\mathcal{U}(\rho) = \{ \rho \}$*
2. *if $n > 0$ then $\mathcal{U}(\rho) = \{ l \rightarrow U_1^\rho(s_1, x_1^*) \llbracket \varphi \rrbracket \} \cup$ (1)
 $\{ U_{i-1}^\rho(t_{i-1}, x_{i-1}^*) \rightarrow U_i^\rho(s_i, x_i^*) \llbracket \varphi \rrbracket \mid 2 \leq i \leq n \} \cup$ (2)
 $\{ U_n^\rho(t_n, x_n^*) \rightarrow r \llbracket \varphi \rrbracket \}$ (3)*

Here, the U_i^ρ are fresh function symbols and, for $1 \leq i \leq n$, the expression x_i^* denotes the sorted list of variables in the set $\mathcal{V}(l) \cup \mathcal{V}(t_1) \cup \dots \cup \mathcal{V}(t_{i-1})$ according to some fixed order on the set \mathcal{V} of all variables. For a finite set \mathcal{R} of conditional constrained rewrite rules, let $\mathcal{U}(\mathcal{R}) = \bigcup_{\rho \in \mathcal{R}} \mathcal{U}(\rho)$.

Example 10.15. Continuing the running example, the transformation produces the following unconditional constrained rewrite rules:

$$\begin{aligned}
 \text{app}(\text{nil}, zs) &\rightarrow zs \\
 \text{app}(\text{cons}(x, ys), zs) &\rightarrow \text{cons}(x, \text{app}(ys, zs)) \\
 \text{split}(x, \emptyset) &\rightarrow \langle \emptyset, \emptyset \rangle \\
 \text{split}(x, \text{ins}(y, zs)) &\rightarrow U_1(\text{split}(x, zs), x, y, zs) \llbracket x > y \rrbracket \\
 U_1(\langle zl, zh \rangle, x, y, zs) &\rightarrow \langle \text{ins}(y, zl), zh \rangle \llbracket x > y \rrbracket \\
 \text{split}(x, \text{ins}(y, zs)) &\rightarrow U_2(\text{split}(x, zs), x, y, zs) \llbracket x \not> y \rrbracket \\
 U_2(\langle zl, zh \rangle, x, y, zs) &\rightarrow \langle zl, \text{ins}(y, zh) \rangle \llbracket x \not> y \rrbracket \\
 \text{qsort}(\emptyset) &\rightarrow \text{nil} \\
 \text{qsort}(\text{ins}(x, ys)) &\rightarrow U_3(\text{split}(x, ys), x, ys) \\
 U_3(\langle yl, yh \rangle, x, ys) &\rightarrow \text{app}(\text{qsort}(yl), \text{cons}(x, \text{qsort}(yh)))
 \end{aligned}$$

Chapter 10. Conditional Rewriting

In order to ease readability, the function symbols U_i^ρ from Definition 10.14 are denoted using simplified names. Termination of this unconditional system can easily be shown using the methods developed in this dissertation. \triangle

The main result of this chapter relates operational termination of $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ to operational termination of $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$.

Lemma 10.16. *For any well-formed proof tree T for $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ whose head goal is either $s \rightarrow t$ or $s \rightarrow^* t$, there exists a well-formed proof tree $\beta(T)$ for $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ whose head goal is $s \rightarrow^* t$. Furthermore, if $T \subset T'$ for some T' , then $\beta(T) \subset \beta(T')$.*

Using this lemma, it is now easy to show that operational termination of the transformed system implies operational termination of the original system.

Theorem 10.17. *If $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ is operationally terminating, then $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is operationally terminating.*

By combining Theorem 10.17 and Lemma 10.13, the desired result that termination of $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ implies operational termination of $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is obtained.

Corollary 10.18. *If $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ is terminating, then $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is operationally terminating.*

Given that the transformation is sound for proving termination, it is of course a valid question whether it is also complete, i.e., if operational termination of $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ implies that $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ is terminating. The following example from [131] show that this is in general not the case.

Example 10.19. Consider the following ordinary conditional TRS \mathcal{R} [131]:

$$\begin{array}{l}
 a \rightarrow c \\
 a \rightarrow d \\
 b \rightarrow c \\
 b \rightarrow d \\
 c \rightarrow e \\
 c \rightarrow l \\
 k \rightarrow l \\
 k \rightarrow m \\
 d \rightarrow m \\
 A \rightarrow h(f(a), f(b)) \\
 h(x, x) \rightarrow g(x, x, f(k)) \\
 g(d, x, x) \rightarrow A \\
 x \rightarrow^* e \mid f(x) \rightarrow x
 \end{array}$$

Then \mathcal{R} is operationally terminating [137], but $\mathcal{U}(\mathcal{R})$ is non-terminating since the term $h(f(a), f(b))$ starts an infinite reduction w.r.t. $\rightarrow_{\mathcal{U}(\mathcal{R})}$. \triangle

For an ordinary conditional TRS \mathcal{R} , quasi-decreasingness (and thus operational termination) of \mathcal{R} implies that $\mathcal{U}(\mathcal{R})$ is *innermost* terminating [136]. Thus, while the converse of Corollary 10.18 is not true in general, it could be investigated in future work whether $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ is innermost terminating whenever $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is operationally terminating.

10.4 Summary

The CERSs as introduced in Chapter 3 make it possible to naturally model algorithms in many cases. The framework of CERSs can, however, be made even more expressive and natural by allowing more general conditions for the rewrite rules. The

Chapter 10. Conditional Rewriting

constraints of rewrite rules are restricted to properties expressible in the (decidable) theory \mathcal{Th} , while conditional rewriting makes it possible to use conditions that may also utilize user-defined functions.

Since the conditions of a rewrite rule are evaluated by recursively rewriting them, termination analysis of conditional systems is more complex than for unconditional systems. This chapter has introduced the notion of *operational termination* [127, 59] of such systems. Then, the main result of this chapter has shown that operational termination of a CCERS can be reduced to (regular) termination of a CERS by a simple syntactic transformation. Therefore, the methods developed in this dissertation for showing termination of CERSs can also be applied for showing operational termination of CCERSs. The implementation in AProVE has been successfully applied to several CERSs that have been obtained from CCERSs by the transformation introduced in this chapter, cf. Chapter 15.

Chapter 11

Context-Sensitive Rewriting and Dependency Pairs

Context-sensitive rewriting [121, 123] is an operational restriction of term rewriting that can be used to model lazy (non-strict) evaluation as used in functional programming languages such as **Haskell** (for more on the relationship between lazy evaluation and context-sensitive rewriting, see [124]). Additionally, declarative specification and programming languages such as **Maude** [43] directly support context-sensitive rewriting strategies. This chapter introduces context-sensitive rewriting strategies for CERSs as defined in Chapter 3.

In context-sensitive rewriting, the arguments where an evaluation may take place are specified for each function symbol. Then a reduction is only allowed if it takes place at a position that is not forbidden by a function symbol occurring somewhere above it.

Example 11.1. Consider the CERS where \mathcal{S} and \mathcal{E} are used to model integers and sets using \emptyset and ins as in Chapter 3 and where \mathcal{R} consists of the following rules:

$$\begin{aligned}
 \text{from}(x) &\rightarrow \text{ins}(x, \text{from}(x + 1)) \\
 \text{take}(0, xs) &\rightarrow \text{nil} \\
 \text{take}(x, \text{ins}(y, ys)) &\rightarrow \text{cons}(y, \text{take}(x - 1, ys)) \llbracket x > 0 \rrbracket \\
 \text{pick}(\text{ins}(x, xs)) &\rightarrow x \\
 \text{drop}(\text{ins}(x, xs)) &\rightarrow xs
 \end{aligned}$$

Here, the function symbol `from` is used to generate the (infinite) subsets of integers that are greater than or equal to the argument of `from`. Thus, the term `take(2, from(0))` admits an infinite reduction in which the `from`-rule is applied over and over again. However, there also is a finite reduction of that term which results in the normal form `cons(0, cons(1, nil))`. This reduction can be enforced using context-sensitive rewriting if evaluation of the second argument of `ins` is forbidden since the recursive call to `from` is then blocked. \triangle

Since context-sensitive rewriting may result in a terminating rewrite relation for CERSs where regular rewriting is not terminating, proving the termination of context-sensitive rewriting is challenging.

For ordinary TRSs, there are two approaches to proving termination of context-sensitive rewriting. The first approach is to apply a syntactic transformation in such a way that termination of context-sensitive rewriting with a TRS is implied by (regular) termination of the TRS obtained by the transformation. For details on this approach, see [83, 126]. While the application of these transformations makes it possible to use any method for proving termination of the transformed TRS, it often generates TRSs whose termination cannot be established using existing methods.

The second approach consists of the development of dedicated methods for proving termination of context-sensitive rewriting. Examples for adaptations of classical methods are context-sensitive recursive path orderings [30] and context-sensitive polynomial interpretations [125]. The main drawback of these adaptations of classical

methods is the limited power which is inherited from the classical methods. Adapting the more powerful dependency pair method [12] to context-sensitive TRSs has been a challenge. A first adaptation of the dependency pair method to context-sensitive TRSs has been presented in [2]. But this adaptation has severe disadvantages compared to the ordinary dependency pair method since it requires the introduction of collapsing dependency pairs (i.e., dependency pairs of the form $f^\#(t_1, \dots, t_n) \rightarrow x$ where x is a variable). These collapsing dependency pairs make it necessary to impose strong restrictions on how the method can be applied.

An alternative adaptation of the dependency pair method to context-sensitive TRSs has recently been presented in [1]. This adaptation does not require collapsing dependency pairs and makes it much easier to adapt termination techniques developed within the dependency pair method to the context-sensitive case. Empirical evaluations show the superiority of this most recent formulation, see [1].

After introducing the basic terminology of context-sensitive rewriting with CERSs in Section 11.1 and briefly discussing context-sensitive rewriting with CCERSs in Section 11.2, the main technical result of this chapter is proven in Section 11.3. By a non-trivial extension of [1], termination of context-sensitive rewriting with a CERS is reduced to showing absence of infinite chains of dependency pairs. This makes it possible to develop a dependency pair framework for the termination analysis of context-sensitive rewriting with CERSs, similar to the case of non-context-sensitive rewriting with CERSs as discussed in Chapter 5.

11.1 Context-Sensitive Rewriting

Context-sensitive rewriting strategies are obtained using *replacement maps* that are used to define the context under which a reduction may take place. This is done by specifying the argument positions of a function symbol f where a reduction is

allowed. Intuitively, if the replacement map restricts reductions in a certain argument position, then the whole subterm below that argument position may not be reduced.

Definition 11.2 (Replacement Maps). *A replacement map is a mapping μ with $\mu(f) \subseteq \{1, \dots, \text{arity}(f)\}$ for every function symbol $f \in \mathcal{F} \cup \mathcal{F}_{Th}$.*

Replacement maps are used to denote a subset of all positions in a term as *active*. A position is active if it can be reached from the root of the term by only descending into argument positions that are not restricted by the replacement map μ .

Definition 11.3 (Active and Inactive Positions). *Let μ be a replacement map and let t be a term. Then the set of active positions of t , written $\mathcal{Pos}^\mu(t)$, is defined by*

1. $\mathcal{Pos}^\mu(x) = \{\Lambda\}$ for $x \in \mathcal{V}$, and
2. $\mathcal{Pos}^\mu(f(t_1, \dots, t_n)) = \{\Lambda\} \cup \{i.p \mid i \in \mu(f) \text{ and } p \in \mathcal{Pos}^\mu(t_i)\}$.

The set of inactive positions of t is defined as $\mathcal{Pos}^{-\mu}(t) = \mathcal{Pos}(t) - \mathcal{Pos}^\mu(t)$.

Example 11.4. Consider a replacement map with $\mu(f) = \{1\}$, $\mu(g) = \{1, 2\}$, and $\mu(h) = \emptyset$. If $t = f(g(x, h(y)), h(x))$, then $\mathcal{Pos}^\mu(t) = \{\Lambda, 1, 1.1, 1.2\}$ and $\mathcal{Pos}^{-\mu}(t) = \{1.2.1, 2, 2.1\}$. △

The concept of active positions can also be used to define active (and inactive) subterms of a given term. This is similar to Definition 2.5.4.

Definition 11.5 (Active and Inactive Subterms). *Let μ be a replacement map and let t be a term. If $t|_p = s$ for an active position $p \in \mathcal{Pos}^\mu(t)$, then s is an active subterm of t , written $t \succeq_\mu s$. If additionally $p \neq \Lambda$, then s is an active strict subterm of t , written $t \triangleright_\mu s$. If $t|_p = s$ for an inactive position $p \in \mathcal{Pos}^{-\mu}(t)$, then s is an inactive strict subterm of t , written $t \triangleright_{-\mu} s$.¹*

¹Notice that there are no inactive (non-strict) subterms since Λ is always active.

Chapter 11. Context-Sensitive Rewriting and Dependency Pairs

The classification of active and inactive subterms can easily be extended to other notions as well to obtain the sets $\mathcal{V}^\mu(t)$ of variables occurring in active positions in t , $\mathcal{V}^{\neg\mu}(t)$ of variables occurring in inactive positions in t , $\mathcal{F}^\mu(t)$ of function symbols occurring in active positions in t , $\mathcal{F}^{\neg\mu}(t)$ of function symbols occurring in inactive positions in t , etc.

Example 11.6. Continuing Example 11.4,

1. $f(g(x, h(y)), h(x)) \triangleright_\mu f(g(x, h(y)), h(x))$
 $f(g(x, h(y)), h(x)) \triangleright_\mu g(x, h(y))$
 $f(g(x, h(y)), h(x)) \triangleright_\mu x$
 $f(g(x, h(y)), h(x)) \triangleright_\mu h(y)$
2. $f(g(x, h(y)), h(x)) \triangleright_{\neg\mu} y$
 $f(g(x, h(y)), h(x)) \triangleright_{\neg\mu} h(x)$
 $f(g(x, h(y)), h(x)) \triangleright_{\neg\mu} x$
3. $\mathcal{V}^\mu(f(g(x, h(y)), h(x))) = \{x\}$
 $\mathcal{V}^{\neg\mu}(f(g(x, h(y)), h(x))) = \{x, y\}$
4. $\mathcal{F}^\mu(f(g(x, h(y)), h(x))) = \{f, g, h\}$
 $\mathcal{F}^{\neg\mu}(f(g(x, h(y)), h(x))) = \{h\}$

Notice that $\mathcal{V}^\mu(t)$ and $\mathcal{V}^{\neg\mu}(t)$ (and $\mathcal{F}^\mu(t)$ and $\mathcal{F}^{\neg\mu}(t)$) are not necessarily disjoint. \triangle

Now a *context-sensitive constrained equational rewrite system* combines a CERS as in Definition 3.12 with a replacement map. Notice that the replacement map μ needs to satisfy several conditions on the occurrences of variables in \mathcal{S} and \mathcal{E} . As already noticed for the associative-commutative case in [71], this is due to the permutative nature of equations in \mathcal{E} that may otherwise bring subterms from inactive positions into active positions, and vice versa. Figure 11.1 lists the replacement maps that are allowed by this definition for the data structures from Chapter 3.

| | Constructors | Conditions on μ |
|-----------------|---|---|
| Natural numbers | $0, 1, +$ | $\mu(+) = \{1, 2\}$ |
| Integers | $0, 1, +, -$ | $\mu(+) = \{1, 2\}$ $\mu(-) = \{1\}$ |
| Lists | nil, cons | — |
| Lists | $\text{nil}, \langle \cdot \rangle, ++$ | $\mu(++) = \{1, 2\}$ |
| Compact Lists | nil, cons | $\mu(\text{cons}) = \emptyset$ or $\mu(\text{cons}) = \{1, 2\}$ |
| Compact Lists | $\text{nil}, \langle \cdot \rangle, ++$ | $\mu(++) = \{1, 2\}$ |
| Multisets | \emptyset, ins | $\mu(\text{ins}) = \emptyset$ or $\mu(\text{ins}) = \{1, 2\}$ |
| Multisets | $\emptyset, \{ \cdot \}, \cup$ | $\mu(\cup) = \{1, 2\}$ |
| Sets | \emptyset, ins | $\mu(\text{ins}) = \emptyset$ or $\mu(\text{ins}) = \{1, 2\}$ |
| Sets | $\emptyset, \{ \cdot \}, \cup$ | $\mu(\cup) = \{1, 2\}$ |

Figure 11.1: Replacement maps allowed for context-sensitive rewriting.

Definition 11.7 (Context-Sensitive CERSs). *A context-sensitive CERS (CS-CERS) has the form $(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ for a CERS $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ and a replacement map μ such that the following conditions are satisfied:*

1. \mathcal{E} is collapse-free.
2. For all $u \approx v \in \mathcal{E}$,
 - (a) $\mathcal{V}^\mu(u) = \mathcal{V}^\mu(v)$ and $\mathcal{V}^{-\mu}(u) = \mathcal{V}^{-\mu}(v)$,
 - (b) for all inactive non-variable subterms u' of u , $u' \triangleright_\mu x$ for a variable x implies $v' \triangleright_\mu x$ for an inactive non-variable subterm v' of v , and
 - (c) for all inactive non-variable subterms v' of v , $v' \triangleright_\mu x$ for a variable x implies $u' \triangleright_\mu x$ for an inactive non-variable subterm u' of u .
3. For all $l \rightarrow r \in \mathcal{S}$,
 - (a) $\mathcal{V}^\mu(r) \cap \mathcal{V}^{-\mu}(l) = \emptyset$, and
 - (b) for all inactive non-variable subterms r' of r , $r' \triangleright_\mu x$ for a variable x implies $l' \triangleright_\mu x$ for an inactive non-variable subterm l' of l .

Chapter 11. Context-Sensitive Rewriting and Dependency Pairs

The rewrite relation of a CS-CERS is obtained by a small modification of Definition 3.15. The only difference is that for CS-CERSs, the position where the reduction takes place has to be active. Notice that it is also easily possible to consider an innermost rewrite relation or even a general restricted rewrite relation for CS-CERSs, just as this was done in Section 3.4 for standard CERSs. For simplicity of presentation, this is not considered in this dissertation.

Definition 11.8 (Rewrite Relation of a CS-CERS). *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ be a CS-CERS. Then $s \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} t$ iff there exist a constrained rewrite rule $l \rightarrow r[\![\varphi]\!] \in \mathcal{R}$, an active position $p \in \mathcal{P}os^\mu(s)$, and a Th -based substitution σ such that*

1. $s|_p \xrightarrow{\geq \Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \circ \overset{\geq \Lambda}{\sim}_{\mathcal{E}} l\sigma$,
2. $\varphi\sigma$ is Th -valid, and
3. $t = s[r\sigma]_p$.

Example 11.9. The CERS from Example 11.1 becomes a CS-CERS by considering the replacement map μ with $\mu(\text{ins}) = \emptyset$ and $\mu(f) = \{1, \dots, \text{arity}(f)\}$ for all $f \neq \text{ins}$. Then the reduction of the term $\text{take}(2, \text{from}(0))$ has the following form:

$$\begin{aligned}
 \text{take}(2, \text{from}(0)) &\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \text{take}(2, \text{ins}(0, \text{from}(1))) \\
 &\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \text{cons}(0, \text{take}(2 - 1, \text{from}(1))) \\
 &\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \text{cons}(0, \text{cons}(1, \text{take}(1 - 1, \text{from}(2)))) \\
 &\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \text{cons}(0, \text{cons}(1, \text{nil}))
 \end{aligned}$$

Notice that an infinite reduction of this term is not possible since the recursive call to from in the rule $\text{from}(x) \rightarrow \text{ins}(x, \text{from}(x + 1))$ occurs in an inactive position. \triangle

The following properties of rewriting with CERSs are needed in the remainder of this dissertation. They are analogous to Lemmas 2.21 and 3.26. Here, a context C is an *active context* iff \square occurs in an active position in it.

Lemma 11.10. *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ be a CS-CERS and let s, t be terms.*

Chapter 11. Context-Sensitive Rewriting and Dependency Pairs

1. Let $s = C[f(s^*)]$ for an active context C where $f \notin \mathcal{F}(\mathcal{E})$. If $s \sim_{\mathcal{E}} t$, then $t = C'[f(t^*)]$ for an active context C' such that $C \sim_{\mathcal{E}} C'$ and $f(s^*) \stackrel{\geq \Lambda}{\sim}_{\mathcal{E}} f(t^*)$.
2. $\sim_{\mathcal{E}} \circ \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \subseteq \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \sim_{\mathcal{E}}$, where the $\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu}$ steps can be performed using the same constrained rewrite rule and Th -based substitution.
3. $\rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \subseteq \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu}^+ \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}}^-$

Just as for Corollary 3.27, the following can easily be obtained.

Corollary 11.11. *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ be a CS-CERS and let s, t be terms.*

1. If $s \sim_{\mathcal{E}} t$, then s starts an infinite $\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction iff t starts an infinite $\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction.
2. If $s \rightarrow_{\mathcal{E}\setminus\mathcal{S}} t$ and t starts an infinite $\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction, then s starts an infinite $\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction.

11.2 Context-Sensitive Conditional Rewriting

Recall the conditional CERSs from Chapter 10: In addition to Th -constraints, the rewrite rules of CCERSs also contain conditions that need to be established before applying a rule. In contrast to the Th -constraints, whose validity is established by a decision procedure for Th , these conditions need to be established by recursively rewriting them since they may contain user-defined functions.

A combination of conditional rewrite rules with context-sensitive reduction strategies is easily possible in the obvious way, giving rise to *context-sensitive conditional constrained equational rewrite systems (CS-CCERSs)*. As for CCERSs, operational termination is a crucial property of CS-CCERSs, and the question of how to verify operational termination of a CS-CCERS naturally arises.

Recall from Chapter 10 that operational termination of CCERSs can be reduced to termination of (unconditional) CERSs by a simple syntactic transformation. This transformation can be adapted to the context-sensitive case by a simple extension of the replacement map μ : for the fresh function symbols U_i^p introduced in Definition 10.14, let $\mu(U_i^p) = \{1\}$, i.e., only the subterms corresponding to the conditions of the conditional rewrite rules may be reduced. This is the same approach that is taken in [59] as well. Then, the statement of Corollary 10.18 is still true, i.e., the CS-CCERS $(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ is operationally terminating if the (unconditional) CS-CERS $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E}, \mu)$ is terminating.

11.3 Context-Sensitive Dependency Pairs

Extending the dependency pair method from ordinary rewriting to ordinary context-sensitive rewriting has been a challenge. Recall from Section 5.1 that dependency pairs are built from recursive calls to defined function symbols occurring on the right-hand side of rewrite rules since only these recursive calls may cause non-termination. For context-sensitive rewriting, one might be tempted to restrict the generation of dependency pairs to recursive calls occurring in active positions since these are the only positions where reductions may take place. The following example from [2] shows that this results in an unsound method.

Example 11.12. Consider the following ordinary TRS [2]:

$$\begin{aligned} a &\rightarrow c(f(a)) \\ f(c(x)) &\rightarrow x \end{aligned}$$

Let $\mu(c) = \emptyset$ and $\mu(f) = \{1\}$. Since the recursive calls in the first rule are in inactive positions, no dependency pair would be generated if only recursive calls occurring in active positions are considered. Then, context-sensitive termination of the TRS

could be concluded, even though it is not terminating: $f(\mathbf{a}) \rightarrow_{\mathcal{R},\mu} f(\mathbf{c}(f(\mathbf{a}))) \rightarrow_{\mathcal{R},\mu} f(\mathbf{a}) \rightarrow_{\mathcal{R},\mu} \dots$. Here, $\rightarrow_{\mathcal{R},\mu}$ is the standard context-sensitive rewrite relation which is obtained from Definition 11.8 by disregarding \mathcal{S} , \mathcal{E} , and every mention of \mathcal{Th} . \triangle

The problem of the naive approach outlined above is that recursive calls in inactive positions of right-hand sides may become active again after applying other rules. In Example 11.12, the recursive call to $f(\mathbf{a})$ that occurs in an inactive position is migrated to an active position by an application of the second rule. This is the reason that the method of [2] has to create collapsing dependency pairs whose right-hand side is a *migrating variable*, where, for a rule $l \rightarrow r$, a variable x is migrating if $r \triangleright_{\mu} x$ but $l \not\triangleright_{\mu} x$. In Example 11.12, the collapsing dependency pair $f^{\#}(\mathbf{c}(x)) \rightarrow x$ is created.

As noticed in [1], the need for collapsing dependency pairs causes severe disadvantages since it becomes quite hard to extend methods for proving termination from ordinary rewriting to ordinary context-sensitive rewriting because the collapsing dependency pairs require a special treatment. While recent work provides some progress [2, 3, 92], the resulting methods are quite weak in practice. As a simple example of this weakness, collapsing dependency pairs have an outgoing arc to every other dependency pair in the estimated dependency graph, thus making this technique not very effective for decomposing the termination proof.

An alternative to the collapsing dependency pairs needed in [2] has recently been presented in [1]. The main observation in [1] is that only certain instantiations of the migrating variables need to be considered. A first, naive approach for this would be to only consider instantiations by *hidden terms*, which are terms with a defined root symbol occurring inactively in right-hand sides.

Definition 11.13 (Hidden Terms). *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ be a CS-CERS. A term t is hidden iff $\text{root}(t) \in \mathcal{D}(\mathcal{R})$ and there exists a rule $l \rightarrow r \llbracket \varphi \rrbracket$ from \mathcal{R} such that $r \triangleright_{-\mu} t$.*

Example 11.14. For the CS-CERS from Example 11.1, the term $\text{from}(x + 1)$ is hidden since $\text{ins}(x, \text{from}(x + 1)) \triangleright_{-\mu} \text{from}(x + 1)$. \triangle

Instantiating the migrating variable in Example 11.12 by the hidden term $f(\mathbf{a})$ is sufficient, since the dependency pair $f^\sharp(c(f(\mathbf{a}))) \rightarrow f^\sharp(\mathbf{a})$ obtained by this instantiation gives rise to an infinite chain. In general, considering only instantiations by hidden terms results in an unsound method, as shown by the following example.

Example 11.15. Consider the following ordinary TRS [1]:

$$\begin{aligned} \mathbf{a} &\rightarrow f(\mathbf{g}(\mathbf{b})) \\ f(x) &\rightarrow h(x) \\ h(x) &\rightarrow x \\ \mathbf{b} &\rightarrow \mathbf{a} \end{aligned}$$

Let $\mu(\mathbf{g}) = \{1\}$ and $\mu(\mathbf{a}) = \mu(\mathbf{b}) = \mu(f) = \mu(h) = \emptyset$. The only hidden term is \mathbf{b} obtained from the first rule. If migrating variables are only instantiated by hidden terms, then the following dependency pairs are obtained:

$$\begin{aligned} \mathbf{a}^\sharp &\rightarrow f^\sharp(\mathbf{g}(\mathbf{b})) \\ f^\sharp(x) &\rightarrow h^\sharp(x) \\ h^\sharp(\mathbf{b}) &\rightarrow \mathbf{b}^\sharp \\ \mathbf{b}^\sharp &\rightarrow \mathbf{a}^\sharp \end{aligned}$$

Since these dependency pairs do not give rise to an infinite chain, termination could be concluded, even though $\mathbf{a} \rightarrow_{\mathcal{R}, \mu} f(\mathbf{g}(\mathbf{b})) \rightarrow_{\mathcal{R}, \mu} h(\mathbf{g}(\mathbf{b})) \rightarrow_{\mathcal{R}, \mu} \mathbf{g}(\mathbf{b}) \rightarrow_{\mathcal{R}, \mu} \mathbf{g}(\mathbf{a}) \rightarrow_{\mathcal{R}, \mu} \dots$ is an infinite reduction. \triangle

As motivated by this example, it becomes necessary to consider certain contexts that may be built above a hidden term using the rewrite rules. In Example 11.15, this context is $\mathbf{g}(\square)$. Formally, this observation is captured using the notion of *hiding*

Chapter 11. Context-Sensitive Rewriting and Dependency Pairs

contexts. The definition in this dissertation extends the one given in [1] by also considering \mathcal{S} and \mathcal{E} .

Definition 11.16 (Hiding Contexts). *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ be a CS-CERS. Then $f \in \mathcal{F} \cup \mathcal{F}_{Th}$ hides position i iff $i \in \mu(f)$ and either*

1. $f \in \mathcal{F}(\mathcal{E}) \cup \mathcal{F}(\mathcal{S})$, or
2. *there exist a rule $l \rightarrow r[\![\varphi]\!] from \mathcal{R} and a term $s = f(s_1, \dots, s_i, \dots, s_n)$ such that $r \triangleright_{-\mu} s$ and $s_i \succeq_{\mu} x$ for a variable x or $s_i \succeq_{\mu} g(\dots)$ for an $g \in \mathcal{D}(\mathcal{R})$.$*

A context C with one hole is hiding iff either

1. $C = \square$, or
2. $C = f(t_1, \dots, t_{i-1}, C', t_{i+1}, \dots, t_n)$ such that f hides position i and C' is a hiding context.

Notice that every hiding context is an active context.

Example 11.17. For the CS-CERS from Example 11.1, $+$ hides positions 1 and 2, $-$ hides position 1, and from hides position 1 due to the first rewrite rule. \triangle

Notice that there are, in general, infinitely many hiding contexts. For example, the hiding contexts in Example 11.15 are $\square, \mathbf{g}(\square), \mathbf{g}(\mathbf{g}(\square)), \dots$. In order to represent these infinitely many hiding contexts using only finitely many dependency pairs, fresh function symbols \mathbf{U}_{base} and \mathbf{U}_{univ} are introduced that will be used to deconstruct a hiding context in order to obtain the hidden term contained in it. This is achieved by introducing dependency pairs of the form $\mathbf{U}_s(g(x_1, \dots, x_i, \dots, x_n)) \rightarrow \mathbf{U}_{s'}(x_i)$ whenever the function symbol g hides position i . Here, $s, s' \in \{\text{base}, \text{univ}\}$ are the appropriate sorts. Thus, the following definition of context-sensitive dependency pairs is obtained. Here, $\text{DP}_{\mathbf{u}}$ is used instead of the collapsing dependency pairs needed in [2]. This is similar to [1].

Definition 11.18 (Context-Sensitive Dependency Pairs). *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ be a CS-CERS. The context-sensitive dependency pairs of \mathcal{R} are defined as $\text{DP}(\mathcal{R}, \mu) = \text{DP}_o(\mathcal{R}, \mu) \cup \text{DP}_u(\mathcal{R}, \mu)$ where*

$$\text{DP}_o(\mathcal{R}, \mu) = \{l^\sharp \rightarrow t^\sharp \llbracket \varphi \rrbracket \mid l \rightarrow r \llbracket \varphi \rrbracket \in \mathcal{R}, r \succeq_\mu t, \text{root}(t) \in \mathcal{D}(\mathcal{R})\}$$

$$\text{DP}_u(\mathcal{R}, \mu) = \{l^\sharp \rightarrow \mathbf{U}_s(x) \llbracket \varphi \rrbracket \mid l \rightarrow r \llbracket \varphi \rrbracket \in \mathcal{R}, r \succeq_\mu x, l \not\prec_\mu x\} \quad (1)$$

$$\cup \{\mathbf{U}_s(g(x_1, \dots, x_i, \dots, x_n)) \rightarrow \mathbf{U}_{s'}(x_i) \llbracket \top \rrbracket \mid g \text{ hides position } i\} \quad (2)$$

$$\cup \{\mathbf{U}_s(h) \rightarrow h^\sharp \llbracket \top \rrbracket \mid h \text{ is a hidden term}\} \quad (3)$$

Here, s and s' are the sorts of x , $g(x_1, \dots, x_i, \dots, x_n)$, x_i , and h , respectively, and $\mathbf{U}_s \in \mathcal{F}^\sharp$ is a fresh function symbol of arity 1 with sort declaration $s \rightarrow \text{top}$. Furthermore, $\mu(\mathbf{U}_s) = \emptyset$ and $\mu(f^\sharp) = \mu(f)$ for all $f^\sharp \in \mathcal{F}^\sharp$.

Example 11.19. For the CS-CERS from Example 11.1, the context-sensitive dependency pairs are as follows:

$$\text{take}^\sharp(x, \text{ins}(y, ys)) \rightarrow \text{take}^\sharp(x - 1, ys) \llbracket x > 0 \rrbracket \quad (11.1)$$

$$\text{take}^\sharp(x, \text{ins}(y, ys)) \rightarrow \mathbf{U}_{\text{base}}(y) \llbracket x > 0 \rrbracket \quad (11.2)$$

$$\text{take}^\sharp(x, \text{ins}(y, ys)) \rightarrow \mathbf{U}_{\text{univ}}(ys) \llbracket x > 0 \rrbracket \quad (11.3)$$

$$\text{pick}^\sharp(\text{ins}(x, xs)) \rightarrow \mathbf{U}_{\text{base}}(x) \quad (11.4)$$

$$\text{drop}^\sharp(\text{ins}(x, xs)) \rightarrow \mathbf{U}_{\text{univ}}(ys) \quad (11.5)$$

$$\mathbf{U}_{\text{univ}}(\text{from}(x + 1)) \rightarrow \text{from}^\sharp(x + 1) \quad (11.6)$$

$$\mathbf{U}_{\text{base}}(x + y) \rightarrow \mathbf{U}_{\text{base}}(x) \quad (11.7)$$

$$\mathbf{U}_{\text{base}}(x + y) \rightarrow \mathbf{U}_{\text{base}}(y) \quad (11.8)$$

$$\mathbf{U}_{\text{base}}(-x) \rightarrow \mathbf{U}_{\text{base}}(x) \quad (11.9)$$

$$\mathbf{U}_{\text{univ}}(\text{from}(x)) \rightarrow \mathbf{U}_{\text{base}}(x) \quad (11.10)$$

For this, recall the hidden term $\text{from}(x + 1)$ from Example 11.14 and the hiding contexts from Example 11.17. △

Chapter 11. Context-Sensitive Rewriting and Dependency Pairs

Just as in Definition 5.3, context-sensitive dependency pairs can be used in order to build chains, and the goal is to show the analogous statement to Theorem 5.6, i.e., that $\xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu}$ is terminating if there are no infinite minimal chains.

Definition 11.20 ((Minimal) $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -Chains). *Let \mathcal{P} be a set of dependency pairs and let $(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ be a CS-CERS. A (possibly infinite) sequence of dependency pairs $s_1 \rightarrow t_1[\varphi_1], s_2 \rightarrow t_2[\varphi_2], \dots$ from \mathcal{P} is a $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -chain iff there exists a Th-based substitution σ such that $t_i\sigma \xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \xrightarrow{>\Lambda}_1_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} s_{i+1}\sigma$, the instantiated Th-constraint $\varphi_i\sigma$ is Th-valid, and $s_i\sigma$ is a normal form w.r.t. $\xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}}$ for all $i \geq 1$. The above $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -chain is minimal iff $t_i\sigma$ does not start an infinite $\xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction for all $i \geq 1$.*

While the definition of chains is virtually identical to the non-context-sensitive case in Definition 5.3, proving the analogous statement to Theorem 5.6 for CS-CERSs is quite complex and requires several technical definitions and lemmas. First, it is convenient to formally introduce *minimal non-terminating terms*, which are terms that start an infinite $\xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction such that none of its proper subterms in an active position starts an infinite reduction. This concept is implicitly used in the proof of Theorem 5.6 as well, recall the discussion in Section 5.1.

Definition 11.21 (Minimal Non-Terminating Terms). *For a CS-CERS $(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$, a term t is minimal non-terminating, written $t \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^{\infty}$, iff t starts an infinite $\xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction but no t' with $t \triangleright_{\mu} t'$ starts an infinite $\xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction.*

The following properties of minimal non-terminating terms are easy consequences of Corollary 11.11.

Lemma 11.22. *Let $t \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^{\infty}$.*

1. *If $t \sim_{\mathcal{E}} t'$, then $t' \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^{\infty}$.*
2. *If $t \rightarrow_{\mathcal{E}\setminus\mathcal{S}}^* t'$ and t' starts an infinite $\xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction, then $t' \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^{\infty}$.*

Chapter 11. Context-Sensitive Rewriting and Dependency Pairs

The next lemma intuitively states that the application of equations from \mathcal{E} or rules from \mathcal{S} transforms a hiding context into another hiding context. This lemma is the reason why any $f \in \mathcal{F}(\mathcal{E}) \cup \mathcal{F}(\mathcal{S})$ is considered to hide any active argument position in Definition 11.16.

Lemma 11.23. *Let C be a hiding context and let $t \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^\infty$.*

1. *If $C[t] \vdash_{\mathcal{E}} s$ and the $\vdash_{\mathcal{E}}$ -step is applied at a position in C , then $s = C'[t]$ for a hiding context C' .*
2. *Let $C[t] \rightarrow_{\mathcal{S}} s$ at a position in C using a rule $l \rightarrow r \in \mathcal{S}$ and a substitution σ such that the variable $x \in \mathcal{V}(l)$ with $x\sigma \succeq_{\mu} t$ (if it exists) satisfies $r \succeq_{\mu} x$. Then $s = C'[t]$ for a hiding context C' .*

The proof of the main theorem, i.e., that $\xrightarrow{\mathcal{S}}_{\text{Th} \parallel \mathcal{E} \setminus \mathcal{R}, \mu}$ is terminating if there are no infinite minimal chains, is modularized by introducing the following abstract property. A term has the *hiding property* if its minimal non-terminating subterms in inactive positions are obtained from hidden terms and surrounded by hiding contexts. The definition in this dissertation differs from the original definition in [1] by making use of \mathcal{E} and \mathcal{S} in order to obtain the minimal non-terminating terms from instances of hidden terms.

Definition 11.24 (Hiding Property). *A term $u \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^\infty$ has the hiding property iff whenever $u \triangleright_{\neg\mu} s \succeq_{\mu} t$ with $t \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^\infty$, then $s = C[t]$ for a hiding context C and there exists an instance t' of a hidden term such that $t' \xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}}^* \circ \succ_{\mathcal{E}}^{\geq \Lambda} t$.*

The following key lemma states that the hiding property is preserved by $\sim_{\mathcal{E}}$, $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$, and $\xrightarrow{\mathcal{S}}_{\text{Th} \parallel \mathcal{E} \setminus \mathcal{R}, \mu}$. This is a key result needed for the proof of the main theorem.

Lemma 11.25. *Let $u \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^\infty$ have the hiding property.*

1. *If $u \vdash_{\mathcal{E}} v \succeq_{\mu} w$ with $w \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^\infty$, then w has the hiding property.*

2. If $u \rightarrow_{\mathcal{E} \setminus \mathcal{S}} v \succeq_{\mu} w$ with $w \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^{\infty}$, then w has the hiding property.
3. If $u \xrightarrow{\mathcal{S}}_{\text{Th} \parallel_{\mathcal{E} \setminus \mathcal{R}, \mu}} v \succeq_{\mu} w$ with $w \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^{\infty}$, then w has the hiding property.

Now the main theorem can be proved, stating that rewriting with a CS-CERS is terminating if there are no infinite chains. It could be investigated in future work whether the converse of this statement is true as well, i.e., whether the absence of infinite minimal chains provides an exact characterization of termination for rewriting with CS-CERSs. Recall that this is the case for non-context-sensitive CERSs, cf. Theorem 5.6.

Theorem 11.26. *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ be a CS-CERS. Then $\xrightarrow{\mathcal{S}}_{\text{Th} \parallel_{\mathcal{E} \setminus \mathcal{R}, \mu}}$ is terminating if there are no infinite minimal $(\text{DP}(\mathcal{R}, \mu), \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -chains.*

The DP framework as introduced in Section 5.2 trivially extends to the context-sensitive setting, resulting in the *CS-DP framework*, where *CS-DP processors* operate on *CS-DP problems* of the form $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$.

11.4 Summary

This chapter has introduced the basic terminology of context-sensitive rewriting with CERSs. For CS-CERSs, the arguments where an evaluation may take place are specified for each function symbol and a reduction is only allowed if it takes place at a position that is not forbidden by a function symbol occurring somewhere above it.

Next, a dependency pair method for showing termination of rewriting with such CS-CERSs has been developed. In order to avoid the need for collapsing dependency pairs as in [2], the recent approach from [1] for ordinary TRSs has been extended to the setting of CS-CERSs. For this approach, the notions of *hidden terms* and *hiding contexts* are crucial, resulting in a definition of dependency pairs that is more

Chapter 11. Context-Sensitive Rewriting and Dependency Pairs

complex than in the non-context-sensitive case. This added complexity will pay off in the next chapter, however, since an adaptation of the DP processors from Chapters 6–8 becomes relatively straightforward.

Chapter 12

Context-Sensitive DP Processors

After introducing the dependency pair framework for context-sensitive rewriting with CERSs in Chapter 11, the goal of this chapter is to introduce several sound CS-DP processors.

Section 12.1 introduces *context-sensitive dependency graphs*, which adapt the dependency graphs from Section 6.3 to the context-sensitive case. This adaptation includes an estimation similar to the one used in Section 6.3 for the non-context-sensitive case.

Next, Section 12.2 adapts the subterm criterion from Section 6.6 to the context-sensitive case. As for the context-sensitive dependency graphs, this adaptation is relatively straightforward. As shown in Section 12.3, the methods based on reduction pairs as introduced in Chapter 7 also extend easily.

Extending the method from Section 8.3 that is based on function dependencies and makes it possible to restrict attention to certain subsets of \mathcal{R} , \mathcal{S} , and \mathcal{E} when considering the CS-DP problem $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ is more challenging. For ordinary context-sensitive TRSs, the goal was achieved only very recently [92, 1]. Section 12.4

presents an extension of this method to CS-CERSs, and Section 12.5 presents an improved method for a restricted class of CS-DP problems.

12.1 Dependency Graphs

Like the DP processor from Section 6.3, the CS-DP processor introduced in this section decomposes a CS-DP problem into several independent CS-DP problems by determining which dependency pairs from \mathcal{P} may follow each other in a $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -chain. The processor again relies on the notion of *dependency graphs*.

Definition 12.1 (Context-Sensitive Dependency Graphs). *Let $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ be a CS-DP problem. The $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -dependency graph $\text{DG}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ has the dependency pairs in \mathcal{P} as nodes and there is an arc from $s_1 \rightarrow t_1[\varphi_1]$ to $s_2 \rightarrow t_2[\varphi_2]$ iff $s_1 \rightarrow t_1[\varphi_1]$, $s_2 \rightarrow t_2[\varphi_2]$ is a $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -chain.*

As for the dependency graph from Section 6.3, $\text{DG}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ cannot be computed exactly in general and an estimation has to be used instead. The estimation used in this section is similar to the estimation of Section 6.3 but has been adapted to the context-sensitive case. This adaptation is similar to the estimated dependency graphs for ordinary context-sensitive rewriting used in [2, 1].

Definition 12.2 (Estimated Context-Sensitive Dependency Graphs). *For a CS-DP problem $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$, the nodes in the estimated $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -dependency graph $\text{EDG}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ are the dependency pairs in \mathcal{P} and there is an arc from $s_1 \rightarrow t_1[\varphi_1]$ to $s_2 \rightarrow t_2[\varphi_2]$ iff there exists a substitution σ that is *Th*-based for $\mathcal{V}(s_1) \cup \mathcal{V}(s_2)$ such that $\text{CAP}_\mu(t_1)\sigma \xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}} s_2 \circ \overset{\geq \Lambda}{\sim}_{\mathcal{E}} s_2\sigma$, the terms $s_1\sigma$ and $s_2\sigma$ are normal forms w.r.t. $\xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}}$, and $\varphi_1\sigma$ and $\varphi_2\sigma$ are *Th*-valid. The function CAP_μ is defined by*

1. $\text{CAP}_\mu(x) = x$ for variables x of sort **base**,

Chapter 12. Context-Sensitive DP Processors

2. $\text{CAP}_\mu(x) = y$ for variables x of sort **univ**,

3. $\text{CAP}_\mu(f(t_1, \dots, t_n)) = f(t'_1, \dots, t'_n)$ where

$$t'_i = \begin{cases} t_i & \text{if } i \notin \mu(f) \\ \text{CAP}_\mu(t_i) & \text{otherwise} \end{cases}$$

if there does not exist a rule $l \rightarrow r[\varphi] \in \mathcal{R}$ such that $f(t'_1, \dots, t'_n)\sigma \xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}} \circ \xrightarrow{\geq \Lambda}_{\mathcal{E}} l\sigma$ for a substitution σ that is *Th*-based for $\mathcal{V}(f(t_1, \dots, t_n)) \cup \mathcal{V}(l)$ where $\varphi\sigma$ is *Th*-valid, and

4. $\text{CAP}_\mu(f(t_1, \dots, t_n)) = y$ otherwise.

Here, y is the next variable in an infinite list y_1, y_2, \dots of fresh variables.

As in Section 6.3, it is also possible to omit the checks for irreducibility by $\xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}}$ and *Th*-validity, and it is possible to replace case 3 in the definition of CAP_μ by a simple check for $f \notin \mathcal{D}(\mathcal{R})$.

Next, it is shown that the estimated dependency graph is indeed an overapproximation of the dependency graph, i.e., $\text{EDG}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ is a supergraph of $\text{DG}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$.

Theorem 12.3 (Correctness of EDG). *For any DP problem $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$, the estimated dependency graph $\text{EDG}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ is a supergraph of the dependency graph $\text{DG}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$.*

The following CS-DP processor is completely analogous to the DP processor from Theorem 6.11.

Theorem 12.4 (CS-DP Processor Based on Dependency Graphs). *Let Proc be a CS-DP processor with $\text{Proc}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu) = \{(\mathcal{P}_1, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu), \dots, (\mathcal{P}_n, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)\}$, where $\mathcal{P}_1, \dots, \mathcal{P}_n$ are the SCCs of $(\text{E})\text{DG}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$. Then Proc is sound.*

Example 12.5. Recall the following dependency pairs from Example 11.19:

$$\text{take}^\sharp(x, \text{ins}(y, ys)) \rightarrow \text{take}^\sharp(x - 1, ys) \llbracket x > 0 \rrbracket \quad (11.1)$$

$$\text{take}^\sharp(x, \text{ins}(y, ys)) \rightarrow \text{U}_{\text{base}}(y) \llbracket x > 0 \rrbracket \quad (11.2)$$

$$\text{take}^\sharp(x, \text{ins}(y, ys)) \rightarrow \text{U}_{\text{univ}}(ys) \llbracket x > 0 \rrbracket \quad (11.3)$$

$$\text{pick}^\sharp(\text{ins}(x, xs)) \rightarrow \text{U}_{\text{base}}(x) \quad (11.4)$$

$$\text{drop}^\sharp(\text{ins}(x, xs)) \rightarrow \text{U}_{\text{univ}}(ys) \quad (11.5)$$

$$\text{U}_{\text{univ}}(\text{from}(x + 1)) \rightarrow \text{from}^\sharp(x + 1) \quad (11.6)$$

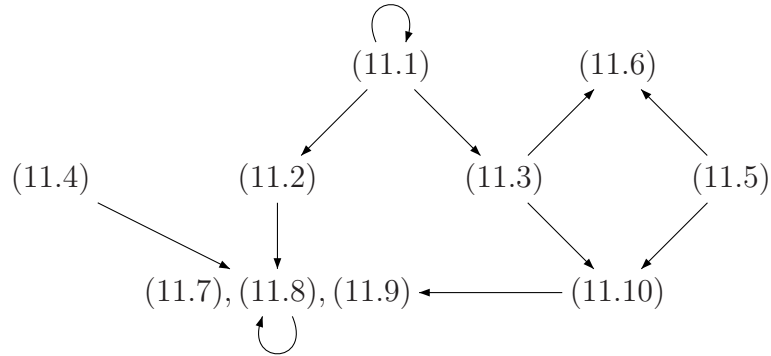
$$\text{U}_{\text{base}}(x + y) \rightarrow \text{U}_{\text{base}}(x) \quad (11.7)$$

$$\text{U}_{\text{base}}(x + y) \rightarrow \text{U}_{\text{base}}(y) \quad (11.8)$$

$$\text{U}_{\text{base}}(-x) \rightarrow \text{U}_{\text{base}}(x) \quad (11.9)$$

$$\text{U}_{\text{univ}}(\text{from}(x)) \rightarrow \text{U}_{\text{base}}(x) \quad (11.10)$$

Then the following estimated dependency graph $\text{EDG}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ is obtained:



Here, the nodes for (11.7), (11.8), and (11.9) have been combined since they have “identical” incoming and outgoing arcs. This estimated dependency graph contains two SCCs, and according to Theorem 12.4, the following CS-DP problems are obtained:

$$(\{(11.1)\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu) \quad (12.1)$$

$$(\{(11.7), (11.8), (11.9)\}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu) \quad (12.2)$$

These CS-DP problem can now be handled independently of each other. \triangle

12.2 Subterm Criterion

The DP processor from Section 6.6 that is based on the subterm criterion can also be extended to the context-sensitive case in a straightforward manner. The only difference is that the subterm relation needs to take the replacement map into account by only considering subterms in active positions. This is analogous to [2].

Definition 12.6 (\mathcal{E} - μ -Subterms). *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ be a CS-CERS and let s, t be terms. Then t is a strict \mathcal{E} - μ -subterm of s , written $s \triangleright_{\mathcal{E}, \mu} t$, iff $s \sim_{\mathcal{E}} \circ \triangleright_{\mu} \circ \sim_{\mathcal{E}} t$. The term t is an \mathcal{E} - μ -subterm of s , written $s \triangleright_{\mathcal{E}, \mu} t$, iff $s \triangleright_{\mathcal{E}, \mu} t$ or $s \sim_{\mathcal{E}} t$.*

Example 12.7. If $\mu(f) = \{1\}$, $\mu(\cup) = \{1, 2\}$, and $\mathcal{E} = \{x \cup y \approx y \cup x, x \cup (y \cup z) \approx (x \cup y) \cup z\}$, then $f(x \cup y, g(z)) \triangleright_{\mathcal{E}, \mu} y \cup x$ but $f(x \cup y, g(z)) \not\triangleright_{\mathcal{E}, \mu} g(z)$. \triangle

If \mathcal{E} is size-preserving, the properties of Lemma 6.23 still hold for the subterm relation defined above.

Lemma 12.8. *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ be a CS-CERS such that \mathcal{E} is size-preserving.*

1. *Given terms s, t , it is decidable whether $s \triangleright_{\mathcal{E}, \mu} t$ or $s \triangleright_{\mathcal{E}, \mu} t$.*
2. *$\triangleright_{\mathcal{E}, \mu}$ is well-founded.*
3. *$\triangleright_{\mathcal{E}, \mu}$ and $\triangleright_{\mathcal{E}, \mu}$ are stable.*
4. *$\triangleright_{\mathcal{E}, \mu}$ and $\triangleright_{\mathcal{E}, \mu}$ are compatible with $\sim_{\mathcal{E}}$.*

Now the DP processor from Theorem 6.24 extends to the context-sensitive case in the obvious way.

Theorem 12.9 (CS-DP Processor Based on the Subterm Criterion). *Let π be a simple projection and let Proc be a CS-DP processor with $\text{Proc}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu) =$*

- $\{(\mathcal{P} - \mathcal{P}', \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, *if \mathcal{E} is size-preserving and $\mathcal{P}' \subseteq \mathcal{P}$ such that*

– $\pi(s) \triangleright_{\mathcal{E}, \mu} \pi(t)$ for all $s \rightarrow t[[\varphi]] \in \mathcal{P}'$, and

– $\pi(s) \triangleright_{\mathcal{E}, \mu} \pi(t)$ for all $s \rightarrow t[[\varphi]] \in \mathcal{P} - \mathcal{P}'$.

- $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$, otherwise.

Then Proc is sound.

Example 12.10. Recall the CS-DP problem (12.2) from Example 12.5, consisting of the following dependency pairs:

$$\mathbf{U}_{\text{base}}(x + y) \rightarrow \mathbf{U}_{\text{base}}(x) \quad (11.7)$$

$$\mathbf{U}_{\text{base}}(x + y) \rightarrow \mathbf{U}_{\text{base}}(y) \quad (11.8)$$

$$\mathbf{U}_{\text{base}}(-x) \rightarrow \mathbf{U}_{\text{base}}(x) \quad (11.9)$$

Using the simple projection with $\pi(\mathbf{U}_{\text{base}}) = 1$, this CS-DP problem can easily be handled. △

In general, if all dependency pairs in a CS-DP problem are of type (2) in Definition 11.18 (i.e., are of the form $\mathbf{U}_s(g(x_1, \dots, x_i, \dots, x_n)) \rightarrow \mathbf{U}_{s'}(x_i)$ where g hides position i), then that CS-DP problem can always be handled using the subterm criterion with the simple projection $\pi(\mathbf{U}_{\text{base}}) = \pi(\mathbf{U}_{\text{univ}}) = 1$. To see this, notice that $i \in \mu(g)$ if g hides position i and thus $g(x_1, \dots, x_i, \dots, x_n) \triangleright_{\mathcal{E}, \mu} x_i$.

12.3 Reduction Pairs

As in Chapter 7, three different kinds of reduction pairs can be used in the context-sensitive case. Using these, dependency pairs that are strictly decreasing can be removed from a CS-DP problem, just like this was done in Chapter 7 in the non-context-sensitive case.

Theorem 12.11 (CS-DP Processor Based on Reduction Pairs). *Let Proc be a CS-DP processor with $\text{Proc}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu) =$*

- $\{(\mathcal{P} - \mathcal{P}', \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, if $\mathcal{P}' \subseteq \mathcal{P}$ and either
 - there exists an ordinary reduction pair (\succsim, \succ) such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \emptyset, \mathcal{R}, \mathcal{S}, \mathcal{E})$, or
 - there exists a $\text{Th}_{\mathbb{N}}$ -reduction pair (\succsim, \succ) for \mathcal{P} such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \emptyset, \mathcal{R}, \mathcal{S}, \mathcal{E})$, or
 - there exists a $\text{Th}_{\mathbb{Z}}$ -reduction pair (\succsim, \succ) for \mathcal{P} such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \emptyset, \mathcal{R}, \mathcal{S}, \mathcal{E})$.
- $\{(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)\}$, otherwise.

Then Proc is sound.

Also, the elimination of arguments as done in Theorem 6.17 can be applied to CS-DP problems without any modification.

Example 12.12. Recall the CS-DP problem (12.1) from Example 12.5, containing the following dependency pair:

$$\text{take}^{\#}(x, \text{ins}(y, ys)) \rightarrow \text{take}^{\#}(x - 1, ys) \llbracket x > 0 \rrbracket \quad (11.1)$$

Using the non-collapsing argument filtering with $\pi(\text{take}^{\#}) = [1]$, this dependency pair is transformed into the following dependency pair:

$$\text{take}^{\#}(x) \rightarrow \text{take}^{\#}(x - 1) \llbracket x > 0 \rrbracket \quad (12.3)$$

For this dependency pair, $\text{RedPos}(\text{take}^{\#}, \{(12.3)\}) = \emptyset$, i.e., $\text{Th}_{\mathbb{Z}}$ -reduction pairs become applicable. △

12.4 Function Dependencies

The DP processor from Theorem 12.11 has to consider all of \mathcal{R} . Recall from Section 8.3 that this requirement can be relaxed substantially in the non-context-sensitive case. This section shows that this requirement can also be weakened for CS-CERSs. In contrast to the simple adaptations of the dependency graph and the subterm criterion, this adaptation is non-trivial. For ordinary context-sensitive rewriting, corresponding results were obtained only very recently [92, 1].

As the following example from [92] shows, the method is unsound if the definition of function dependencies from Section 8.3 is used.

Example 12.13. Consider the following ordinary TRS [92]:

$$\begin{aligned} \mathbf{b} &\rightarrow \mathbf{c}(\mathbf{b}) \\ \mathbf{f}(\mathbf{c}(x), x) &\rightarrow \mathbf{f}(x, x) \end{aligned}$$

Let $\mu(\mathbf{f}) = \{1, 2\}$ and $\mu(\mathbf{c}) = \emptyset$. Then the following dependency pairs are obtained:

$$\mathbf{f}^\sharp(\mathbf{c}(x), x) \rightarrow \mathbf{f}^\sharp(x, x) \tag{12.4}$$

$$\mathbf{f}^\sharp(\mathbf{c}(x), x) \rightarrow \mathbf{U}_{\text{univ}}(x) \tag{12.5}$$

$$\mathbf{U}_{\text{univ}}(\mathbf{b}) \rightarrow \mathbf{b}^\sharp \tag{12.6}$$

$$\mathbf{U}_{\text{base}}(x + y) \rightarrow \mathbf{U}_{\text{base}}(x) \tag{12.7}$$

$$\mathbf{U}_{\text{base}}(x + y) \rightarrow \mathbf{U}_{\text{base}}(y) \tag{12.8}$$

$$\mathbf{U}_{\text{base}}(-x) \rightarrow \mathbf{U}_{\text{base}}(x) \tag{12.9}$$

The CS-DP problem consisting of (12.7)–(12.9) can easily be handled using the subterm criterion of Section 12.2. Using the definition of function dependencies from Section 8.3, \mathcal{R} would not need to be considered for the CS-DP problem consisting of (12.4). But then termination could falsely be concluded using a reduction pair based

on a polynomial interpretation with $\mathcal{Pol}(f^\sharp) = x_1$ and $\mathcal{Pol}(c) = x_1 + 1$, although $f(c(\mathbf{b}), \mathbf{b}) \rightarrow_{\mathcal{R}, \mu} f(\mathbf{b}, \mathbf{b}) \rightarrow_{\mathcal{R}, \mu} f(c(\mathbf{b}), \mathbf{b}) \rightarrow_{\mathcal{R}, \mu} \dots$ is an infinite reduction. \triangle

The function dependencies as defined in Definition 8.10 thus need to be adapted as follows, similarly to [92, 1]. Notice that function symbols occurring in inactive positions in left-hand sides of dependency pairs and rules need to be considered.

Definition 12.14 (Context-Sensitive Function Dependencies). *Let $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ be a CS-DP problem where \mathcal{E} is size-preserving. For two symbols $f, h \in \mathcal{F}$ let $f \blacktriangleright_{(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^1 h$ iff $f = h$ or there exists a symbol g with $g \blacktriangleright_{(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^1 h$ and a rule $l \rightarrow r[\varphi] \in \mathcal{R}$ such that $\text{root}(l) = f$ and $g \in \mathcal{F}^{-\mu}(l) \cup \mathcal{F}(r)$. Let*

$$\begin{aligned} \Delta^1(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu) &= \mathcal{F}_{Th} \cup \mathcal{F}(\mathcal{S}) \cup \mathcal{F}(\mathcal{E}) \\ &\cup \bigcup_{s \rightarrow t[\varphi] \in \mathcal{P}} \{g \mid f \blacktriangleright_{(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^1 g \text{ for an } f \in \mathcal{F}^{-\mu}(s) \cup \mathcal{F}(t)\} \\ &\cup \bigcup_{l \rightarrow r[\varphi] \in \mathcal{R}} \{g \mid f \blacktriangleright_{(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^1 g \text{ for an } f \in \mathcal{F}^{-\mu}(r)\} \end{aligned}$$

Within this section it is assumed that $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ is a CS-DP problem such that \mathcal{E} is size-preserving. Furthermore, let $\Delta^1 = \Delta^1(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$. The mapping \mathcal{I} used in Section 8.3 needs to be modified as well. In particular, it needs to be possible to apply the mapping to non-terminating terms since terminating terms may contain non-terminating subterms in inactive positions.

Definition 12.15 (\mathcal{I}^1). *For any term $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{Th}, \mathcal{V})$ define $\mathcal{I}^1(t)$ by*

- $\mathcal{I}^1(x) = x$ if $x \in \mathcal{V}$
- $\mathcal{I}^1(f(t_1, \dots, t_n)) = f(\mathcal{I}^1(t_1), \dots, \mathcal{I}^1(t_n))$ if $f \in \Delta^1$ or $f(t_1, \dots, t_n)$ starts an infinite $\xrightarrow{S}_{Th \parallel \mathcal{E} \setminus \mathcal{R}, \mu}$ -reduction
- $\mathcal{I}^1(t) = \text{Comp}_{\text{sort}(t)}(\mathcal{R}ed_{\mathcal{S}}^1(t) \cup \mathcal{R}ed_{\mathcal{R}}^1(t) \cup \mathcal{E}q_{\mathcal{E}}^1(t))$ if t is terminating and $t = f(t_1, \dots, t_n)$ with $f \notin \Delta^1$.

Here, the sets $\mathcal{Red}_S^1(t)$, $\mathcal{Red}_R^1(t)$, and $\mathcal{Eq}_E^1(t)$ are defined as

$$\begin{aligned}\mathcal{Red}_S^1(t) &= \{\mathcal{I}^1(t') \mid t \rightarrow_{\mathcal{E} \setminus S} \circ \sim_{\mathcal{E}} t'\} \\ \mathcal{Red}_R^1(t) &= \{\mathcal{I}^1(t') \mid t \xrightarrow{S}_{Th \parallel \mathcal{E} \setminus R, \mu} \circ \sim_{\mathcal{E}} t'\} \\ \mathcal{Eq}_E^1(t) &= \{g(\mathcal{I}^1(t_1), \dots, \mathcal{I}^1(t_m)) \mid t \sim_{\mathcal{E}} g(t_1, \dots, t_m)\}\end{aligned}$$

The function Comp_s is defined as in Definition 8.13. For any substitution σ , define the substitution $\mathcal{I}^1(\sigma)$ by $\mathcal{I}^1(\sigma)(x) = \mathcal{I}^1(\sigma(x))$.

Again, it is not obvious that \mathcal{I}^1 is indeed well-defined, i.e., that $\mathcal{I}^1(t)$ is a finite term for any term t .

Lemma 12.16. *For any term t , the term $\mathcal{I}^1(t)$ is finite.*

The idea for the result in this section is now the same as in Section 8.3: reductions using \mathcal{R} are “simulated” by reductions that only use $\mathcal{R}(\Delta)$ and rules from \mathcal{R}_{Π} as defined in Definition 8.15. For this, several properties of the mapping \mathcal{I}^1 are needed, in complete analogy to Lemma 8.17.

Lemma 12.17. *Let $s, t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{Th}, \mathcal{V})$ and let σ be a Th -based substitution.*

1. *If $s \in \mathcal{T}(\Delta, \mathcal{V})$ then $\mathcal{I}^1(s\sigma) = s\mathcal{I}^1(\sigma)$.*
2. *If $s \triangleright_{-\mu} s'$ implies $s' \in \mathcal{T}(\Delta^1, \mathcal{V})$, then $\mathcal{I}^1(s\sigma) \rightarrow_{\mathcal{R}_{\Pi}}^* s\mathcal{I}^1(\sigma)$.*
3. *If $s \sim_{\mathcal{E}} t$ then $\mathcal{I}^1(s) \sim_{\mathcal{E}} \mathcal{I}^1(t)$.*
4. *If $s \rightarrow_{\mathcal{E} \setminus S}^* t$ then $\mathcal{I}^1(s) \rightsquigarrow_1^* \mathcal{I}^1(t)$,
where $\rightsquigarrow_1 = \rightarrow_{\mathcal{E} \setminus S} \cup \rightarrow_{\mathcal{R}_{\Pi}}^+$.*
5. *If s is terminating and $s \xrightarrow{S}_{Th \parallel \mathcal{E} \setminus R, \mu}^* t$ then $\mathcal{I}^1(s) \rightsquigarrow_2^* \mathcal{I}^1(t)$,
where $\rightsquigarrow_2 = \rightsquigarrow_1^* \circ \sim_{\mathcal{E}} \circ \rightarrow_{\mathcal{R}_{\Pi}}^* \circ \rightarrow_{\mathcal{R}(\Delta^1), \mu} \cup \rightarrow_{\mathcal{R}_{\Pi}}^+$ such that the $\rightarrow_{\mathcal{R}(\Delta^1), \mu}$ step uses a Th -based substitution that makes the instantiated constraint of the used rule Th -valid.*

Chapter 12. Context-Sensitive DP Processors

6. Let $s \in \mathcal{T}(\Delta^1, \mathcal{V})$ be terminating and let $t \triangleright_{\neg\mu} t'$ imply $t' \in \mathcal{T}(\Delta^1, \mathcal{V})$. If $s\sigma \xrightarrow{\mathcal{S}}^*_{Th\|\mathcal{E}\backslash\mathcal{R}, \mu} \circ \rightarrow^!_{\mathcal{E}\backslash\mathcal{S}} \circ \sim_{\mathcal{E}} t\sigma$, then $s\mathcal{I}^1(\sigma) \rightsquigarrow_2^* \circ \rightsquigarrow_1^* \circ \sim_{\mathcal{E}} \circ \rightarrow^*_{\mathcal{R}_{\Pi}} t\mathcal{I}^1(\sigma)$.

Using Lemma 12.17.6, soundness of the following CS-DP processor can be shown. This processor adapts Theorem 8.18 to the context-sensitive case.

Theorem 12.18 (CS-DP Processor Based on Function Dependencies). *Let Proc be the DP processor with $\text{Proc}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu) =$*

- $\{(\mathcal{P} - \mathcal{P}', \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, if \mathcal{E} is size-preserving, $\Delta^1 = \Delta^1(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$, $\mathcal{P}' \subseteq \mathcal{P}$, and either
 - there exists an ordinary reduction pair (\succsim, \succ) such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \emptyset, \mathcal{R}(\Delta^1) \cup \mathcal{R}_{\Pi}, \mathcal{S}, \mathcal{E})$, or
 - there exists a $Th_{\mathbb{N}}$ -reduction pair (\succsim, \succ) for \mathcal{P} such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \emptyset, \mathcal{R}(\Delta^1) \cup \mathcal{R}_{\Pi}, \mathcal{S}, \mathcal{E})$, or
 - there exists a $Th_{\mathbb{Z}}$ -reduction pair (\succsim, \succ) for \mathcal{P} such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \emptyset, \mathcal{R}(\Delta^1) \cup \mathcal{R}_{\Pi, \text{univ}}, \mathcal{S}, \mathcal{E})$.
- $\{(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)\}$, otherwise.

Then Proc is sound.

Example 12.19. Using Theorem 12.18, it is no longer possible to falsely prove termination of the TRS from Example 12.13. Recall the following rewrite rules and dependency pairs:

$$\begin{aligned}
 \mathbf{b} &\rightarrow \mathbf{c}(\mathbf{b}) \\
 \mathbf{f}(\mathbf{c}(x), x) &\rightarrow \mathbf{f}(x, x) \\
 \mathbf{f}^{\#}(\mathbf{c}(x), x) &\rightarrow \mathbf{f}^{\#}(x, x)
 \end{aligned} \tag{12.4}$$

Then $\mathbf{b} \in \Delta^1(\{(12.4), \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu\})$, i.e., the rule $\mathbf{b} \rightarrow \mathbf{c}(\mathbf{b})$ needs to be considered when handling the CS-DP consisting of (12.4). But then no reduction pair satisfies the requirements from Theorem 12.18. \triangle

12.5 Function Dependencies for Strongly Conservative Systems

The CS-DP processor based on function dependencies introduced in the previous section differs from the corresponding DP processor for non-context-sensitive rewriting introduced in Section 8.3 by needing to consider the left-hand sides of the dependency pairs and rules. Additionally, all rules from \mathcal{S} and all equations from \mathcal{E} need to be considered.

In this section, it is shown that the left-hand sides of \mathcal{P} and \mathcal{R} do not need to be considered if \mathcal{P} , \mathcal{R} and \mathcal{S} are *strongly conservative* [92].

Definition 12.20 (Strongly Conservative). *Let $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ be a DP problem and let s, t be terms. Then the pair $\langle s, t \rangle$ is strongly conservative iff $\mathcal{V}^\mu(t) \subseteq \mathcal{V}^\mu(s)$ and $\mathcal{V}^\mu(s) \cap \mathcal{V}^{-\mu}(s) = \mathcal{V}^\mu(t) \cap \mathcal{V}^{-\mu}(t) = \emptyset$.*

Example 12.21. Consider a replacement map with $\mu(\mathbf{f}) = \{1\}$, $\mu(\mathbf{g}) = \{1, 2\}$, and $\mu(\mathbf{h}) = \emptyset$. Then the pair $\langle \mathbf{f}(\mathbf{g}(x, y), \mathbf{h}(z)), \mathbf{f}(x, y) \rangle$ is strongly conservative since $\mathcal{V}^\mu(\mathbf{f}(x, y)) = \{x\} \subseteq \{x, y\} = \mathcal{V}^\mu(\mathbf{f}(\mathbf{g}(x, y), \mathbf{h}(z)))$ and $\{x, y\} \cap \{z\} = \{x\} \cap \{y\} = \emptyset$. The pair $\langle \mathbf{f}(x, \mathbf{h}(x)), \mathbf{g}(x, x) \rangle$ is not strongly conservative since $\mathcal{V}^\mu(\mathbf{f}(x, \mathbf{h}(x))) \cap \mathcal{V}^{-\mu}(\mathbf{f}(x, \mathbf{h}(x))) = \{x\}$. The pair $\langle \mathbf{f}(\mathbf{h}(x), \mathbf{h}(y)), \mathbf{g}(x, x) \rangle$ is not strongly conservative since $x \in \mathcal{V}^\mu(\mathbf{g}(x, x))$ but $x \notin \mathcal{V}^\mu(\mathbf{f}(\mathbf{h}(x), \mathbf{h}(y)))$. \triangle

A set of pairs of terms (i.e., rewrite rules or equations) is strongly conservative if all of its members are. Notice that \mathcal{E} is always strongly conservative by Definition

11.7. For strongly conservative systems, the following definition of function dependencies can be used. The reason why *two* different relations $\overrightarrow{\triangleright}^2$ and \triangleright^2 are used is that reductions with \mathcal{S} and \mathcal{E} may also take place in inactive positions. Introducing two different relations then makes it possible to only consider subsets of \mathcal{S} and \mathcal{E} as well, whereas the CS-DP processor introduced in the previous section needs to consider all of \mathcal{S} and \mathcal{E} .

Definition 12.22 (Context-Sensitive Function Dependencies for Strongly Conservative Systems). *Let $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ be a DP problem where \mathcal{E} is size-preserving. For any term t , let $\mathcal{F}_{\mathcal{S}, \mathcal{E}}(t) = (\mathcal{F}(\mathcal{S}) \cup \mathcal{F}(\mathcal{E})) \cap \mathcal{F}(t)$ and $\mathcal{F}_{\mathcal{S}, \mathcal{E}}^\mu(t) = \mathcal{F}^\mu(t) \cup \mathcal{F}_{\mathcal{S}, \mathcal{E}}(t)$.*

1. For two symbols $f, h \in \mathcal{F}$ let $f \overrightarrow{\triangleright}_{(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^2 h$ iff $f = h$ or there exists a symbol g with $g \overrightarrow{\triangleright}_{(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^2 h$ and either
 - (a) a rule $l \rightarrow r[\![\varphi]\!] \in \mathcal{R}$ with $\text{root}(l) = f$ and $g \in \mathcal{F}(r)$, or
 - (b) a rule $l \rightarrow r \in \mathcal{S}$ with $\text{root}(l) = f$ and $g \in \mathcal{F}(r)$, or
 - (c) an equation $u \approx v$ (or $v \approx u$) in \mathcal{E} with $\text{root}(u) = f$ and $g \in \mathcal{F}(u \approx v)$.

In the following, let

$$\overline{\Delta}^2(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu) = \mathcal{F}_{Th} \cup \bigcup_{s \rightarrow t[\![\varphi]\!] \in \mathcal{P}} \{g \mid f \overrightarrow{\triangleright}_{(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^2 g \text{ for an } f \in \mathcal{F}(t)\}$$

2. For two symbols $f, h \in \mathcal{F}$ let $f \triangleright_{(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^2 h$ iff $f = h$ or there exists a symbol g with $g \triangleright_{(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^2 h$ and either
 - (a) a rule $l \rightarrow r[\![\varphi]\!] \in \mathcal{R}$ with $\text{root}(l) = f$ and $g \in \mathcal{F}_{\mathcal{S}, \mathcal{E}}^\mu(r)$, or
 - (b) a rule $l \rightarrow r \in \mathcal{S}$ with $\text{root}(l) = f$ and $g \in \mathcal{F}(r)$, or
 - (c) an equation $u \approx v$ (or $v \approx u$) in \mathcal{E} with $\text{root}(u) = f$ and $g \in \mathcal{F}(u \approx v)$.

In the following, let

$$\Delta^2(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu) = \mathcal{F}_{Th} \cup \bigcup_{s \rightarrow t[\![\varphi]\!] \in \mathcal{P}} \{g \mid f \triangleright_{(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^2 g \text{ for an } f \in \mathcal{F}_{\mathcal{S}, \mathcal{E}}^\mu(t)\}$$

Within this section it is assumed that $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ is a DP problem such that \mathcal{E} is size-preserving. Also, let $\Delta^2 = \Delta^2(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ and $\overline{\Delta}^2 = \overline{\Delta}^2(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$. Notice that $\Delta^2 \subseteq \overline{\Delta}^2$, which implies $\mathcal{S}(\Delta^2) \subseteq \mathcal{S}(\overline{\Delta}^2)$ and $\mathcal{E}(\Delta^2) \subseteq \mathcal{E}(\overline{\Delta}^2)$. Finally, it is assumed that $\mathcal{R}(\Delta^2)$ and $\mathcal{S}(\overline{\Delta}^2)$ are strongly conservative.

The mappings \mathcal{I} and \mathcal{I}^1 from Definitions 8.13 and 12.15, respectively, need to be modified for the strongly conservative case. Since two relations $\overrightarrow{\triangleright}^2$ and \blacktriangleright^2 are considered, *two* mappings $\overline{\mathcal{I}}^2$ and \mathcal{I}^2 are needed as well. Notice that $\overline{\mathcal{I}}^2$ is only used in order to simulate reductions with \mathcal{S} and \mathcal{E} , but not reductions with \mathcal{R} .

Definition 12.23 ($\overline{\mathcal{I}}^2$ and \mathcal{I}^2).

1. For any term $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{Th}, \mathcal{V})$ define $\overline{\mathcal{I}}^2(t)$ by

- $\overline{\mathcal{I}}^2(x) = x$ if $x \in \mathcal{V}$
- $\overline{\mathcal{I}}^2(f(t_1, \dots, t_n)) = f(\overline{\mathcal{I}}^2(t_1), \dots, \overline{\mathcal{I}}^2(t_n))$ if $f \in \overline{\Delta}^2$
- $\overline{\mathcal{I}}^2(t) = \text{Comp}_{\text{sort}(t)}(\overline{\mathcal{R}ed}_{\mathcal{S}}^2(t) \cup \overline{\mathcal{E}q}_{\mathcal{E}}^2(t))$ if $t = f(t_1, \dots, t_n)$ with $f \notin \overline{\Delta}^2$.

Here, the sets $\overline{\mathcal{R}ed}_{\mathcal{S}}^2(t)$ and $\overline{\mathcal{E}q}_{\mathcal{E}}^2(t)$ are defined as

$$\begin{aligned} \overline{\mathcal{R}ed}_{\mathcal{S}}^2(t) &= \{\overline{\mathcal{I}}^2(t') \mid t \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \circ \sim_{\mathcal{E}} t'\} \\ \overline{\mathcal{E}q}_{\mathcal{E}}^2(t) &= \{g(\overline{\mathcal{I}}^2(t_1), \dots, \overline{\mathcal{I}}^2(t_m)) \mid t \sim_{\mathcal{E}} g(t_1, \dots, t_m)\} \end{aligned}$$

For any substitution σ , define the substitution $\overline{\mathcal{I}}^2(\sigma)$ by letting $\overline{\mathcal{I}}^2(\sigma)(x) = \overline{\mathcal{I}}^2(\sigma(x))$.

2. For any terminating term $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{Th}, \mathcal{V})$ define $\mathcal{I}^2(t)$ by

- $\mathcal{I}^2(x) = x$ if $x \in \mathcal{V}$
- $\mathcal{I}^2(f(t_1, \dots, t_n)) = f(\overline{t}_1, \dots, \overline{t}_n)$ if $f \in \Delta^2$
- $\mathcal{I}^2(t) = \text{Comp}_{\text{sort}(t)}(\mathcal{R}ed_{\mathcal{S}}^2(t) \cup \mathcal{R}ed_{\mathcal{R}}^2(t) \cup \mathcal{E}q_{\mathcal{E}}^2(t) \cup \overline{\overline{\mathcal{E}q}}_{\mathcal{E}}^2(t))$ if $t = f(t_1, \dots, t_n)$ with $f \notin \Delta^2$.

Here, $\bar{t}_i = \mathcal{I}^2(t_i)$ if $i \in \mu(f)$ and $\bar{t}_i = \bar{\mathcal{I}}^2(t_i)$ otherwise. Moreover, the sets $\text{Red}_S^2(t)$, $\text{Red}_R^2(t)$, $\mathcal{E}q_{\mathcal{E}}^2(t)$, and $\bar{\mathcal{E}q}_{\mathcal{E}}^2(t)$ are defined as

$$\begin{aligned} \text{Red}_S^2(t) &= \{\mathcal{I}^2(t') \mid t \rightarrow_{\mathcal{E} \setminus S} \circ \sim_{\mathcal{E}} t'\} \\ \text{Red}_R^2(t) &= \{\mathcal{I}^2(t') \mid t \xrightarrow{S}_{\text{Th} \parallel \mathcal{E} \setminus R, \mu} \circ \sim_{\mathcal{E}} t'\} \\ \mathcal{E}q_{\mathcal{E}}^2(t) &= \{g(\bar{t}_1, \dots, \bar{t}_m) \mid t \sim_{\mathcal{E}} g(t_1, \dots, t_m)\} \\ \bar{\mathcal{E}q}_{\mathcal{E}}^2(t) &= \{\bar{\mathcal{I}}^2(s) \mid t \sim_{\mathcal{E}} s\} \end{aligned}$$

Again, $\bar{t}_i = \mathcal{I}^2(t_i)$ if $i \in \mu(g)$ and $\bar{t}_i = \bar{\mathcal{I}}^2(t_i)$ otherwise. For a terminating substitution σ , let $[t, \sigma]$ be the term that results from t by replacing all occurrences of $x \in \mathcal{V}(t)$ in active positions of t by $\mathcal{I}^2(\sigma(x))$ and all occurrences of $x \in \mathcal{V}(t)$ in inactive positions of t by $\bar{\mathcal{I}}^2(\sigma(x))$.

As usual, it first needs to be shown that these mappings are well-defined.

Lemma 12.24. *For any term t , the term $\bar{\mathcal{I}}^2(t)$ is finite. If t is terminating, then $\mathcal{I}^2(t)$ is finite.*

Next, several properties of the mappings $\bar{\mathcal{I}}^2$ and \mathcal{I}^2 are shown. This is in analogy to Lemmas 8.17 and 12.17.

Lemma 12.25. *Let $s, t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\text{Th}}, \mathcal{V})$ and let σ be a Th -based substitution such that s, t, σ are terminating.*

1. If $s \in \mathcal{T}(\bar{\Delta}^2, \mathcal{V})$ then $\bar{\mathcal{I}}^2(s\sigma) = s\bar{\mathcal{I}}^2(\sigma)$.
2. If $s \in \mathcal{T}(\bar{\Delta}^2, \mathcal{V})$ such that $\mathcal{F}^\mu(s) \subseteq \Delta^2$, then $\mathcal{I}^2(s\sigma) = [s, \sigma]$.
3. $\bar{\mathcal{I}}^2(s\sigma) \rightarrow_{\mathcal{R}_\Pi}^* s\bar{\mathcal{I}}^2(\sigma)$.
4. $\mathcal{I}^2(s\sigma) \rightarrow_{\mathcal{R}_\Pi}^* [s, \sigma]$.
5. $\mathcal{I}^2(s) \rightarrow_{\mathcal{R}_\Pi}^* \bar{\mathcal{I}}^2(s)$.
6. If $s \sim_{\mathcal{E}} t$ then $\bar{\mathcal{I}}^2(s) \sim_{\mathcal{E}(\bar{\Delta}^2)} \bar{\mathcal{I}}^2(t)$.

Chapter 12. Context-Sensitive DP Processors

7. If $s \sim_{\mathcal{E}} t$ then $\mathcal{I}^2(s) \sim_{\mathcal{E}(\overline{\Delta}^2)} \mathcal{I}^2(t)$.
8. If $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}}^* t$ then $\overline{\mathcal{I}}^2(s) \rightsquigarrow_1^* \overline{\mathcal{I}}^2(t)$,
where $\rightsquigarrow_1 = \sim_{\mathcal{E}(\overline{\Delta}^2)} \circ \rightarrow_{\mathcal{R}_{\Pi}}^* \circ \rightarrow_{\mathcal{S}(\overline{\Delta}^2)} \cup \rightarrow_{\mathcal{R}_{\Pi}}^+$.
9. If $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}}^* t$ then $\mathcal{I}^2(s) \rightsquigarrow_1^* \mathcal{I}^2(t)$.
10. If $s \xrightarrow{\mathcal{S}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}, \mu}^* t$ then $\mathcal{I}^2(s) \rightsquigarrow_2^* \mathcal{I}^2(t)$,
where $\rightsquigarrow_2 = \rightsquigarrow_1^* \circ \sim_{\mathcal{E}(\overline{\Delta}^2)} \circ \rightarrow_{\mathcal{R}_{\Pi}}^* \circ \rightarrow_{\mathcal{R}(\Delta^2), \mu} \cup \rightarrow_{\mathcal{R}_{\Pi}}^+$ such that the $\rightarrow_{\mathcal{R}(\Delta^2), \mu}$ step uses a Th -based substitution that makes the instantiated constraint of the used rule Th -valid.
11. If $s \xrightarrow{\mathcal{S}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}, \mu}^* \rightarrow_{\mathcal{E} \setminus \mathcal{S}}^! \circ \sim_{\mathcal{E}} t\sigma$, then $\mathcal{I}^2(s) \rightsquigarrow_2^* \circ \rightsquigarrow_1^* \circ \sim_{\mathcal{E}(\overline{\Delta}^2)} \mathcal{I}^2(t)$.

As before, Lemma 12.25.11 makes it possible to show soundness of the following CS-DP processor. In contrast to the CS-DP processor introduced in the previous section, it is only applicable if $\mathcal{R}(\Delta^2)$, $\mathcal{S}(\Delta^2)$, and \mathcal{P} are strongly conservative.

Theorem 12.26 (CS-DP Processor Based on Function Dependencies for Strongly Conservative Systems). *Let Proc be the CS-DP processor with $\text{Proc}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu) =$*

- $\{(\mathcal{P} - \mathcal{P}', \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, if \mathcal{E} is size-preserving, $\overline{\Delta}^2 = \overline{\Delta}^2(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$, $\Delta^2 = \Delta^2(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$, all of $\mathcal{R}(\Delta^2)$, $\mathcal{S}(\overline{\Delta}^2)$, and \mathcal{P} are strongly conservative, $\mathcal{P}' \subseteq \mathcal{P}$, and either
 - there exists an ordinary reduction pair (\succsim, \succ) such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \emptyset, \mathcal{R}(\Delta^2) \cup \mathcal{R}_{\Pi}, \mathcal{S}(\overline{\Delta}^2), \mathcal{E}(\overline{\Delta}^2))$, or
 - there exists a $Th_{\mathbb{N}}$ -reduction pair (\succsim, \succ) for \mathcal{P} such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \emptyset, \mathcal{R}(\Delta^2) \cup \mathcal{R}_{\Pi}, \mathcal{S}(\overline{\Delta}^2), \mathcal{E}(\overline{\Delta}^2))$, or
 - there exists a $Th_{\mathbb{Z}}$ -reduction pair (\succsim, \succ) for \mathcal{P} such that $(\succsim, \succ) \models (\mathcal{P}', \mathcal{P}, \emptyset, \mathcal{R}(\Delta^2) \cup \mathcal{R}_{\Pi, \text{univ}}, \mathcal{S}(\overline{\Delta}^2), \mathcal{E}(\overline{\Delta}^2))$.
- $\{(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)\}$, otherwise.

Then Proc is sound.

Example 12.27. The CS-DP processor of Theorem 12.26 makes it possible to finish the termination proof of the running example. Recall the following dependency pair from Example 12.12:

$$\text{take}^\sharp(x) \rightarrow \text{take}^\sharp(x - 1) \llbracket x > 0 \rrbracket \quad (12.3)$$

Then this dependency pair is strongly conservative, and the same holds true for $\mathcal{R}(\Delta^2) = \emptyset$ and $\mathcal{S}(\overline{\Delta}^2) = \mathcal{S}_{\text{base}}$. It is thus possible to apply the $\mathcal{Th}_{\mathbb{Z}}$ -polynomial interpretation with $c_{\text{pol}} = 0$ and $\mathcal{Pol}(\text{take}^\sharp) = x_1$. \triangle

12.6 Implementation

The techniques presented for CS-CERSs in this chapter have been implemented in the automated termination checker AProVE [84] for $\mathcal{Th}_{\mathbb{Z}}$. As for the non-context-sensitive case, most techniques can be implemented straightforwardly. The more challenging implementations of the estimated dependency graph EDG and the automated generation of $\mathcal{Th}_{\mathbb{Z}}$ -reduction pairs based on $\mathcal{Th}_{\mathbb{Z}}$ -polynomial interpretations is done as described in Chapter 9 for the non-context-sensitive case.

12.7 Summary

This chapter has introduced several sound CS-DP processors, all of which have been implemented in the automated termination checker AProVE. These CS-DP processors adapt DP processors introduced in Chapters 6–8 to the context-sensitive case.

While dependency graphs and the subterm criterion can be adapted to the context-sensitive case quite easily, an adaptation of the method based on function

Chapter 12. Context-Sensitive DP Processors

dependencies is more challenging. Following the recent achievement for ordinary context-sensitive TRSs [92, 1], an adaptation to CS-CERSs has been presented in this chapter. While the definition of function dependencies for CS-CERSs is more complex than the one for regular CERSs in general, the definition from Section 8.3 is essentially re-obtained for strongly conservative systems.

An implementation of the techniques presented in this chapter in the termination prover **AProVE** has been evaluated on a collection of examples (including several examples obtained from functional **Maude** modules containing **strat**-annotations). This evaluation shows that the techniques developed in this dissertation are very successful, cf. Chapter 15.

Chapter 13

Inductive Theorem Proving with CERSs

In the previous chapters, CERSs have been used as a tool for showing termination of algorithms that are modeled in the form of rewrite rules. Recall from Chapter 1 that reasoning about the partial correctness of such algorithms often require reasoning about the functions defined by a CERS.

While this kind of reasoning can be done using generic theorem proving methods, this approach is not very satisfactory since generic theorem proving is concerned with deriving properties that are valid in *all* models of a CERS (also non-standard ones).¹ For the purpose of reasoning about the functions defined by a CERS $(\mathcal{R}, \mathcal{S}, \mathcal{E})$, only one particular model is of interest, namely the *standard model* consisting of the $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ -equivalence classes of ground terms. It is important to distinguish between validity in all models of a CERS and validity in only this specific model of a CERS since these two notions do not coincide in general.

¹A model of a CERS is a many-sorted theory in which all rewrite rules and equations of the CERS are valid. Here, a many-sorted theory is the obvious generalization of a theory as given in Definition 3.1, see Definition 13.35 below.

Example 13.1. Consider the following ordinary TRS \mathcal{R} that defines a function to add two natural numbers given in a Peano representation:

$$\begin{aligned} x + \mathcal{O} &\rightarrow x \\ x + \mathfrak{s}(y) &\rightarrow \mathfrak{s}(x + y) \end{aligned}$$

The first rule specifies \mathcal{O} to be a right-neutral element, and it is a valid question whether it follows that \mathcal{O} is also left-neutral, i.e., if the conjecture $\mathcal{O} + y \equiv y$ follows from the rewrite rules. Using the generic theorem proving approach this is not true since there are models of the rewrite rules (considered as equations) that falsify $\mathcal{O} + y \equiv y$. For instance, consider the model $\mathcal{M} = (M, \mathcal{O}^{\mathcal{M}}, \mathfrak{s}^{\mathcal{M}}, +^{\mathcal{M}})$ with $M = \{a, b\}$ and

- $\mathcal{O}^{\mathcal{M}} = a$
- $\mathfrak{s}^{\mathcal{M}}(a) = a$ and $\mathfrak{s}^{\mathcal{M}}(b) = b$
- $a +^{\mathcal{M}} a = a$, $a +^{\mathcal{M}} b = a$, $b +^{\mathcal{M}} a = b$, and $b +^{\mathcal{M}} b = b$

Then all rewrite rules of the TRS are valid, but the conjecture $\mathcal{O} + y \equiv y$ is *not* valid since $\mathcal{O}^{\mathcal{M}} +^{\mathcal{M}} b = a +^{\mathcal{M}} b = a \neq b$. Notice that the model \mathcal{M} is non-standard in the sense that b is not the interpretation of any constructor ground term (i.e., ground term built using the constructors \mathcal{O} and \mathfrak{s}). Thus, this model is not relevant in the context of program verification, since all data values in programs are built using the constructors.

For program verification, it needs to be established whether $\mathcal{O} + y \equiv y$ is valid in the standard model, i.e., whether $\mathcal{O} + t \equiv t$ follows from \mathcal{R} for all ground terms t . Since \mathcal{R} is quasi-reductive (see Definition 13.14 below), each \mathcal{R} -equivalence class of ground terms contains a constructor ground term and it thus suffices to show that $\mathcal{O} + t \equiv t$ follows from \mathcal{R} for all constructor ground terms t . Due to the recursive nature of the definition of $+$, this can be shown using inductive reasoning.

Chapter 13. Inductive Theorem Proving with CERSs

- In the base case, the constructor ground term \mathcal{O} is considered. Then, $\mathcal{O} + \mathcal{O} \equiv \mathcal{O}$ easily follows from \mathcal{R} .
- In the step case, the constructor ground term has the form $s(t')$ for some constructor ground term t' and the inductive hypothesis states that $\mathcal{O} + t' \equiv t'$ follows from \mathcal{R} . But then $\mathcal{O} + s(t') \equiv s(t')$ also follows from \mathcal{R} since $\mathcal{O} + s(t') \equiv s(\mathcal{O} + t')$ follows from the second rule in \mathcal{R} and $s(\mathcal{O} + t') \equiv s(t')$ follows from the inductive hypothesis. \triangle

As demonstrated by this example, showing that the functions defined by a CERS satisfy certain properties usually requires inductive reasoning since the functions are most commonly defined using recursion.

In order to make inductive theorem proving with CERSs possible, this chapter introduces a restricted class of CERSs and shows how this class can effectively be used for proving properties of the functions defined by the CERS using inductive reasoning.

There are two commonly used paradigms for inductive theorem proving: *explicit induction* and *implicit induction*. In explicit induction (see, e.g., [35, 174, 112, 39, 40, 98, 113, 171]), a concrete induction scheme is computed for each conjecture, and the subsequent reasoning is based on this induction scheme. Here, an induction scheme explicitly gives the base cases and the step cases, where the step cases consists of an *obligation* and one or more *hypotheses*. This is the kind of reasoning employed in Example 13.1.

In implicit induction (see, e.g., [134, 97, 105, 102, 72, 107, 147, 33, 7, 157]), no concrete induction scheme is constructed a priori. Instead, an induction scheme is implicitly constructed during the proof attempt. Implicit induction is largely based on the term rewriting framework and is rooted in the Knuth-Bendix completion method [114]. While explicit induction is commonly considered to be more powerful

than implicit induction, implicit induction is more automatic in the sense that less (or no) user interaction is required.

The proof method presented in this chapter is based on the implicit induction paradigm and couples inductive reasoning with a decision procedure for the theory LIAC, which combines the linear theory of integers with the constructors of a CERS. The integration of a decision procedures for the linear theory of integers into inductive reasoning has been previously considered in [110, 11]. The proof method developed in this chapter is in general incomparable to these methods. On the one hand, the methods presented in [110, 11] are more complex and powerful. On the other hand, the use of $\mathcal{Th}_{\mathbb{Z}}$ -constraints in this dissertation gives rise to an elegant proof method ([110, 11] are based on ordinary rewriting without constraints). Inductive theorem proving for rewrite systems with certain kinds of constraints has been investigated in [34]. That method, however, does not support $\mathcal{Th}_{\mathbb{Z}}$ -constraints and is thus incomparable to the method presented below.

13.1 Preliminaries

The restricted class of CERSs that is used for inductive theorem proving is based on $\mathcal{Th}_{\mathbb{Z}}$, i.e., the linear theory of integers. Consequently, the sort `base` of the built-in theory will be denoted by `int`. Collection data structures are currently not allowed (with the exception of lists built using `nil` and `cons`), and it is furthermore assumed that the signature \mathcal{F} does not contain function symbols with resulting sort `int`. Terms of sort `int` are thus built using $\{0, 1, +, -\}$ and variables. Relaxing these restrictions is left for future work. Nonetheless, this restricted class of CERSs is already sufficient for presenting the new ideas for decidable induction developed in Chapter 14.

The left- and right-hand sides of rules in the restricted class of CERSs need

to satisfy the following requirement. As will become apparent later, this makes it possible to have a simple definition of the rewrite relation since it disallows pattern-matching with $+$ and $-$.

Definition 13.2 (\mathbb{Z} -Free). *A term t is \mathbb{Z} -free iff $\mathcal{F}(t) \cap \{+, -\} = \emptyset$.*

Now the class of \mathbb{Z} -CERSs is defined as follows. Notice that only free constructors are allowed, i.e., compact lists, sets, and multisets are not supported.

Definition 13.3 (\mathbb{Z} -CERS). *A CERS $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is a \mathbb{Z} -CERS iff*

1. $\mathcal{S} = \mathcal{S}_{Th_{\mathbb{Z}}}$
2. $\mathcal{E} = \mathcal{E}_{Th_{\mathbb{Z}}}$
3. For all $l \rightarrow r \llbracket \varphi \rrbracket \in \mathcal{R}$, both l and r are \mathbb{Z} -free.

A \mathbb{Z} -CERS $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is also identified with \mathcal{R} .

Example 13.4. For two lists built using `nil` and `cons`, `prefix(xs, ys)` computes the longest prefix p of xs such that all elements of p occur in ys in the same order as in p (but not necessarily consecutively).

$$\begin{aligned}
 \text{prefix}(\text{nil}, ys) &\rightarrow \text{nil} \\
 \text{prefix}(\text{cons}(x, xs), \text{nil}) &\rightarrow \text{nil} \\
 \text{prefix}(\text{cons}(x, xs), \text{cons}(y, ys)) &\rightarrow \text{cons}(x, \text{prefix}(xs, ys)) \llbracket x \simeq y \rrbracket \\
 \text{prefix}(\text{cons}(x, xs), \text{cons}(y, ys)) &\rightarrow \text{prefix}(\text{cons}(x, xs), ys) \llbracket x \not\simeq y \rrbracket
 \end{aligned}$$

This CERS is a \mathbb{Z} -CERS. △

The restriction that left- and right-hand sides of the rewrite rules in a \mathbb{Z} -CERS are \mathbb{Z} -free makes it possible to have a simple definition of the rewrite relation that does not take the sets $\mathcal{S}_{Th_{\mathbb{Z}}}$ and $\mathcal{E}_{Th_{\mathbb{Z}}}$ into account, while still representing all of $\xrightarrow{\mathcal{S}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}}$. This goal is achieved by restricting attention to particular representatives

of terms that are equivalent up to $\mathcal{S}_{Th_{\mathbb{Z}}}$ and $\mathcal{E}_{Th_{\mathbb{Z}}}$. These representatives are irreducible by $\rightarrow_{\mathcal{E}_{Th_{\mathbb{Z}}}\setminus\mathcal{S}_{Th_{\mathbb{Z}}}}$ and satisfy certain conditions on nested occurrences of the function symbol $+$.

Definition 13.5 (\mathbb{Z} -Normal Terms and Substitutions). *Let $>_{\mathcal{T}}$ be a fixed total well-founded order on $\mathcal{V} \cup \{0, 1\}$. A term t is \mathbb{Z} -normal w.r.t. $>_{\mathcal{T}}$ iff the following conditions are satisfied:*

1. t is irreducible by $\rightarrow_{\mathcal{E}_{Th_{\mathbb{Z}}}\setminus\mathcal{S}_{Th_{\mathbb{Z}}}}$.
2. Whenever t contains a subterm of the form $t_1 + t_2$, then $\text{root}(t_1) \neq +$.
3. Whenever t contains a subterm of the form $t_1 + (t_2 + \dots + (t_{n-1} + t_n) \dots)$ with $t_1, \dots, t_n \in \mathcal{V} \cup \{0, 1\}$, then $t_1 >_{\mathcal{T}} t_2 >_{\mathcal{T}} \dots >_{\mathcal{T}} t_{n-1} >_{\mathcal{T}} t_n$.

A substitution σ is \mathbb{Z} -normal iff $\sigma(x)$ is \mathbb{Z} -normal for all variables x .

Example 13.6. Let $x >_{\mathcal{T}} y >_{\mathcal{T}} z >_{\mathcal{T}} 1 >_{\mathcal{T}} 0$. Then $f(x + y, y + (z + 1))$ is \mathbb{Z} -normal. $f(x + 0, 0)$ is not \mathbb{Z} -normal since $x + 0$ is reducible by $\rightarrow_{\mathcal{E}_{Th_{\mathbb{Z}}}\setminus\mathcal{S}_{Th_{\mathbb{Z}}}}$. $f(0, 1 + x)$ is not \mathbb{Z} -normal since $1 \not>_{\mathcal{T}} x$. \triangle

Notice that for ground terms, the conditions in Definition 13.5 imply that nested occurrences of the function symbol $+$ have the form $1 + (1 + \dots + (1 + 1) \dots)$ or $-1 + (-1 + \dots + (-1 + -1) \dots)$.

The actual choice of the well-founded order $>_{\mathcal{T}}$ used in the definition of \mathbb{Z} -normal terms is not important since it will not be used explicitly. Furthermore, the definition of \mathbb{Z} -normal terms can be replaced by a different definition as long as the statements of Lemma 13.7 and Lemma 13.8 below are satisfied.

The following lemma relates the notions of \mathbb{Z} -free terms and \mathbb{Z} -normal terms. In particular, every \mathbb{Z} -free term is also \mathbb{Z} -normal.

Lemma 13.7. *Let t be a \mathbb{Z} -free term.*

Chapter 13. Inductive Theorem Proving with CERSs

1. t is \mathbb{Z} -normal.
2. For any substitution σ , the term $t\sigma$ is \mathbb{Z} -normal iff σ is \mathbb{Z} -normal.

Next, it is possible to show that every term is equivalent up to $\mathcal{S}_{Th_{\mathbb{Z}}}$ and $\mathcal{E}_{Th_{\mathbb{Z}}}$ to a term that is \mathbb{Z} -normal. Therefore, the set of \mathbb{Z} -normal terms can indeed serve as a set of representatives for all terms.

Lemma and Definition 13.8. *For any term t , a \mathbb{Z} -normal term $\text{norm}(t)$ can be computed such that $t \leftrightarrow_{\mathcal{E}_{Th_{\mathbb{Z}}} \cup \mathcal{S}_{Th_{\mathbb{Z}}}}^* \text{norm}(t)$. Furthermore, $s \leftrightarrow_{\mathcal{E}_{Th_{\mathbb{Z}}} \cup \mathcal{S}_{Th_{\mathbb{Z}}}}^* t$ implies $\text{norm}(s) = \text{norm}(t)$ for all terms s, t . For any substitution σ , the substitution $\text{norm}(\sigma)$ is given by $\text{norm}(\sigma)(x) = \text{norm}(\sigma(x))$.*

Example 13.9. If $x >_{\mathcal{T}} y >_{\mathcal{T}} 1$, then $\text{norm}(\mathbf{f}(\mathbf{c}(x), (\mathbf{1} + y) + (x + \mathbf{0}))) = \mathbf{f}(\mathbf{c}(x), x + (y + \mathbf{1}))$. △

The rewrite relation of a \mathbb{Z} -CERS is now restricted to operate on \mathbb{Z} -normal terms. Notice that the only difference to Definition 3.15 is in condition 1, where $s|_p = l\sigma$ is used instead of the more complex $s|_p \xrightarrow{>\Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \circ \overset{>\Lambda}{\sim}_{\mathcal{E}} l\sigma$.

Definition 13.10 (Rewrite Relation of a \mathbb{Z} -CERS). *Let \mathcal{R} be a \mathbb{Z} -CERS and let s be a \mathbb{Z} -normal term. Then $s \rightarrow_{\mathcal{R}, \mathbb{Z}} t$ iff there exist a constrained rewrite rule $l \rightarrow r[\varphi] \in \mathcal{R}$, a position $p \in \text{Pos}(s)$, and a substitution σ such that*

1. $s|_p = l\sigma$,
2. $\varphi\sigma$ is $Th_{\mathbb{Z}}$ -valid, and
3. $t = s[r\sigma]_p$.

Notice that all (sort-correct) substitutions are $Th_{\mathbb{Z}}$ -based since \mathcal{F} does not contain function symbols with resulting sort `int`. Also, notice that the substitution σ in Definition 13.10 is \mathbb{Z} -normal since s is \mathbb{Z} -normal.

For the rewrite relation defined this way to be meaningful, it has to be ensured that rewriting a \mathbb{Z} -normal term again results in a \mathbb{Z} -normal term. This easily follows from the assumption that left- and right-hand sides of the rules in a \mathbb{Z} -CERS are \mathbb{Z} -free. Furthermore, it becomes possible to relate $\rightarrow_{\mathcal{R},\mathbb{Z}}$ to $\xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}}$. In particular, for any \mathbb{Z} -CERS, $\rightarrow_{\mathcal{R},\mathbb{Z}}$ is terminating if $\xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}}$ is terminating. Thus, the methods developed for proving termination of general CERSs in this dissertation can also be used for showing termination of \mathbb{Z} -CERSs.

Lemma 13.11. *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a \mathbb{Z} -CERS, let s be a \mathbb{Z} -normal term, and let t be a term such that $s \rightarrow_{\mathcal{R},\mathbb{Z}} t$.*

1. t is \mathbb{Z} -normal.
2. If s is furthermore \mathbb{Z} -free, then t is \mathbb{Z} -free as well.
3. $s \xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}} t$. Thus, $\rightarrow_{\mathcal{R},\mathbb{Z}}$ is terminating on \mathbb{Z} -normal terms if $\xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}}$ is terminating.

For the converse of Lemma 13.11.3, the following result can be obtained. In particular, termination of $\rightarrow_{\mathcal{R},\mathbb{Z}}$ on \mathbb{Z} -normal terms is thus equivalent to termination of $\xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}}$.

Lemma 13.12. *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a \mathbb{Z} -CERS and let s, t be terms. If $s \xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}} t$, then $\text{norm}(s) \rightarrow_{\mathcal{R},\mathbb{Z}} \text{norm}(t)$. Thus, $\xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}}$ is terminating if $\rightarrow_{\mathcal{R},\mathbb{Z}}$ is terminating on \mathbb{Z} -normal terms.*

13.2 Quasi-Reductivity and Confluence

In order to use \mathbb{Z} -CERSs in the context of inductive theorem proving, it becomes necessary to impose certain (semantical) restrictions. Similar conditions need to be imposed for inductive theorem proving with ordinary TRSs, and this section extends

these properties to \mathbb{Z} -CERSs. Additionally, it is discussed how these properties can be ensured for a class of \mathbb{Z} -CERSs that is sufficient for most practical purposes. First, the following definitions are needed.

Definition 13.13 (Constructor Ground Terms and Substitutions). *A ground term t is a constructor ground term if $t \in \mathcal{T}(\mathcal{C}(\mathcal{R}) \cup \mathcal{F}_{Th_{\mathbb{Z}}})$. A ground substitution σ is a constructor ground substitution if $\sigma(x) \in \mathcal{T}(\mathcal{C}(\mathcal{R}) \cup \mathcal{F}_{Th_{\mathbb{Z}}})$ for all variables x .*

In the following, it is assumed that each sort of a \mathbb{Z} -CERS has at least two distinct constructor ground terms.²

The first property of a \mathbb{Z} -CERS \mathcal{R} needed for inductive theorem proving is that the defined functions in $\mathcal{D}(\mathcal{R})$ are total, i.e., result in a constructor ground term when applied to constructor ground terms. More precisely, it suffices to consider \mathbb{Z} -normal constructor ground terms since $\rightarrow_{\mathcal{R},\mathbb{Z}}$ is only defined on \mathbb{Z} -normal terms.

Definition 13.14 (Quasi-Reductivity). *A \mathbb{Z} -CERS \mathcal{R} is quasi-reductive iff every \mathbb{Z} -normal ground term of the form $f(t_1, \dots, t_n)$ with $f \in \mathcal{D}(\mathcal{R})$ and \mathbb{Z} -normal constructor ground terms t_1, \dots, t_n is reducible by $\rightarrow_{\mathcal{R},\mathbb{Z}}$.*

Example 13.15. The \mathbb{Z} -CERS from Example 13.4 is quasi-reductive. If the last rule is omitted, then the resulting \mathbb{Z} -CERS is not quasi-reductive since the \mathbb{Z} -normal ground term $\text{prefix}(\text{cons}(0, \text{nil}), \text{cons}(1, \text{nil}))$ is not reducible by $\rightarrow_{\mathcal{R},\mathbb{Z}}$. △

For ordinary TRSs, quasi-reductivity is equivalent to *sufficient completeness* [93] under suitable assumption (see [106], where quasi-reductivity is called *quasi-reducibility w.r.t. constructors*).

The second property imposed on a \mathbb{Z} -CERS is *confluence*, i.e., \mathcal{E} -confluence for $\mathcal{E} = \emptyset$. This property will only be needed in order to disprove false conjectures. For the sake of completeness, the definition of confluence is as follows.

²This restriction is not severe in practice and excludes conjectures such as $x \equiv y$ for distinct variables x and y from being true.

Definition 13.16 (Confluence). *A \mathbb{Z} -CERS \mathcal{R} is confluent iff $\leftarrow_{\mathcal{R},\mathbb{Z}}^* \circ \rightarrow_{\mathcal{R},\mathbb{Z}}^* \subseteq \rightarrow_{\mathcal{R},\mathbb{Z}}^* \circ \leftarrow_{\mathcal{R},\mathbb{Z}}^*$.*

Confluence implies the uniqueness of normal forms, whereas quasi-reductivity implies that each ground term can be reduced to a constructor ground term (if the \mathbb{Z} -CERS is terminating). Thus, if both properties are satisfied, then each ground term has a unique constructor ground term as its normal form.

Checking whether a \mathbb{Z} -CERS is quasi-reductive and confluent seems to be a hard problem in general. Thus, a restricted class of \mathbb{Z} -CERSs is considered in the following. For this class, checking for quasi-reductivity is easily possible. Furthermore, \mathbb{Z} -CERSs from this class are always confluent.

First, it is required that the left-hand sides of rules are linear and constructor-based. This is important in order to check for quasi-reductivity. In order to ensure confluence, it is required that the rules are “disjoint” in the sense that at most one rule is applicable to each position in any term. Notice that two rules might have identical left-hand sides as long as the conjunction of the $\mathcal{Th}_{\mathbb{Z}}$ -constraints of these rules is not satisfiable.

Notational Convention 13.17. *For any \mathbb{Z} -CERS \mathcal{R} , $\widehat{\mathcal{C}}(\mathcal{R}) := \mathcal{C}(\mathcal{R}) \cup \{0, 1\}$*

Definition 13.18 (Normal \mathbb{Z} -CERS). *A \mathbb{Z} -CERS \mathcal{R} is normal iff*

1. *For all $l \rightarrow r[[\varphi]] \in \mathcal{R}$, the term l is linear and has the form $f(l_1, \dots, l_n)$ with $l_1, \dots, l_n \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$.*
2. *For any two rules $l_1 \rightarrow r_1[[\varphi_1]], l_2 \rightarrow r_2[[\varphi_2]]$, either $l_1 = l_2^3$ or l_1, l_2 are not unifiable after their variables have been renamed apart.*
3. *For any two non-identical rules $l_1 \rightarrow r_1[[\varphi_1]], l_2 \rightarrow r_2[[\varphi_2]]$ with $l_1 = l_2$, the constraint $\varphi_1 \wedge \varphi_2$ is $\mathcal{Th}_{\mathbb{Z}}$ -unsatisfiable.*

³This condition could be relaxed by identifying terms that are identical up to a variable-renaming.

4. Whenever $l_1 \rightarrow r_1[\varphi_1], \dots, l_n \rightarrow r_n[\varphi_n]$ are all rules with identical left-hand sides, then the constraint $\varphi_1 \vee \dots \vee \varphi_n$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid.

Example 13.19. The \mathbb{Z} -CERS from Example 13.4 is a normal \mathbb{Z} -CERS. \triangle

Using conditions 1, 2, and 4 in Definition 13.18, the following decidability result can be obtained by reducing quasi-reductivity of normal \mathbb{Z} -CERSs to quasi-reductivity of ordinary left-linear constructor-based TRSs. Quasi-reductivity of ordinary TRSs satisfying these conditions is well-known to be decidable (see, e.g., [78, 94]).

Theorem 13.20. *It is decidable whether a normal \mathbb{Z} -CERS is quasi-reductive.*

Next, it can be shown that normal \mathbb{Z} -CERSs are always confluent, regardless of whether they are terminating or not. This result follows from conditions 1, 2, and 3 in Definition 13.18 since these conditions imply that normal \mathbb{Z} -CERSs are a suitable generalization of orthogonal ordinary TRSs (which are also known to be confluent, regardless of whether they are terminating or not [149]).

Theorem 13.21. *Every normal \mathbb{Z} -CERS is confluent.*

13.3 Inductive Theorem Proving

The atomic conjectures in inductive theorem proving are equalities between terms. In this dissertation, a generalized form of these atomic conjectures is used that also incorporates a $\mathcal{Th}_{\mathbb{Z}}$ -constraint.

Definition 13.22 (Atomic Conjectures). *An atomic conjecture has the form $s \equiv t[\varphi]$ for \mathbb{Z} -free terms s, t and a $\mathcal{Th}_{\mathbb{Z}}$ -constraint φ such that $\text{sort}(s) = \text{sort}(t)$ and $\text{sort}(s) \neq \text{int}$. As usual, a constraint of the form \top will be omitted.*

Example 13.23. For the \mathbb{Z} -CERS from Example 13.4, $\text{prefix}(xs, xs) \equiv xs$ is an atomic conjecture. \triangle

Notice that atomic conjectures satisfy the same requirements that are imposed on the rewrite rules in a \mathbb{Z} -CERS (except for the variable condition $\mathcal{V}(t) \cup \mathcal{V}(\varphi) \subseteq \mathcal{V}(s)$). Intuitively, an atomic conjecture $s \equiv t[\varphi]$ is true whenever $s\sigma$ and $t\sigma$ are “equal” up to the rules of a \mathbb{Z} -CERS for all \mathbb{Z} -normal constructor ground substitutions σ that make φ true. This is of course equivalent to showing that the implication $\varphi\sigma \Rightarrow s\sigma \equiv t\sigma$ is true in the context of the \mathbb{Z} -CERS for all \mathbb{Z} -normal constructor ground substitutions σ .

Definition 13.24 (Inductive Theorems). *An atomic conjecture $s \equiv t[\varphi]$ is an inductive theorem of a \mathbb{Z} -CERS \mathcal{R} iff $s\sigma \leftrightarrow_{\mathcal{R}, \mathbb{Z}}^* t\sigma$ for all \mathbb{Z} -normal constructor ground substitutions σ such that $\varphi\sigma$ is $\text{Th}_{\mathbb{Z}}$ -valid. A set of atomic conjectures in an inductive theorem iff all of its elements are inductive theorems.*

Example 13.25. It will be shown below that the atomic conjecture $\text{prefix}(xs, xs) \equiv xs$ from Example 13.23 is an inductive theorem. \triangle

There are other possible definitions of when an atomic conjecture is an inductive theorem. The most general (sensible) definition of an inductive theorem would be that $s\sigma \leftrightarrow_{\mathcal{R} \cup \mathcal{E}_{\text{Th}_{\mathbb{Z}}} \cup \mathcal{S}_{\text{Th}_{\mathbb{Z}}, \mathbb{Z}}}^* t\sigma$ for all ground substitutions σ such that $\varphi\sigma$ is $\text{Th}_{\mathbb{Z}}$ -valid (not only for all \mathbb{Z} -normal constructor ground substitutions σ). Due to the shape of \mathbb{Z} -CERSs and atomic conjectures, however, this is already implied by the condition from Definition 13.24 if the \mathbb{Z} -CERS is quasi-reductive.

Lemma 13.26. *If $s \equiv t[\varphi]$ is an inductive theorem of a quasi-reductive \mathbb{Z} -CERS \mathcal{R} , then $s\sigma \leftrightarrow_{\mathcal{R} \cup \mathcal{E}_{\text{Th}_{\mathbb{Z}}} \cup \mathcal{S}_{\text{Th}_{\mathbb{Z}}, \mathbb{Z}}}^* t\sigma$ for all ground substitutions σ such that $\varphi\sigma$ is $\text{Th}_{\mathbb{Z}}$ -valid.*

The inductive theorem proving method for \mathbb{Z} -CERSs developed in this dissertation is based on Reddy’s *term rewriting induction* [147]. The presentation follows

[7, 9]. The main idea of this method is to *expand* certain subterms of an atomic conjecture using narrowing with the rewrite rules of a \mathbb{Z} -CERS.

Definition 13.27 (Basic Terms). *A \mathbb{Z} -free term t is basic iff $t = f(t_1, \dots, t_n)$ where $f \in \mathcal{D}(\mathcal{R})$ and $t_1, \dots, t_n \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$.*

Example 13.28. In Example 13.4, the term $\text{prefix}(xs, xs)$ is basic. \triangle

Expansion of a basic subterm is now done as follows. Notice that the constraints of the atomic conjecture and the rewrite rule are combined and instantiated and that it is checked whether the resulting constraint is still satisfiable. Furthermore, notice that the substitutions used for narrowing are computed using syntactic unification and *not* $\mathcal{Th}_{\mathbb{Z}}$ -unification. This is in analogy to the definition of the rewrite relation of a \mathbb{Z} -CERS which is based on syntactic matching and not on $\mathcal{Th}_{\mathbb{Z}}$ -matching.

Definition 13.29 (Expd). *For an atomic conjecture $s \equiv t[\varphi]$, a basic term u such that $s = C[u]$, and a \mathbb{Z} -CERS \mathcal{R} , the set $\text{Expd}_u(s, t, \varphi)$ is defined as*

$$\text{Expd}_u(s, t, \varphi) = \left\{ C[r]\sigma \equiv t\sigma[\varphi\sigma \wedge \psi\sigma] \mid l \rightarrow r[\psi] \in \mathcal{R}, \sigma = \text{mgu}(u, l), \text{ and } \varphi\sigma \wedge \psi\sigma \text{ is } \mathcal{Th}_{\mathbb{Z}}\text{-satisfiable} \right\}$$

Here, it has been assumed that the variables of $l \rightarrow r[\psi]$ have been renamed to be disjoint from the variables of $s \equiv t[\varphi]$.

Example 13.30. Consider the \mathbb{Z} -CERS from Example 13.4 and the atomic conjecture $\text{prefix}(xs, xs) \equiv xs$ from Example 13.23.

1. For the first rule, $\text{mgu}(\text{prefix}(xs, xs), \text{prefix}(\text{nil}, ys')) = \{xs \mapsto \text{nil}, ys' \mapsto \text{nil}\}$.
2. For the second rule, the terms $\text{prefix}(xs, xs)$ and $\text{prefix}(\text{cons}(x', xs'), \text{nil})$ are not unifiable.
3. For the third rule, $\text{mgu}(\text{prefix}(xs, xs), \text{prefix}(\text{cons}(x', xs'), \text{cons}(y', ys')))) = \{xs \mapsto \text{cons}(x', xs'), y' \mapsto x', ys' \mapsto xs'\}$ and the constraint $x' \simeq x'$ obtained after instantiation with the most general unifier is $\mathcal{Th}_{\mathbb{Z}}$ -satisfiable.

4. For the fourth rule, $\text{mgu}(\text{prefix}(xs, xs), \text{prefix}(\text{cons}(x', xs'), \text{cons}(y', ys')))) = \{xs \mapsto \text{cons}(x', xs'), y' \mapsto x', ys' \mapsto xs'\}$ and the constraint $x' \neq x'$ obtained after instantiation with the most general unifier is *not* $\text{Th}_{\mathbb{Z}}$ -satisfiable.

Therefore, $\text{Expd}_u(s, t, \varphi)_{\text{prefix}(xs, xs)}(\text{prefix}(xs, xs), xs, \top)$ consists of the two atomic conjectures $\text{nil} \equiv \text{nil}$ and $\text{cons}(x', \text{prefix}(xs', xs')) \equiv \text{cons}(x', xs')[[x' \simeq x']]$. \triangle

It is easy to see that expanding an atomic conjecture using this definition produces atomic conjectures. This is important since it makes it possible to apply further expansions to these newly obtained atomic conjectures as well.

Lemma 13.31. *Let $s \equiv t[[\varphi]]$ be an atomic conjecture. Then $\text{Expd}_u(s, t, \varphi)$ consists of atomic conjectures as well.*

From now on it is assumed that \mathcal{R} is a quasi-reductive and terminating \mathbb{Z} -CERS. In this case, the following technical result relates the atomic conjectures in $\text{Expd}_u(s, t, \varphi)$ to the atomic conjecture $s \equiv t[[\varphi]]$ and the rules in \mathcal{R} . It is needed for the soundness proof of the inductive proof method (cf. Theorem 13.40 below).

Lemma 13.32. *Let $s \equiv t[[\varphi]]$ be an atomic constraint and let u be a basic term such that $s = C[u]$.*

1. $s\sigma \rightarrow_{\mathcal{R}, \mathbb{Z}} \circ \leftrightarrow_{\text{Expd}_u(s, t, \varphi), \mathbb{Z}} t\sigma$ for any \mathbb{Z} -normal constructor ground substitution σ such that $\varphi\sigma$ is $\text{Th}_{\mathbb{Z}}$ -valid.
2. If $v \leftrightarrow_{\text{Expd}_u(s, t, \varphi), \mathbb{Z}} w$, then $v \leftrightarrow_{\mathcal{R} \cup \{s \rightarrow t[[\varphi]]\}, \mathbb{Z}}^* w$.

The inductive proof method for \mathbb{Z} -CERSs is given in Figure 13.1. Here, the notation $s \doteq t[[\varphi]]$ is used to stand for one of $s \equiv t[[\varphi]]$ and $t \equiv s[[\varphi]]$. The inference rules operate on tuples $\langle E, H \rangle$, where E consists of atomic conjectures that are to be proven and H consists of atomic conjectures that have been oriented as rewrite rules. These rules constitute the *hypotheses* in a proof by induction. The goal of an

| | | |
|---------------------|--|--|
| Expand | $\frac{\langle E \uplus \{s \doteq t[\varphi]\}, H \rangle}{\langle E \cup \text{Expd}_u(s, t, \varphi), H \cup \{s \rightarrow t[\varphi]\} \rangle}$ | if $\mathcal{V}(t) \cup \mathcal{V}(\varphi) \subseteq \mathcal{V}(s)$ and $\mathcal{R} \cup H \cup \{s \rightarrow t[\varphi]\}$ is terminating |
| Simplify | $\frac{\langle E \uplus \{s \doteq t[\varphi]\}, H \rangle}{\langle E \cup \{s' \doteq t[\varphi]\}, H \rangle}$ | if $s[\varphi] \rightarrow_{\mathcal{R} \cup H, \mathbb{Z}} s'[\varphi]$ |
| Delete | $\frac{\langle E \uplus \{s \doteq s[\varphi]\}, H \rangle}{\langle E, H \rangle}$ | |
| Theory ₁ | $\frac{\langle E \uplus \{s \doteq t[\varphi]\}, H \rangle}{\langle E, H \rangle}$ | if s, t do not contain symbols from $\mathcal{D}(\mathcal{R})$ and $\varphi \Rightarrow s \simeq t$ is LIAC-valid |
| Theory ₂ | $\frac{\langle E \uplus \{s \doteq t[\varphi]\}, H \rangle}{\perp}$ | if s, t do not contain symbols from $\mathcal{D}(\mathcal{R})$ and $\varphi \Rightarrow s \simeq t$ is not LIAC-valid |

 Figure 13.1: The inference system \mathcal{I} .

inductive proof attempt is to obtain a tuple of the form $\langle \emptyset, H \rangle$ starting from the tuple $\langle E, \emptyset \rangle$. As shown below, this implies that all atomic conjectures in E are inductive theorems. If none of the inference rules is applicable to $\langle E, H \rangle$ where $E \neq \emptyset$, then the inductive proof attempt fails. Finally, an inductive proof attempt may also diverge (i.e., not terminate) or end in \perp . As shown in Theorem 13.42, the later constitutes a disproof of (at least) one of the initial atomic conjectures.

The inference rule **Expand** uses Definition 13.29 to expand a basic subterm of an atomic conjecture. Then, this atomic conjecture is oriented as a rewrite rule and added to the set H of hypotheses. Notice that this addition is only allowed if the \mathbb{Z} -CERS consisting of $\mathcal{R} \cup H$ and this newly obtained rule is terminating. This restriction is needed in order to obtain a sound inductive proof method.⁴

The rule **Simplify** uses simplification with \mathcal{R} and the hypotheses in H . For this,

⁴Using the recent approach for proving non-orientable equations in term rewriting induction presented in [7, 8] it might be possible to relax this requirement.

the constraint of the atomic conjecture that is to be simplified is taken into account by considering the following rewrite relation. It only differs from Definition 13.10 in condition 2, which now requires that the instantiated constraint of the rewrite rule is valid under the assumption of the constraint that is attached to the atomic conjecture that is getting simplified.

Definition 13.33 (Rewrite Relation of a \mathbb{Z} -CERS on Constrained Terms). *Let \mathcal{R} be a \mathbb{Z} -CERS, let s be a \mathbb{Z} -normal term, and let ψ be a $\mathit{Th}_{\mathbb{Z}}$ -constraint. Then $s[\psi] \rightarrow_{\mathcal{R},\mathbb{Z}} t[\psi]$ iff there exist a constrained rewrite rule $l \rightarrow r[\varphi] \in \mathcal{R}$, a position $p \in \mathit{Pos}(s)$, and a substitution σ such that*

1. $s|_p = l\sigma$,
2. $\psi \Rightarrow \varphi\sigma$ is $\mathit{Th}_{\mathbb{Z}}$ -valid, and
3. $t = s[r\sigma]_p$.

Example 13.34. Given the rewrite rule $f(x) \rightarrow g(x) \llbracket x \geq 0 \rrbracket$ and the constrained term $f(y) \llbracket y > 0 \rrbracket$, Definition 13.33 gives $f(y) \llbracket y > 0 \rrbracket \rightarrow_{\mathcal{R},\mathbb{Z}} g(y) \llbracket y > 0 \rrbracket$ since, for $\sigma = \{x \mapsto y\}$, $f(y) = f(x)\sigma$ and the constraint $y > 0 \Rightarrow (x \geq 0)\sigma$, i.e., $y > 0 \Rightarrow y \geq 0$, is $\mathit{Th}_{\mathbb{Z}}$ -valid. Notice that $f(y) \not\rightarrow_{\mathcal{R},\mathbb{Z}} g(y)$ since $y \geq 0$ is not $\mathit{Th}_{\mathbb{Z}}$ -valid. △

The rule **Delete** removes trivial atomic conjectures, and the rules **Theory₁** and **Theory₂** can be applied to atomic conjectures that do not contain any defined function symbols. These rules make use of a decision procedure for the theory **LIAC** that combines the linear theory of integers with the (free) constructor symbols from $\mathcal{C}(\mathcal{R})$. For this, Definition 3.1 is slightly extended to a many-sorted theory.

Definition 13.35 (LIAC). *For a \mathbb{Z} -CERS \mathcal{R} , the theory **LIAC** has the form $\mathit{LIAC} = (\mathcal{F}_{\mathit{LIAC}}, \mathit{P}_{\mathit{LIAC}}, \mathcal{M}_{\mathit{LIAC}})$ where*

1. $\mathcal{F}_{\mathit{LIAC}} = \{0, 1, +, -\} \cup \mathcal{C}(\mathcal{R})$

2. $P_{\text{LIAC}} = \{\simeq, \geq, >\}^5$
3. $\mathcal{M}_{\text{LIAC}} = (M, (f^{\text{LIAC}})_{f \in \mathcal{F}_{\text{LIAC}}}, (P^{\text{LIAC}})_{P \in P_{\text{LIAC}}})$ where $M = \mathcal{T}(\mathcal{C}(\mathcal{R}) \cup \mathbb{Z})$, the function symbols in $\{0, 1, +, -\}$ and predicate symbols in P_{LIAC} are interpreted in the obvious way, and $f^{\text{LIAC}}(t_1, \dots, t_n) = f(t_1, \dots, t_n)$ for $f \in \mathcal{C}(\mathcal{R})$.

LIAC-validity and LIAC-satisfiability are decidable and decision procedures have been implemented, for instance in the SMT-solver CVC3 [24].

Example 13.36. Continuing Example 13.4, consider the (true) atomic conjecture $\text{prefix}(xs, xs) \equiv xs$. The following derivation is a proof of this conjecture using the inference system \mathcal{I} :

$$\begin{array}{c}
 \frac{\langle \{\text{prefix}(xs, xs) \equiv xs\}, \emptyset \rangle}{\langle \{\text{nil} \equiv \text{nil}, \text{cons}(x, \text{prefix}(xs, xs)) \equiv \text{cons}(x, xs)[[x \simeq x]]\}, \{\text{prefix}(xs, xs) \rightarrow xs\} \rangle} \text{Expand} \\
 \frac{\langle \{\text{prefix}(xs, xs) \rightarrow xs\} \rangle}{\langle \{\text{cons}(x, \text{prefix}(xs, xs)) \equiv \text{cons}(x, xs)[[x \simeq x]]\}, \{\text{prefix}(xs, xs) \rightarrow xs\} \rangle} \text{Delete} \\
 \frac{\langle \{\text{cons}(x, \text{prefix}(xs, xs)) \equiv \text{cons}(x, xs)[[x \simeq x]]\}, \{\text{prefix}(xs, xs) \rightarrow xs\} \rangle}{\langle \{\text{cons}(x, xs) \equiv \text{cons}(x, xs)[[x \simeq x]]\}, \{\text{prefix}(xs, xs) \rightarrow xs\} \rangle} \text{Simplify} \\
 \frac{\langle \{\text{cons}(x, xs) \equiv \text{cons}(x, xs)[[x \simeq x]]\}, \{\text{prefix}(xs, xs) \rightarrow xs\} \rangle}{\langle \emptyset, \{\text{prefix}(xs, xs) \rightarrow xs\} \rangle} \text{Delete}
 \end{array}$$

As shown below, this implies that $\text{prefix}(xs, xs) \equiv xs$ is an inductive theorem. \triangle

The notation $\langle E, H \rangle \vdash_{\mathcal{I}} \langle E', H' \rangle$ is used to denote that the tuple $\langle E', H' \rangle$ has been obtained from $\langle E, H \rangle$ by one of the inference rules in Figure 13.1. As usual, $\vdash_{\mathcal{I}}^*$ denotes the reflexive-transitive closure of $\vdash_{\mathcal{I}}$.

Next, several properties of the inference system \mathcal{I} are shown. First, application of any inference rule except **Theory**₂ leaves the convertibility relation of $\mathcal{R} \cup E \cup H$ on \mathbb{Z} -normal ground terms unchanged. In particular, if $\langle E, \emptyset \rangle \vdash_{\mathcal{I}}^* \langle \emptyset, H \rangle$, then $\leftrightarrow_{\mathcal{R} \cup E, \mathbb{Z}}^* = \leftrightarrow_{\mathcal{R} \cup H, \mathbb{Z}}^*$ on \mathbb{Z} -normal ground terms.

⁵Strictly speaking, there is one predicate symbol \simeq_s for each sort s . To simplify notation, these predicate symbols have been identified. Also, the predicate symbols \geq and $>$ take two arguments of sort `int`.

Lemma 13.37. *If $\langle E_n, H_n \rangle \vdash_{\mathcal{I}} \langle E_{n+1}, H_{n+1} \rangle$ using an inference rule other than Theory_2 , then $\leftrightarrow_{\mathcal{R} \cup E_n \cup H_n, \mathbb{Z}}^* = \leftrightarrow_{\mathcal{R} \cup E_{n+1} \cup H_{n+1}, \mathbb{Z}}^*$ on \mathbb{Z} -normal ground terms.*

The soundness proof of the inference system \mathcal{I} is based on the following principle of Koike and Toyama [115] as reported in [7, 9] (where $\rightarrow_1 = \rightarrow_{\mathcal{R}, \mathbb{Z}}$, $\rightarrow_2 = \rightarrow_{H, \mathbb{Z}}$, and A is the set of \mathbb{Z} -normal ground terms):

Let $\rightarrow_1, \rightarrow_2$ be binary relations on a set A and let $\rightarrow_{1 \cup 2} = \rightarrow_1 \cup \rightarrow_2$.

Assume that

1. $\rightarrow_{1 \cup 2}$ is well-founded.
2. $\rightarrow_2 \subseteq \rightarrow_1 \circ \rightarrow_{1 \cup 2}^* \circ \leftarrow_{1 \cup 2}^*$

Then $\leftrightarrow_1^* = \leftrightarrow_{1 \cup 2}^*$.

In order to show condition 2 of this principle, the next lemma first shows that if $\langle E, \emptyset \rangle \vdash_{\mathcal{I}}^* \langle \emptyset, H \rangle$, then each application of an atomic conjecture from E can be simulated by a “valley proof” using \mathcal{R} and H .

Lemma 13.38. *If $\langle E_n, H_n \rangle \vdash_{\mathcal{I}}^* \langle \emptyset, H \rangle$ using inference rules other than Theory_2 , then $\leftrightarrow_{E_n, \mathbb{Z}} \subseteq \rightarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* \circ \leftarrow_{\mathcal{R} \cup H, \mathbb{Z}}^*$ on \mathbb{Z} -normal ground terms.*

Using this property, the following statement can be shown. It relates the final set of hypotheses H to the rules of the \mathbb{Z} -CERS \mathcal{R} and establishes condition 2 of the principle due to Koike and Toyama.

Lemma 13.39. *If $\langle E, \emptyset \rangle \vdash_{\mathcal{I}}^* \langle \emptyset, H \rangle$ using inference rules other than Theory_2 , then $\rightarrow_{H, \mathbb{Z}} \subseteq \rightarrow_{\mathcal{R}, \mathbb{Z}} \circ \rightarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* \circ \leftarrow_{\mathcal{R} \cup H, \mathbb{Z}}^*$ on \mathbb{Z} -normal ground terms.*

With these lemmas at hand, soundness of the inductive proof method based on the inference system \mathcal{I} can be shown.

Theorem 13.40. *For a quasi-reductive and terminating \mathbb{Z} -CERS \mathcal{R} , if $\langle E, \emptyset \rangle \vdash_{\mathcal{I}}^* \langle \emptyset, H \rangle$, then all atomic conjectures in E are inductive theorems of \mathcal{R} .*

Example 13.41. The function `minmax` computes a pair consisting of the minimum and the maximum of two integers.

$$\begin{aligned} \text{minmax}(x, y) &\rightarrow \text{pair}(y, x) \llbracket x \geq y \rrbracket \\ \text{minmax}(x, y) &\rightarrow \text{pair}(x, y) \llbracket y > x \rrbracket \end{aligned}$$

The following derivation is a proof of $\text{minmax}(x, y) \equiv \text{pair}(x, y) \llbracket y \geq x \rrbracket$:

$$\frac{\langle \{\text{minmax}(x, y) \equiv \text{pair}(x, y) \llbracket y \geq x \rrbracket\}, \emptyset \rangle}{\langle \{\text{pair}(y, x) \equiv \text{pair}(x, y) \llbracket y \geq x \wedge x \geq y \rrbracket, \text{pair}(x, y) \equiv \text{pair}(x, y) \llbracket y \geq x \wedge y > x \rrbracket\}, \{\text{minmax}(x, y) \rightarrow \text{pair}(x, y) \llbracket y \geq x \rrbracket\} \rangle} \text{Expand}$$

$$\frac{\langle \{\text{pair}(x, y) \equiv \text{pair}(x, y) \llbracket y \geq x \wedge y > x \rrbracket\}, \{\text{minmax}(x, y) \rightarrow \text{pair}(x, y) \llbracket y \geq x \rrbracket\} \rangle}{\langle \emptyset \{\text{minmax}(x, y) \rightarrow \text{pair}(x, y) \llbracket y \geq x \rrbracket\} \rangle} \text{Theory}_1 \text{Delete}$$

For the application of Theory_1 , notice that $y \geq x \wedge x \geq y \Rightarrow \text{pair}(y, x) \simeq \text{pair}(x, y)$ is LIAC-valid. △

While Theorem 13.40 only relies on the assumption that \mathcal{R} is quasi-reductive and terminating, the soundness of the inference rule Theory_2 that makes it possible to disprove atomic conjectures relies on \mathcal{R} being confluent.

Theorem 13.42. *For a quasi-reductive, confluent, and terminating \mathbb{Z} -CERS \mathcal{R} , if $\langle E, \emptyset \rangle \vdash_{\mathcal{I}}^* \perp$, then at least one conjecture in E is not an inductive theorem of \mathcal{R} .*

Example 13.43. The following normal \mathbb{Z} -CERS defines the function `app` that appends one list to the end of another list.

$$\begin{aligned} \text{app}(\text{nil}, ys) &\rightarrow ys \\ \text{app}(\text{cons}(x, xs), ys) &\rightarrow \text{cons}(x, \text{app}(xs, ys)) \end{aligned}$$

Then, consider the atomic conjecture $\mathbf{app}(xs, ys) \equiv ys$, for which the following derivation can be obtained:

$$\begin{array}{c}
 \langle \{\mathbf{app}(xs, ys) \equiv ys\}, \emptyset \rangle \\
 \hline
 \langle \{ys \equiv ys, \mathbf{cons}(x, \mathbf{app}(xs, ys)) \equiv ys\}, \{\mathbf{app}(xs, ys) \rightarrow ys\} \rangle \quad \text{Expand} \\
 \hline
 \langle \{\mathbf{cons}(x, \mathbf{app}(xs, ys)) \equiv ys\}, \{\mathbf{app}(xs, ys) \rightarrow ys\} \rangle \quad \text{Delete} \\
 \hline
 \langle \{\mathbf{cons}(x, ys) \equiv ys\}, \{\mathbf{app}(xs, ys) \rightarrow ys\} \rangle \quad \text{Simplify} \\
 \hline
 \langle \{\mathbf{cons}(x, ys) \equiv ys\}, \{\mathbf{app}(xs, ys) \rightarrow ys\} \rangle \quad \text{Theory}_2 \\
 \hline
 \perp
 \end{array}$$

Thus, the atomic conjecture $\mathbf{app}(xs, ys) \equiv ys$ is not an inductive theorem. \triangle

13.4 Summary

This chapter has presented an inductive proof method for \mathbb{Z} -CERSs. This proof method is based on the implicit induction paradigm and couples inductive reasoning with a decision procedure for the theory LIAC. Here, the (decidable) theory LIAC combines the linear theory of integers with the constructors of the \mathbb{Z} -CERS. The inductive proof method does not only make it possible to prove inductive conjectures, but also to disprove false conjectures. For this, the \mathbb{Z} -CERS needs to be confluent. Sufficient condition for ensuring confluence of \mathbb{Z} -CERSs have been developed in this chapter as well.

Chapter 14

Inductive Theorem Proving as a Decision Procedure

While the inference system \mathcal{I} from Chapter 13 provides a completely mechanical way to prove or disprove inductive conjectures once a strategy for the application of the inference rules has been fixed, it does not provide a decision procedure for inductive validity since derivations of the system may diverge or fail. The reason for a possible divergence is the inference rule **Expand** which could be applied again and again.

For methods employed in program verification, however, a decision procedure that can be used as a “black box” is preferable since an interactive use of inductive reasoning methods is typically only possible by trained experts. The goal of this chapter is to derive conditions on \mathbb{Z} -CERSs and conjectures under which the inference system \mathcal{I} can be used as a decision procedure, i.e., will always produce a proof or disproof of a conjecture if a suitable strategy on the use of the inference rules is employed. These conditions are mostly based on properties of the rewrite rules in a \mathbb{Z} -CERS that can be pre-computed during parsing. Thus, checking whether a

conjecture satisfies the conditions under which \mathcal{I} provides a decision procedure is easily possible and requires much less time than attempting a proof or disproof.

Work on identifying conditions under which inductive theorem proving provides a decision procedure with ordinary TRSs was initiated in [111] and later extended in [80, 82], also see [104]. These previous papers impose strong restrictions on both the TRSs and the conjectures. The functions defined by the TRS have to be given in such a way that any function f may only make recursive calls to the function f again. Often, it is necessary to allow calls to other auxiliary functions or even mutually recursive definitions. The first contribution of this chapter is to allow for both of these extension. All of [111, 80, 82, 104] impose the restriction that the conjectures contain a subterm of the form $f(x_1, \dots, x_n)$ for an $f \in \mathcal{D}(\mathcal{R})$ and pairwise distinct variables x_1, \dots, x_n . This term is then chosen as the basic subterm upon which the proof by induction is based. In this chapter, this restriction is relaxed as well by making it possible to have basic subterms where the arguments are not necessarily pairwise distinct variables.

In this chapter it is assumed that \mathcal{R} is a quasi-reductive, terminating normal \mathbb{Z} -CERS. Recall that \mathcal{R} is thus confluent, which is required for disproving conjectures.

14.1 Simple Decidable Conjectures

For the purpose of decidable induction, a restricted class of function definitions is considered. In its most simple form, functions may only make recursive calls to themselves. Furthermore, nesting of recursive calls is not permitted. This is captured by the following definition, adapted from [111].

Definition 14.1 (LIAC-Based Functions). *A function $g \in \mathcal{D}(\mathcal{R})$ is LIAC-based iff all right-hand sides of rules in $\mathcal{R}(g)$ have the form $C[g(r_1^*), \dots, g(r_m^*)]$ for some context*

C over $\widehat{\mathcal{C}}(\mathcal{R})$ such that $r_k^* \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$ for all $1 \leq k \leq m$.

The strategy for the application of the inference rules in \mathcal{I} that turns \mathcal{I} into a decision procedure on certain conjectures is quite natural: First, a basic subterm of the conjecture is expanded. One condition on the conjecture will ensure that there is only one basic subterm, thus eliminating the non-determinism caused by several such subterms. After expanding this basic subterm, the newly obtained conjectures are simplified using \mathcal{R} and the inductive hypothesis. For this, it needs to be ensured that the hypothesis is always applicable. Finally, further conditions on the conjecture ensure that inductive validity of the conjectures obtained after simplification can be decided using the decision procedure for LIAC, i.e., using the inference rules Theory_1 and Theory_2 .

The basic subterm used for expansion is restricted to have the form $g(x_1, \dots, x_n)$, where x_1, \dots, x_n are not necessarily pairwise distinct variables. This is a significant extension over [111, 80, 82, 104], where these variables need to be pairwise distinct. The reason for this restriction in [111, 80, 82, 104] is that the inductive hypothesis is then always applicable (under certain further assumptions discussed below).

In order to ensure that a non-linear hypothesis is applicable to all recursive calls of a LIAC-based function after application of the **Expand**-rule, it needs to be ensured that the corresponding arguments of the recursive calls are “equal”. More precisely, this needs to be required only under the assumption that these arguments are equal in the left-hand side of the rule since **Expand** does otherwise not create any new atomic conjectures to which the hypothesis needs to be applied. Notice that this property depends only on the rules in $\mathcal{R}(g)$ and is independent of the conjecture.

Definition 14.2 (*ImpEq*). *Let g be LIAC-based. Then $\langle i, j \rangle \in \text{ImpEq}(g)$ iff $1 \leq i < j \leq \text{arity}(g)$ such that the i^{th} and j^{th} argument of g have the same sort and*

$$l_i \simeq l_j \Rightarrow \bigwedge_{k=1}^m r_{k,i} \simeq r_{k,j}$$

is LIAC-valid for all rules $g(l^*) \rightarrow C[g(r_1^*), \dots, g(r_m^*)][[\varphi]] \in \mathcal{R}(f)$ where $l_i \simeq l_j \wedge \varphi$ is LIAC-satisfiable.

Hence, if a term of the form $g(l^*)\sigma$ is simplified using the rewrite rule $g(l^*) \rightarrow C[g(r_1^*), \dots, g(r_m^*)][[\varphi]]$ and $\langle i, j \rangle \in \mathcal{ImpEq}(g)$, then $r_{k,i}\sigma = r_{k,j}\sigma$ for all $1 \leq k \leq m$ whenever $l_i\sigma = l_j\sigma$. The set $\mathcal{ImpEq}(g)$ can easily be computed from the rules defining g with the help of a decision procedure for LIAC.

Example 14.3. The following normal \mathbb{Z} -CERS determines whether a list is point-wise bigger than another list of the same length.

$$\begin{aligned} \text{ptwise}(\text{nil}, \text{nil}) &\rightarrow \text{true} \\ \text{ptwise}(\text{nil}, \text{cons}(y, ys)) &\rightarrow \text{false} \\ \text{ptwise}(\text{cons}(x, xs), \text{nil}) &\rightarrow \text{false} \\ \text{ptwise}(\text{cons}(x, xs), \text{cons}(y, ys)) &\rightarrow \text{ptwise}(xs, ys) \llbracket x \geq y \rrbracket \\ \text{ptwise}(\text{cons}(x, xs), \text{cons}(y, ys)) &\rightarrow \text{false} \llbracket y > x \rrbracket \end{aligned}$$

Then $\mathcal{ImpEq}(\text{ptwise}) = \{\langle 1, 2 \rangle\}$. So see this, notice that the implications from Definition 14.2 are trivially true for the first, second, third, and fifth rules since these rules do not contain any recursive calls. For the fourth rule, the LIAC-validity of

$$\text{cons}(x, xs) \simeq \text{cons}(y, ys) \Rightarrow xs \simeq ys$$

is easily shown. △

The first version of decidable conjectures is now given as follows. Notice that only a simple form of basic terms is allowed, but that non-linearity is possible.

Definition 14.4 (Simple Conjectures). *A simple conjecture is an atomic conjecture of the form $g(x^*) \equiv t[[\varphi]]$ such that the following conditions are satisfied:*

1. $\mathcal{R} \cup \{g(x^*) \rightarrow t[[\varphi]]\}$ is terminating.

2. $\varphi = \top$
3. The function g is LIAC-based.
4. x^* consists of variables and $t \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$.
5. Whenever $x_i = x_j$ for $i < j$, then $\langle i, j \rangle \in \text{ImpEq}(g)$.

Example 14.5. For the \mathbb{Z} -CERS from Example 14.3, the conjecture $\text{ptwise}(xs, xs) \equiv \text{true}$ is simple. △

Theorem 14.6. Using the strategy¹ $\text{Expand} \cdot \text{Simplify}^! \cdot (\text{Theory}_1 + \text{Theory}_2)^!$, where Simplify uses only the hypothesis from H , it is decidable whether a simple conjecture is an inductive theorem.

The concept of LIAC-based functions is quite restrictive since a LIAC-based function may only make recursive calls to itself and not to any other function. The next definition generalizes the concept of a LIAC-based function by considering a set of function symbols that may make recursive calls to each other. Notice that nested recursive calls in right-hand sides are still not allowed.

Definition 14.7 (LIAC-Based Functions–Version 2). A set $\mathcal{G} = \{g_1, \dots, g_n\} \subseteq \mathcal{D}(\mathcal{R})$ is LIAC-based iff all right-hand sides of rules in $\mathcal{R}(\mathcal{G})$ are of the form $C[g_{k_1}(r_1^*), \dots, g_{k_m}(r_m^*)]$ for some context C over $\widehat{\mathcal{C}}(\mathcal{R})$ such that $r_i^* \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$ and $g_{k_i} \in \mathcal{G}$ for all $1 \leq i \leq m$.

Example 14.8. This example uses free constructors to model lists and natural numbers using a Peano representation.

$$\begin{aligned} \max(\mathcal{O}, y) &\rightarrow y \\ \max(\text{s}(x), \mathcal{O}) &\rightarrow \text{s}(x) \\ \max(\text{s}(x), \text{s}(y)) &\rightarrow \text{s}(\max(x, y)) \end{aligned}$$

¹In stating strategies, \cdot denotes sequential application, $+$ denotes alternative application, $*$ denotes iterated application, and $^!$ denotes exhaustive iterative application.

$$\begin{aligned}
 \text{maxlist}(xs, \text{nil}) &\rightarrow \text{nil} \\
 \text{maxlist}(\text{nil}, \text{cons}(y, ys)) &\rightarrow \text{nil} \\
 \text{maxlist}(\text{cons}(x, xs), \text{cons}(y, ys)) &\rightarrow \text{cons}(\text{max}(x, y), \text{maxlist}(xs, ys))
 \end{aligned}$$

Since `maxlist` makes a recursive call to `max`, the function `maxlist` is not LIAC-based. The set $\{\text{max}, \text{maxlist}\}$, however, is LIAC-based. \triangle

In order to ensure that non-linear hypotheses are still applicable, the definition of ImpEq needs to be adapted as well. For this, the idea is to collect conditions on all members of a LIAC-based set of functions under which recursive calls to one of these functions g have equal arguments in positions i and j . These conditions are of the form $\langle g', i', j' \rangle$, meaning that equality of the arguments in positions i' and j' of the function g' ensures that recursive calls to g have equal arguments in positions i and j .

Definition 14.9 (*ImpEq-Version 2*). Let $\mathcal{G} = \{g_1, \dots, g_n\}$ be a LIAC-based set of functions. Then $\langle g, i, j, \Gamma \rangle \in \text{ImpEq}(\mathcal{G})$ for $g \in \mathcal{G}$ iff $1 \leq i < j \leq \text{arity}(g)$, the i^{th} and j^{th} argument of g have the same sort, and $\Gamma = \{\langle g_{k_1}, i_1, j_1 \rangle, \dots, \langle g_{k_m}, i_m, j_m \rangle\}$ such that for all $1 \leq \kappa \leq m$, $1 \leq i_\kappa < j_\kappa \leq \text{arity}(g_{k_\kappa})$ where the i_κ^{th} and j_κ^{th} argument of g_{k_κ} have the same sort and

$$\bigwedge_{\langle g_k, i', j' \rangle \in \Gamma} l_{i'} \simeq l_{j'} \Rightarrow \bigwedge_{g_{k_\kappa} = g} r_{\kappa, i} \simeq r_{\kappa, j}$$

is LIAC-valid for all rules $g_k(l^*) \rightarrow C[g_{k_1}(r_1^*), \dots, g_{k_m}(r_m^*)][[\varphi]] \in \mathcal{R}(\mathcal{G})$ for which

$$\bigwedge_{\langle g_k, i', j' \rangle \in \Gamma} l_{i'} \simeq l_{j'} \wedge \varphi \text{ is LIAC-satisfiable.}$$

Notice that the set $\text{ImpEq}(\mathcal{G})$ is still easily computable from the rules defining \mathcal{G} with the help of a decision procedure for LIAC.

Definition 14.9 strictly generalizes Definition 14.2 even for a single LIAC-based function. To see this, consider the rule $f(\text{cons}(x, xs), \text{cons}(y, ys), \text{cons}(z, zs)) \rightarrow$

$f(ys, zs, xs)$. Using Definition 14.2, $\mathcal{I}mp\mathcal{E}q(f) = \emptyset$ since none of

$$\begin{aligned} \text{cons}(x, xs) \simeq \text{cons}(y, ys) &\Rightarrow ys \simeq zs \\ \text{cons}(x, xs) \simeq \text{cons}(z, zs) &\Rightarrow ys \simeq xs \\ \text{cons}(y, ys) \simeq \text{cons}(z, zs) &\Rightarrow zs \simeq xs \end{aligned}$$

is LIAC-valid. Using Definition 14.9, $\mathcal{I}mp\mathcal{E}q(\{f\})$ is non-empty, containing (amongst others) $\langle f, 1, 2, \{\langle f, 2, 3 \rangle\} \rangle$. Notice, however, that $\mathcal{I}mp\mathcal{E}q(g)$ from Definition 14.2 and $\mathcal{I}mp\mathcal{E}q(\{g\})$ from Definition 14.9 coincide if g is a LIAC-based function of arity 2.

The definition of a simple conjecture immediately generalizes to LIAC-based sets \mathcal{G} of functions. Now, an atomic conjecture for each member of the \mathcal{G} is needed. Also, notice the use of $\mathcal{I}mp\mathcal{E}q(\mathcal{G})$ to ensure applicability of the inductive hypotheses.

Definition 14.10 (Simple Conjectures–Version 2). *A simple conjecture is a set of atomic conjectures of the form $\{g_1(x_1^*) \equiv t_1[\varphi_1], \dots, g_n(x_n^*) \equiv t_n[\varphi_n]\}$ such that the following conditions are satisfied:*

1. $\mathcal{R} \cup \{g_1(x_1^*) \rightarrow t_1[\varphi_1], \dots, g_n(x_n^*) \rightarrow t_n[\varphi_n]\}$ is terminating.
2. $\varphi_i = \top$ for all $1 \leq i \leq n$.
3. The set $\mathcal{G} = \{g_1, \dots, g_n\}$ is LIAC-based.
4. x_i^* consists of variables and $t_i \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$ for all $1 \leq i \leq n$.
5. Whenever $x_{k,i} = x_{k,j}$ for $i < j$, then there exists an $\langle g_k, i, j, \Gamma \rangle \in \mathcal{I}mp\mathcal{E}q(\mathcal{G})$ such that $x_{k',i'} = x_{k',j'}$ for all $\langle g_{k'}, i', j' \rangle \in \Gamma$.

Example 14.11. In Example 14.8, the set $\{\text{max}(x, x) \equiv x, \text{maxlist}(xs, xs) \equiv xs\}$ is a simple conjecture. To see this, notice that $\langle \text{maxlist}, 1, 2, \{\langle \text{maxlist}, 1, 2 \rangle\} \rangle$ and $\langle \text{max}, 1, 2, \{\langle \text{max}, 1, 2 \rangle, \langle \text{maxlist}, 1, 2 \rangle\} \rangle$ are in $\mathcal{I}mp\mathcal{E}q(\{\text{max}, \text{maxlist}\})$ since the implications

$$s(x) \simeq s(y) \Rightarrow x \simeq y$$

$$\text{cons}(x, xs) \simeq \text{cons}(y, ys) \Rightarrow x \simeq y$$

$$\text{cons}(x, xs) \simeq \text{cons}(y, ys) \Rightarrow xs \simeq ys$$

are LIAC-valid. △

Theorem 14.12. *Using the strategy $\text{Expand}^* \cdot \text{Simplify}^!$ · (Theory₁ + Theory₂)[!], where Expand is applied once to each atomic conjecture of the set and Simplify uses only the hypotheses from H , it is decidable whether a simple conjecture is an inductive theorem.*

Example 14.13. All of the following conjectures are simple (see <http://www.cs.unm.edu/~spf/sail2/> for the function definitions):

| | |
|---|--|
| $\text{app}(xs, ys) \equiv ys$ | $\text{gtr}(x, x) \equiv \text{false}$ |
| $\text{geq}(x, x) \equiv \text{true}$ | $\text{maxlist}(xs, xs) \equiv xs$ |
| $\{\text{maxlist}(xs, xs) \equiv xs, \text{max}(x, x) \equiv x\}$ | $\text{max}(x, x) \equiv x$ |
| $\text{min}(x, x) \equiv x$ | $\text{minus}(x, x) \equiv \mathcal{O}$ |
| $\{\text{mix}(xs, xs) \equiv xs, \text{mix}'(xs, xs) \equiv xs\}$ | $\text{plus}(x, y) \equiv x$ |
| $\text{prefix}(xs, xs) \equiv xs$ | $\text{ptwise}(xs, xs) \equiv \text{true}$ |

According to Theorems 14.6 and 14.12, it can be decided whether these conjectures are inductive theorems. △

14.2 Simple Decidable Conjectures with Nesting

The main restriction of the simple decidable conjectures from Section 14.1 is that nesting of defined function symbols is not permitted. This restriction was imposed in order to ensure that the inductive hypotheses are always applicable (if the *ImpEq*-requirement is satisfied), resulting in an atomic conjecture whose validity can be

decided using the inference rules Theory_1 or Theory_2 that make use of a decision for the theory LIAC.

For atomic conjectures with nested defined function symbols, this is not always the case since **Expand** might introduce a context from the right-hand side of a rule around the recursive calls. This context needs to be removed before the inductive hypotheses can be applied. This observation leads to the concept of *compatibility*, meaning that the \mathbb{Z} -CERS can handle the contexts introduced in right-hand sides of rules. This concept was initially defined in [111], but the presentation in this section follows the presentation in [82] which presents similar results for ordinary TRSs.

In order to present the ideas developed in this section uniformly for both LIAC-based functions and LIAC-based set of functions, the following definition generalizes these concepts.

Definition 14.14 (LIAC-Good Rewrite Rules and Functions). *A constrained rewrite rule $l \rightarrow r[[\varphi]]$ is LIAC-good iff $r = C[g_1(r_1^*), \dots, g_n(r_n^*)]$ where C is a context over $\widehat{\mathcal{C}}(\mathcal{R})$ and $r_1^*, \dots, r_n^* \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$ for $g_1, \dots, g_n \in \mathcal{D}(\mathcal{R})$. A function $f \in \mathcal{D}(\mathcal{R})$ is LIAC-good if all rules in $\mathcal{R}(f)$ are LIAC-good.*

Notice that, in particular, the rules of a LIAC-based function or a LIAC-based set of functions are LIAC-good.

Definition 14.15 (Compatibility). *Let g be LIAC-based, let $1 \leq j \leq \text{arity}(g)$, and let \mathcal{Q} be a set of LIAC-good rewrite rules. Then g is compatible with \mathcal{Q} on argument j iff for all² $f(l^*) \rightarrow C[g_1(r_1^*), \dots, g_n(r_n^*)][[\varphi]] \in \mathcal{Q}$ such that the j^{th} argument of g has the same sort as f ,*

$$\frac{g(x_1, \dots, x_{j-1}, C[z_1, \dots, z_n], x_{j+1}, \dots, x_m) \rightarrow_{\mathcal{R}, \mathbb{Z}}^* D[g(x_1, \dots, x_{j-1}, z_{i_1}, x_{j+1}, \dots, x_m), \dots, g(x_1, \dots, x_{j-1}, z_{i_k}, x_{j+1}, \dots, x_m)]}{}$$

²In [82], *compatibility with exceptions* has been defined. In order to keep the presentation simple, this is not considered in this dissertation.

for a context D over $\widehat{\mathcal{C}}(\mathcal{R})$ and $i_1, \dots, i_k \in \{1, \dots, n\}$ such that $z_i \notin \mathcal{V}(D)$ for all $1 \leq i \leq n$.

If $\mathcal{Q} = \mathcal{R}(f)$ for a function symbol f , then g is said to be compatible with f on argument j , and similarly for sets of function symbols.

Example 14.16. Consider the following \mathbb{Z} -CERS:

$$\begin{aligned} \text{zip}(xs, \text{nil}) &\rightarrow \text{pnil} \\ \text{zip}(\text{nil}, \text{cons}(y, ys)) &\rightarrow \text{pnil} \\ \text{zip}(\text{cons}(x, xs), \text{cons}(y, ys)) &\rightarrow \text{pcons}(\text{pair}(x, y), \text{zip}(xs, ys)) \\ \text{fst}(\text{pnil}) &\rightarrow \text{nil} \\ \text{fst}(\text{pcons}(\text{pair}(x, y), xs)) &\rightarrow \text{cons}(x, \text{fst}(xs)) \end{aligned}$$

Then fst is compatible with zip on argument 1. For the first two zip -rules, C is pnil (a context without holes), and $\text{fst}(\text{pnil})$ rewrites to nil , i.e., $D = \text{nil}$. For the third zip -rule, C is $\text{pcons}(\text{pair}(x, y), \square)$ and $\text{fst}(\text{pcons}(\text{pair}(x, y), z_1))$ rewrites to $\text{cons}(x, \text{fst}(z_1))$, i.e., $D = \text{cons}(x, \square)$. \triangle

While Definition 14.15 considers the rules in \mathcal{Q} independently for each context C from a right-hand side, the property from the definition can be lifted to nested contexts C that are obtained from several rules' right-hand sides. These contexts can be obtained if several rules from \mathcal{Q} are applied after another.

Definition 14.17 (Repeated \mathcal{Q} -Contexts). *Let \mathcal{Q} be a set of LIAC-good rewrite rules. A context C is a \mathcal{Q} -context iff there exists a constrained rewrite rule $f(l^*) \rightarrow C[g_1(r_1^*), \dots, g_n(r_n^*)][[\varphi]] \in \mathcal{Q}$. A context C is a repeated \mathcal{Q} -context iff C is a \mathcal{Q} -context or there are repeated \mathcal{Q} -contexts D, C_1, \dots, C_m such that $C = D[C_1, \dots, C_m]$.*

If $\mathcal{Q} = \mathcal{R}(f)$, (repeated) \mathcal{Q} -contexts are also called (repeated) f -contexts, and similarly for sets of function symbols. Now, it can be shown that the property from Definition 14.15 lifts from the \mathcal{Q} -contexts considered there to repeated \mathcal{Q} -contexts.

Lemma 14.18. *Let g be compatible with \mathcal{Q} on argument j . Then, for every repeated \mathcal{Q} -context $C_{\mathcal{Q}}$, there exists a repeated g -context C_g such that*

$$\begin{aligned} g(x_1, \dots, x_{j-1}, C_{\mathcal{Q}}[z_1, \dots, z_n], x_{j+1}, \dots, x_m) &\rightarrow_{\mathcal{R}, \mathcal{Z}}^* \\ C_g[g(x_1, \dots, x_{j-1}, z_{i_1}, x_{j+1}, \dots, x_m), \dots, g(x_1, \dots, x_{j-1}, z_{i_k}, x_{j+1}, \dots, x_m)] \end{aligned}$$

where $i_1, \dots, i_k \in \{1, \dots, n\}$ and $z_i \notin \mathcal{V}(C_g)$ for all $1 \leq i \leq n$.

The concept of compatibility can be extended to arbitrarily deep nestings of functions, resulting in *compatibility sequences*.

Definition 14.19 (Compatibility Sequences). *Let f_1, \dots, f_{d-1} be LIAC-based and let f_d be LIAC-good for some $d \geq 1$. The sequence $\langle f_1, \dots, f_d \rangle$ is a compatibility sequence on arguments $\langle j_1, \dots, j_{d-1} \rangle$ iff f_i is compatible with f_{i+1} on argument j_i for all $1 \leq i \leq d-1$.*

A term $s \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ has this compatibility sequence iff

$$s = f_1(p_1^*, f_2(p_2^*, \dots, f_{d-1}(p_{d-1}^*, f_d(x^*), q_{d-1}^*) \dots, q_2^*), q_1^*)$$

such that the variables in x^* do not occur elsewhere in s , the p_i^* and q_i^* are from $\mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$, and $f_i(p_i^*, f_{i+1}(\dots), q_i^*)|_{j_i} = f_{i+1}(\dots)$ for all $1 \leq i \leq d-1$.

If s is as in this definition, then $s\langle t \rangle$ denotes the term obtained from s by replacing the term $f_d(x^*)$ by the term t . Then, it can be shown that any f_d -context can be handled in a term having a compatibility sequence.

Lemma 14.20. *Let s be a term with the compatibility sequence $\langle f_1, \dots, f_d \rangle$ on arguments $\langle j_1, \dots, j_{d-1} \rangle$. Then, for every rule $f_d(l^*) \rightarrow C[g_1(r_1^*), \dots, g_n(r_n^*)][[\varphi]]$,*

$$s\langle C[g_1(r_1^*), \dots, g_n(r_n^*)] \rangle \rightarrow_{\mathcal{R}, \mathcal{Z}}^* D[s\langle g_{i_1}(r_{i_1}^*) \rangle, \dots, s\langle g_{i_k}(r_{i_k}^*) \rangle]$$

for some context D over $\widehat{\mathcal{C}}(\mathcal{R})$ and $i_1, \dots, i_k \in \{1, \dots, n\}$.

Chapter 14. Inductive Theorem Proving as a Decision Procedure

Now *simple nested conjectures* generalize the simple conjectures from Section 14.1 by allowing nested defined functions on the left-hand side, provided the left-hand side has a compatibility sequence.

Definition 14.21 (Simple Nested Conjectures). *A simple nested conjecture is an atomic conjecture of the form $D[f(x^*)] \equiv t[\varphi]$ such that the following conditions are satisfied:*

1. $\mathcal{R} \cup \{D[f(x^*)] \rightarrow t[\varphi]\}$ is terminating.
2. $\varphi = \top$
3. The term $D[f(x^*)]$ has a compatibility sequence and f is LIAC-based.
4. x^* consists of variables and $t \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$.
5. Whenever $x_i = x_j$ for $i < j$, then $\langle i, j \rangle \in \text{ImpEq}(f)$.

Example 14.22. Continuing Example 14.16, the term $\text{fst}(\text{zip}(xs, xs))$ has the compatibility sequence $\langle \text{fst}, \text{zip} \rangle$ on arguments $\langle 1 \rangle$. Furthermore, $\langle 1, 2 \rangle \in \text{ImpEq}(\text{zip})$. Thus, $\text{fst}(\text{zip}(xs, xs)) \equiv xs$ is a simple nested conjecture. \triangle

Theorem 14.23. *Using the strategy $\text{Expand} \cdot \text{Simplify}^* \cdot (\text{Theory}_1 + \text{Theory}_2)^!$, it is decidable whether a simple nested conjecture is an inductive theorem.*

Of course, the concept of simple nested conjectures can be extended from LIAC-based functions to LIAC-based sets of functions, similarly to how this was done for simple conjectures in Section 14.1. First, notice that the definition of compatibility can already be applied to LIAC-based sets of functions.

Example 14.24. Take the function fst defined in Example 14.16 and add the following rules defining stitch .

$$\begin{aligned}
\text{stitch}(x, \text{nil}) &\rightarrow \text{pnil} \\
\text{stitch}(\text{nil}, \text{cons}(y, ys)) &\rightarrow \text{pnil} \\
\text{stitch}(\text{cons}(x, xs), \text{cons}(y, ys)) &\rightarrow \text{pcons}(\text{pair}(x, y), \text{stitch}'(xs, ys)) \\
\text{stitch}'(x, \text{nil}) &\rightarrow \text{pnil} \\
\text{stitch}'(\text{nil}, \text{cons}(y, ys)) &\rightarrow \text{pnil} \\
\text{stitch}'(\text{cons}(x, xs), \text{cons}(y, ys)) &\rightarrow \text{pcons}(\text{pair}(y, x), \text{stitch}(xs, ys))
\end{aligned}$$

Then $\mathcal{G} = \{\text{stitch}, \text{stitch}'\}$ is LIAC-based and fst is compatible with \mathcal{G} on argument 1, since for the third stitch -rule, the term $\text{fst}(\text{pcons}(\text{pair}(x, y), z_1))$ rewrites to $\text{cons}(x, \text{fst}(z_1))$, and similarly for the third stitch' -rule. \triangle

Now, the definition of simple nested conjectures can be revised as follows.

Definition 14.25 (Simple Nested Conjectures–Version 2). *A simple nested conjecture is a set of atomic conjectures of the form $\{D[f_1(x_1^*)] \equiv t_1[\varphi_1], \dots, D[f_n(x_n^*)] \equiv t_n[\varphi_n]\}$ such that the following conditions are satisfied:*

1. $\mathcal{R} \cup \{D[f_1(x_1^*)] \rightarrow t_1[\varphi_1], \dots, D[f_n(x_n^*)] \rightarrow t_n[\varphi_n]\}$ is terminating.
2. $\varphi_i = \top$ for all $1 \leq i \leq n$.
3. All of $D[f_1(x_1^*)], \dots, D[f_n(x_n^*)]$ have a compatibility sequence and the set $\mathcal{G} = \{f_1, \dots, f_n\}$ is LIAC-based.
4. x_i^* consists of variables and $t_i \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$ for all $1 \leq i \leq n$.
5. Whenever $x_{k,i} = x_{k,j}$ for $i < j$, then there exists an $\langle f_k, i, j, \Gamma \rangle \in \text{ImpEq}(\mathcal{G})$ such that $x_{k',i'} = x_{k',j'}$ for all $\langle f_{k'}, i', j' \rangle \in \Gamma$.

Example 14.26. In Example 14.24, the term $\text{fst}(\text{stitch}(xs, xs))$ has the compatibility sequence $\langle \text{fst}, \text{stitch} \rangle$ on arguments $\langle 1 \rangle$, and the term $\text{fst}(\text{stitch}'(xs, xs))$ has the compatibility sequence $\langle \text{fst}, \text{stitch}' \rangle$ on arguments $\langle 1 \rangle$. Since $\{\text{stitch}, \text{stitch}'\}$ is LIAC-based, $\{\text{fst}(\text{stitch}(xs, xs)) \equiv xs, \text{fst}(\text{stitch}'(xs, xs)) \equiv xs\}$ is a simple nested

conjecture because $\text{ImpEq}(\{\text{stitch}, \text{stitch}'\})$ contains $\langle \text{stitch}, 1, 2, \{\langle \text{stitch}', 1, 2 \rangle\} \rangle$ and $\langle \text{stitch}', 1, 2, \{\langle \text{stitch}, 1, 2 \rangle\} \rangle$. \triangle

Theorem 14.27. *Using the strategy $\text{Expand}^* \cdot \text{Simplify}^* \cdot (\text{Theory}_1 + \text{Theory}_2)^!$, where Expand is applied once to each atomic conjecture of the set, it is decidable whether a simple nested conjecture is an inductive theorem.*

Example 14.28. All of the following conjectures are simple nested, but not simple:

$$\begin{array}{ll}
 \text{oddlst}(\text{alternate}(xs, xs)) \equiv xs & \text{evenlst}(\text{alternate}(xs, xs)) \equiv xs \\
 \text{oddlst}(\text{alternate}(xs, ys)) \equiv xs & \text{evenlst}(\text{alternate}(xs, ys)) \equiv ys \\
 \text{half}(\text{double}(x)) \equiv x & \text{even}(\text{double}(x)) \equiv \text{true} \\
 \text{not}(\text{gtr}(x, x)) \equiv \text{true} & \text{not}(\text{geq}(x, x)) \equiv \text{false} \\
 \text{fst}(\text{zip}(xs, xs)) \equiv xs & \text{fst}(\text{zip}(xs, ys)) \equiv xs \\
 \{\text{fst}(\text{stitch}(xs, xs)) \equiv xs, \text{fst}(\text{stitch}'(xs, xs)) \equiv xs\} & \\
 \{\text{fst}(\text{stitch}(xs, ys)) \equiv xs, \text{fst}(\text{stitch}'(xs, ys)) \equiv xs\} &
 \end{array}$$

According to Theorems 14.23 and 14.27, it can be decided whether these conjectures are inductive theorems. \triangle

14.3 Safe Generalizations

The conditions imposed on simple (nested) conjectures in Sections 14.1 and 14.2 ensure that the inductive hypotheses are always applicable and that an atomic conjecture containing no defined function symbols is obtained after application of the hypotheses. This is due to the restriction that the right-hand side of simple (nested) conjectures do not contain any defined symbols. For conjectures that contain defined symbols in both sides it is no longer the case that an atomic conjecture containing no defined function symbols is obtained after application of the inductive hypotheses. Thus, neither of the inference rules Theory_1 nor Theory_2 is applicable in general.

| | |
|---|---|
| $\text{Theory}'_1 \frac{\langle E \uplus \{s \doteq t[\varphi]\}, H \rangle}{\langle E, H \rangle}$ | if $s' \equiv t'[\varphi']$ is a safe generalization of $s \equiv t[\varphi]$ such that s', t' do not contain symbols from $\mathcal{D}(\mathcal{R})$ and $\varphi' \Rightarrow s' \simeq t'$ is LIAC-valid |
| $\text{Theory}'_2 \frac{\langle E \uplus \{s \doteq t[\varphi]\}, H \rangle}{\perp}$ | if $s' \equiv t'[\varphi']$ is a safe generalization of $s \equiv t[\varphi]$ such that s', t' do not contain symbols from $\mathcal{D}(\mathcal{R})$ and $\varphi' \Rightarrow s' \simeq t'$ is not LIAC-valid |

Figure 14.1: The inference system \mathcal{I}' is obtained from \mathcal{I} by replacing Theory_1 by Theory'_1 and Theory_2 by Theory'_2 .

In order to circumvent this problem, a generalization that replaces subterms with a defined root symbol by fresh variables may be applied to the atomic conjecture. While this approach is clearly sound (i.e., if the generalized conjecture is an inductive theorem, then the original conjecture is an inductive theorem), it is not complete in general (i.e., the generalized conjecture might not be an inductive theorem, even though the original conjecture is an inductive theorem). In order to employ inductive reasoning as a decision procedure, these over-generalizations need to be ruled out, i.e., only *safe* generalizations should be considered.

Definition 14.29 (Safe Generalizations). *Let $s \equiv t[\varphi]$ be an atomic conjecture. A safe generalization of $s \equiv t[\varphi]$ is an atomic conjecture $s' \equiv t'[\varphi']$ such that $s \equiv t[\varphi]$ is an inductive theorem iff $s' \equiv t'[\varphi']$ is an inductive theorem.*

Safe generalizations can be added to the inference system \mathcal{I} from Section 13.3 by modifying the inference rules Theory_1 and Theory_2 . For this, a safe generalization that eliminates all defined symbols from the conjecture is applied before the decision procedure for LIAC is employed. The inference rules Theory'_1 and Theory'_2 given in Figure 14.1 are based on this idea, giving rise to the inference system \mathcal{I}' .

The statements of Theorems 13.40 and 13.42 extend to the system \mathcal{I}' , i.e., the system is still correct and makes it possible to disprove conjectures.

Theorem 14.30. *For a quasi-reductive, confluent, and terminating \mathbb{Z} -CERS \mathcal{R} , if $\langle E, \emptyset \rangle \vdash_{\mathcal{T}}^* \langle \emptyset, H \rangle$, then all atomic conjectures in E are inductive theorems of \mathcal{R} .*

Theorem 14.31. *For a quasi-reductive, confluent, and terminating \mathbb{Z} -CERS \mathcal{R} , if $\langle E, \emptyset \rangle \vdash_{\mathcal{T}}^* \perp$, then at least one atomic conjecture in E is not an inductive theorem of \mathcal{R} .*

In the following, it is assumed that all rules in \mathcal{R} have the constraint \top . Consequently, $l \rightarrow r$ is written instead of $l \rightarrow r[\varphi]$. Extending the ideas presented in the following to \mathbb{Z} -CERSs with constrained rewrite rules is left for future work.

In order to obtain safe generalizations, [82] has introduced the *no-theory condition*, meaning that a given term is not equivalent to any term containing no defined function symbols.

Definition 14.32 (No-Theory Condition). *A \mathbb{Z} -free term t satisfies the no-theory condition (w.r.t. \mathcal{R}) iff there exists no term $q \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$ such that $t \equiv q$ is an inductive theorem of \mathcal{R} . The defined function $f \in \mathcal{D}(\mathcal{R})$ satisfies the no-theory condition iff the term $f(x^*)$ satisfies the no-theory condition for pairwise distinct variables x_1, \dots, x_n .*

As already noticed in [82], the no-theory condition is in practice satisfied for almost all defined functions since defined functions that do not satisfy the no-theory condition could just be replaced by the term q from Definition 14.32.

Using the no-theory condition, safe generalizations can be obtained by replacing terms that satisfy this condition by fresh variables.

Theorem 14.33. *Assume that $t_1, \dots, t_n, s_1, \dots, s_m$ are pairwise equal or variable-disjoint terms satisfying the no-theory condition. For all contexts C, D over $\widehat{\mathcal{C}}(\mathcal{R})$ and fresh variables x_{t_i}, x_{s_j} , the atomic conjecture $C[t_1, \dots, t_n] \equiv D[s_1, \dots, s_m]$ is an inductive theorem if and only if $C[x_{t_1}, \dots, x_{t_n}] \simeq D[x_{s_1}, \dots, x_{s_m}]$ is LIAC-valid.*

It is in general unclear how to check whether a term satisfies the no-theory condition.³ In the following, this problem is investigated for defined functions, i.e., conditions under which a defined function f satisfies the no-theory condition are derived. For LIAC-based functions, the method based on *candidate sets* as defined in [82] can be used. For this, the right-hand side of a non-recursive f -rule is used in order to obtain a finite set of candidates for the term q from Definition 14.32.

Definition 14.34 (Candidate Sets). *Let f be a LIAC-based function of arity n . The candidate set $Q(f)$ is defined as $Q_{s^*}(r)$ for a non-recursive rule $f(s_1, \dots, s_n) \rightarrow r \in \mathcal{R}(f)$. For the definition of $Q_{s^*}(r)$, let $x^* = x_1, \dots, x_n$ be pairwise distinct fresh variables and define $Q_{s^*}(t)$ for any $t \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$:*

1. $Q_{s^*}(x) = \{x_i \mid s_i = x\}$ if $x \in \mathcal{V}$
2. $Q_{s^*}(c(t_1, \dots, t_k)) = \{x_i \mid s_i = c(t_1, \dots, t_k)\} \cup \{c(q_1, \dots, q_k) \mid q_i \in Q_{s^*}(t_i) \text{ for all } 1 \leq i \leq k\}$

As in [82], candidate sets can be used in order to conclude that a LIAC-based function satisfies the no-theory condition.

Theorem 14.35. *Let f be LIAC-based. Then f satisfies the no-theory condition if for every $q \in Q(f)$, there exists a rule $l \rightarrow r \in \mathcal{R}(f)$ such that $l \downarrow_{f(x^*) \rightarrow q} \neq r \downarrow_{f(x^*) \rightarrow q}$. Here, $\downarrow_{f(x^*) \rightarrow q}$ denotes normalization by the rewrite rule $f(x^*) \rightarrow q$.*

Example 14.36. Consider the following function definition.

$$\begin{aligned} \text{len}(\text{nil}) &\rightarrow \mathcal{O} \\ \text{len}(\text{cons}(x, xs)) &\rightarrow \text{s}(\text{len}(xs)) \end{aligned}$$

Then, it can be shown that len satisfies the no-theory condition. To see this, notice that $Q(\text{len}) = Q_{\text{nil}}(\mathcal{O}) = \{\mathcal{O}\}$ and $\text{len}(\text{cons}(x, xs)) \downarrow_{\text{len}(x_1) \rightarrow \mathcal{O}} = \mathcal{O} \neq \text{s}(\mathcal{O}) = \text{s}(\text{len}(xs)) \downarrow_{\text{len}(x_1) \rightarrow \mathcal{O}}$. △

³Some sufficient conditions for this are given in [82].

Deriving conditions under which a member of a LIAC-based set of functions satisfies the no-theory condition is harder. As a first special case, simple mutual recursive functions can be handled by *unrolling* [109].

Definition 14.37 (Simple Mutual Recursion). *Let $\{f_0, f_1\}$ be a LIAC-based set of functions. Then $\{f_0, f_1\}$ is simple mutual recursive iff all f_i -rules for $i = 0, 1$ have the form $f_i(l^*) \rightarrow C[f_{1-i}(x_1^*), \dots, f_{1-i}(x_n^*)]$ such that x_j^* consists of pairwise distinct variables for all $1 \leq j \leq n$ and the x_1^*, \dots, x_n^* are pairwise disjoint.*

Simple mutual recursive definitions can automatically be transformed into definitions that only use direct recursion. For this, the recursive calls to f_{1-i} on the right-hand sides are matched to the left-hand sides of the f_{1-i} -rules.

Definition 14.38 (Unrolling). *Let $\{f_0, f_1\}$ be simple mutual recursive. The unrolling of the f_i -rule $f_i(l^*) \rightarrow C[f_{1-i}(x_1^*), \dots, f_{1-i}(x_n^*)]$ is the set*

$$\begin{aligned} \{f_i(l^*)\tau \rightarrow C\tau[r_1, \dots, r_n] \mid f_{1-i}(l_1^*) \rightarrow r_1, \dots, f_{1-i}(l_n^*) \rightarrow r_n \in \mathcal{R}(f_{1-i}), \\ \tau_j = \{x_j^* \mapsto l_j^*\} \text{ for all } 1 \leq j \leq n, \\ \text{and } \tau = \tau_1 \cup \dots \cup \tau_n\} \end{aligned}$$

The unrolling of f_i is the union of the unrollings of all f_i -rules.

Example 14.39. Consider the following simple mutual recursive definitions of even and odd.

$$\begin{aligned} \text{even}(\mathcal{O}) &\rightarrow \text{true} \\ \text{even}(s(x)) &\rightarrow \text{odd}(x) \\ \text{odd}(\mathcal{O}) &\rightarrow \text{false} \\ \text{odd}(s(x)) &\rightarrow \text{even}(x) \end{aligned}$$

Then the unrollings of even and odd are as follows:

$$\begin{aligned}
 \text{even}(\mathcal{O}) &\rightarrow \text{true} \\
 \text{even}(s(\mathcal{O})) &\rightarrow \text{false} \\
 \text{even}(s(s(x))) &\rightarrow \text{even}(x) \\
 \text{odd}(\mathcal{O}) &\rightarrow \text{false} \\
 \text{odd}(s(\mathcal{O})) &\rightarrow \text{true} \\
 \text{odd}(s(s(x))) &\rightarrow \text{odd}(x)
 \end{aligned}$$

Notice that a single `even`-rule gives rise to two `even`-rules in the unrolling, and similarly for `odd`. △

Theorem 14.40. *Let $\{f_0, f_1\}$ be simple mutual recursive. Then f_i satisfies the no-theory condition w.r.t. \mathcal{R} if f_i satisfies the no-theory condition w.r.t. f_i 's unrolling.*

Example 14.41. Using this result, it can be shown that `even` from Example 14.39 satisfies the no-theory condition by using its unrolled definition and Theorem 14.35. Using the first rule of the unrolling, $Q(\text{even}) = \{\text{true}\}$, but $\text{even}(s(\mathcal{O})) \downarrow_{\text{even}(x_1) \rightarrow \text{true}} = \text{true} \neq \text{false} = \text{false} \downarrow_{\text{even}(x_1) \rightarrow \text{true}}$. Using similar reasoning, it can also be shown that `odd` satisfies the no-theory condition. △

For functions that are neither LIAC-based nor members of a simple mutual recursive set of functions, Theorem 14.35 can be adapted as follows. The only difference is that subterms with a defined root symbol are abstracted by fresh variables after normalization with $f(x^*) \rightarrow q$ and that instead of testing for equality a test for LIAC-satisfiability is performed.

Theorem 14.42. *Let f be LIAC-good and let $\text{CAP}_{\mathcal{D}}$ replaces subterms with defined root symbols by fresh variables. Then f satisfies the no-theory condition if for every $q \in Q(f)$, there exists a rule $l \rightarrow r \in \mathcal{R}(f)$ such that $l \downarrow_{f(x^*) \rightarrow q} \simeq \text{CAP}_{\mathcal{D}}(r \downarrow_{f(x^*) \rightarrow q})$ is LIAC-unsatisfiable.*

Example 14.43. With this result, it can be shown that `maxlist` from Example 14.8 satisfies the no-theory condition. Using the first rule, $Q(\text{maxlist}) = \{x_2, \text{nil}\}$ is

obtained. For the candidate x_2 , a contradiction is obtained with the second **maxlist**-rule since $\text{maxlist}(\text{nil}, \text{cons}(y, ys)) \downarrow_{\text{maxlist}(x_1, x_2) \rightarrow x_2} = \text{cons}(y, ys)$ and $\text{cons}(y, ys) \simeq \text{nil}$ is LIAC-unsatisfiable. For nil , the third **maxlist**-rule gives rise to a contradiction since $\text{CAP}_{\mathcal{D}}(\text{cons}(\text{max}(x, y), \text{maxlist}(xs, ys))) \downarrow_{\text{maxlist}(x_1, x_2) \rightarrow \text{nil}} = \text{cons}(z, \text{nil})$ and $\text{nil} \simeq \text{cons}(z, \text{nil})$ is LIAC-unsatisfiable. \triangle

14.4 Complex Conjectures

In this section, a class of decidable conjectures that contains defined function symbols on both sides is identified. In order to decide the inductive validity of these conjectures, safe generalizations using the no-theory condition as introduced in Section 14.3 need to be utilized. Again, only single LIAC-based functions are considered initially and the case of LIAC-based sets of functions is considered afterwards.

In addition to the concept of compatibility as defined in Section 14.2, it becomes necessary to investigate how the left-hand sides of the rules defining a symbol from $\mathcal{D}(\mathcal{R})$ can be applied to the basic subterm of the conjecture using **Expand**. The *definition schemes* as defined below collect all substitutions σ that are generated using **Expand**. Recall that **Expand** computes the most general syntactic unifiers of left-hand sides and a basic subterm of the conjecture.

Definition 14.44 (Definition Schemes). *Let f be LIAC-good, let x^* consist of variables, and let $l \rightarrow r \in \mathcal{R}(f)$. Then the definition scheme of $f(x^*)$ for the rule $l \rightarrow r$ is given by $\mathcal{D}\text{ef}(f(x^*), l \rightarrow r) = \{\sigma \mid \sigma = \text{mgu}(f(x^*), l)\}$. Furthermore, let $\mathcal{D}\text{ef}(f(x^*)) = \bigcup_{l \rightarrow r \in \mathcal{R}(f)} \mathcal{D}\text{ef}(f(x^*), l \rightarrow r)$ and $\mathcal{D}\text{ef}\mathcal{R}(f(x^*)) = \{ \langle l \rightarrow r, \mathcal{D}\text{ef}(f(x^*), l \rightarrow r) \rangle \mid l \rightarrow r \in \mathcal{R}(f) \text{ and } \mathcal{D}\text{ef}(f(x^*), l \rightarrow r) \neq \emptyset \}$.*

Notice that $\mathcal{D}\text{ef}(f(x^*), l \rightarrow r)$ has cardinality one if $f(x^*)$ and l are unifiable and cardinality zero otherwise.

Example 14.45. Take the function `len` from Example 14.36 and add the rules defining `zip` from Example 14.16. Then, the term `len(zs)` has the definition scheme $\mathcal{D}ef(\mathit{len}(zs)) = \{\sigma_1, \sigma_2\}$ with $\sigma_1 = \{zs \mapsto \mathit{nil}\}$ and $\sigma_2 = \{zs \mapsto \mathit{cons}(x, xs)\}$. Furthermore, $\mathcal{D}ef(\mathit{zip}(zs, zs)) = \{\sigma'_1, \sigma'_2\}$ with $\sigma'_1 = \{zs \mapsto \mathit{nil}\}$ and $\sigma'_2 = \{zs \mapsto \mathit{cons}(x, xs), y \mapsto x, ys \mapsto xs\}$. \triangle

Related to definition schemes are *call schemes*. A call scheme collects the instantiations of $f(x^*)$ that are generated by `Expand`. Definition schemes and call scheme together contain roughly the same information as cover sets in the explicit induction framework [174]. Notice, however, that cover sets are only defined for terms $f(x^*)$ such that x^* consists of pairwise distinct variables.

Definition 14.46 (Call Schemes). *Let f be LIAC-based, let x^* consist of variables, and let $\langle l \rightarrow r, \{\sigma\} \rangle \in \mathcal{D}ef\mathcal{R}(f(x^*))$. Then the call scheme of $f(x^*)$ for the rule $l \rightarrow r$ and the substitution σ is given by $\mathcal{C}all(f(x^*), l \rightarrow r, \sigma) = \{\tau \mid f(x^*)\tau = f(t^*)\sigma \text{ where } f(t^*) \text{ is a subterm of } r\}$.*

Example 14.47. Continuing Example 14.45, $\mathcal{C}all(\mathit{len}(zs), \mathit{len}(\mathit{nil}) \rightarrow \mathcal{O}, \{zs \mapsto \mathit{nil}\}) = \emptyset$ and $\mathcal{C}all(\mathit{len}(zs), \mathit{len}(\mathit{cons}(x, xs)) \rightarrow \mathit{s}(\mathit{len}(xs)), \{zs \mapsto \mathit{cons}(x, xs)\}) = \{\tau\}$ with $\tau = \{zs \mapsto xs\}$. \triangle

Now a conjecture that contains defined symbols on both sides needs to satisfy various conditions. The first five conditions are nearly identical to the simple nested conjectures from Definition 14.21, but it now needs to be required that both sides of the conjecture have compatibility sequences. Conditions 5 and 6 are more complex. First, it is required that the definition schemes of the innermost basic terms on both sides of the conjecture coincide. The reason for this is that only the left-hand side of the conjecture is used for computing instantiations using `Expand`, and it needs to be ensured that the right-hand side can “handle” these substitutions, i.e., that the right-hand side can be simplified using rewriting. Finally, it needs to be ensured that either Theory'_1 or Theory'_2 can be applied after simplifying the instantiated conjecture. For

this, it has to be ensured that maximal subterms with a defined root symbol satisfy the no-theory condition since Theorem 14.33 can then be applied.

Definition 14.48 (Complex Conjectures). *A complex conjecture is an atomic conjecture of the form $D[f(x^*)] \equiv E[g(y^*)][\varphi]$ such that the following conditions are satisfied:*

1. $\mathcal{R} \cup \{D[f(x^*)] \rightarrow E[g(y^*)][\varphi]\}$ is terminating.
2. $\varphi = \top$
3. The terms $D[f(x^*)]$ and $E[g(y^*)]$ have compatibility sequences, f is LIAC-based, and g is LIAC-good.
4. x^*, y^* consist of variables.
5. Whenever $x_i = x_j$ for $i < j$, then $\langle i, j \rangle \in \text{ImpEq}(f)$.
6. $\text{Def}(f(x^*))$ and $\text{Def}(g(y^*))$ are identical up to variable-renamings if the substitutions are restricted to y^* .
7. For any $\langle l \rightarrow r, \{\sigma\} \rangle \in \text{Def}\mathcal{R}(f(x^*))$, the terms in $M(l \rightarrow r, \sigma)$ are pairwise variable-disjoint and satisfy the no-theory condition. Here, the set $M(l \rightarrow r, \sigma)$ consists of all maximal subterms with a defined root symbol⁴ of any term in

$$\begin{aligned} & \{E[g_j(s_j^*)]\sigma' \downarrow_{\mathcal{R}, \mathbb{Z}} \mid \langle g(s^*) \rightarrow C[g_1(s_1^*), \dots, g_n(s_n^*)], \{\sigma'\} \rangle \in \text{Def}\mathcal{R}(g(y^*)) \\ & \quad \text{and } 1 \leq j \leq n \text{ where } \sigma' \text{ "corresponds" to } \sigma\} \\ & \cup \{E[g(y^*)]\tau \downarrow_{\mathcal{R}, \mathbb{Z}} \mid \tau \in \text{Call}(f(x^*), l \rightarrow r, \sigma)\} \end{aligned}$$

Here, the substitution corresponding to σ is the substitution that is identical to σ up to a variable-renaming if restricted to y^* , cf. condition 6.

Example 14.49. Take the function `len` from Example 14.36 and add the rules defining `zip` from Example 14.16. Furthermore, add the following rules:

⁴For a term $C[t_1, \dots, t_n]$ such that C is over $\widehat{\mathcal{C}}(\mathcal{R})$ and $\text{root}(t_i) \in \text{Def}(\mathcal{R})$ for all $1 \leq i \leq n$, these terms are t_1, \dots, t_n .

$$\begin{aligned} \text{plen}(\text{pnil}) &\rightarrow \mathcal{O} \\ \text{plen}(\text{pcons}(x, xs)) &\rightarrow \text{s}(\text{plen}(xs)) \end{aligned}$$

Then, the conjecture $\text{plen}(\text{zip}(zs, zs)) \equiv \text{len}(zs)$ is complex. To see this, notice that plen is compatible with zip , that $\langle 1, 2 \rangle \in \text{ImpEq}(\text{zip})$, and that the definition schemes $\text{Def}(\text{zip}(zs, zs))$ and $\text{Def}(\text{len}(zs))$ are identical if the substitutions are restricted to zs , recall Example 14.45. Finally, notice that all terms in $M(\text{zip}(xs, \text{nil}) \rightarrow \text{pnil}, \sigma_1) = \emptyset$ and $M(\text{zip}(\text{cons}(x, xs), \text{cons}(y, ys)) \rightarrow \text{pcons}(\text{pair}(x, y), \text{zip}(xs, ys)), \sigma_2) = \{\text{len}(xs)\}$ are pairwise variable-disjoint and satisfy the no-theory condition. \triangle

Theorem 14.50. *Using the strategy $\text{Expand} \cdot \text{Simplify}^* \cdot (\text{Theory}'_1 + \text{Theory}'_2)^!$, it is decidable whether a complex conjecture is an inductive theorem.*

In order to extend complex conjectures from a single LIAC-based function to a LIAC-based set of functions, the notion a call schemes needs to be extended. Now, calls from one function to another function are considered. Notice that definition schemes do not need to be adapted since they only depend on the left-hand sides of rules.

Definition 14.51 (Call Schemes–Version 2). *Let f, g be members of LIAC-based set of functions, let x^*, y^* consist of variables, and let $\langle l \rightarrow r, \{\sigma\} \rangle \in \text{Def}\mathcal{R}(f(x^*))$. Then the call scheme of $f(x^*)$ to $g(y^*)$ for the rule $l \rightarrow r$ is given by $\text{Call}(f(x^*), g(y^*), l \rightarrow r, \sigma) = \{\tau \mid g(y^*)\tau = g(t^*)\sigma \text{ where } g(t^*) \text{ is a subterm of } r\}$.*

Example 14.52. Take the original definitions of `even` and `odd` from Example 14.39. Then $\text{Def}(\text{even}(z)) = \{\sigma_1, \sigma_2\}$, where $\sigma_1 = \{z \mapsto \mathcal{O}\}$ and $\sigma_2 = \{z \mapsto \text{s}(x)\}$. Also, $\text{Call}(\text{even}(z), \text{odd}(z), \text{even}(\text{s}(x)) \rightarrow \text{odd}(x), \sigma_2) = \{\tau\}$ with $\tau = \{z \mapsto x\}$. \triangle

Example 14.53. Take the functions `stitch` and `stitch'` from Example 14.24. For the third `stitch`-rule, $\text{Call}(\text{stitch}(zs, zs), \text{stitch}'(zs, zs), \dots, \{zs \mapsto \text{cons}(x, xs)\}) = \{\tau\}$ where $\tau = \{zs \mapsto xs\}$. \triangle

Complex conjectures for LIAC-based sets of functions are now the straightforward extension of Definition 14.48.

Definition 14.54 (Complex Conjectures–Version 2). *A complex conjecture is a set of atomic conjectures $\{D[f_1(x_1^*)] \equiv E_1[g_1(y_1^*)][[\varphi_1]], \dots, D[f_n(x_n^*)] \equiv E_n[g_n(y_n^*)][[\varphi_n]]\}$ such that the following conditions are satisfied:*

1. $\mathcal{R} \cup \{D[f_1(x_1^*)] \rightarrow E_1[g_1(y_1^*)][[\varphi_1]], \dots, D[f_n(x_n^*)] \rightarrow E_n[g_n(y_n^*)][[\varphi_n]]\}$ is terminating.
2. $\varphi_i = \top$ for all $1 \leq i \leq n$.
3. The terms $D[f_i(x_i^*)]$ and $E_i[g_i(y_i^*)]$ have compatibility sequences for all $1 \leq i \leq n$, the set $\mathcal{G} = \{f_1, \dots, f_n\}$ is LIAC-based, and g_1, \dots, g_n are LIAC-good.
4. x_i^*, y_i^* consist of variables for all $1 \leq i \leq n$.
5. Whenever $x_{k,i} = x_{k,j}$ for $i < j$, there exists an $\langle f_k, i, j, \Gamma \rangle \in \text{ImpEq}(\mathcal{G})$ such that $x_{k',i'} = x_{k',j'}$ for all $\langle f_{k'}, i', j' \rangle \in \Gamma$.
6. $\text{Def}(f_i(x_i^*))$ and $\text{Def}(g_i(y_i^*))$ are identical up to variable-renamings if the substitutions are restricted to y_i^* for all $1 \leq i \leq n$.
7. For all $1 \leq i \leq n$ and any $\langle l \rightarrow r, \{\sigma\} \rangle \in \text{Def}\mathcal{R}(f_i(x_i^*))$, the terms in $M_i(l \rightarrow r, \sigma)$ are pairwise variable-disjoint and satisfy the no-theory condition. Here, the set $M_i(l \rightarrow r, \sigma)$ consists of all maximal subterms with a defined root symbol of any term in

$$\begin{aligned} & \{E_i[h_j(s_j^*)]\sigma' \downarrow_{\mathcal{R}, \mathcal{Z}} \mid \langle g_i(s^*) \rightarrow C[h_1(s_1^*), \dots, h_m(s_m^*)], \{\sigma'\} \rangle \in \text{Def}\mathcal{R}(g_i(y_i^*)) \\ & \quad \text{and } 1 \leq j \leq m \text{ where } \sigma' \text{ “corresponds” to } \sigma\} \\ & \cup \bigcup_{j=1}^n \{E_j[g_j(y_j^*)]\tau \downarrow_{\mathcal{R}, \mathcal{Z}} \mid \tau \in \text{Call}(f_i(x_i^*), f_j(x_j^*), l \rightarrow r, \sigma)\} \end{aligned}$$

Example 14.55. Continuing Example 14.52, add the following rules defining not:

$$\begin{aligned} \text{not}(\text{true}) &\rightarrow \text{false} \\ \text{not}(\text{false}) &\rightarrow \text{true} \end{aligned}$$

Then the conjecture $\{\text{not}(\text{even}(z)) \equiv \text{odd}(z), \text{not}(\text{odd}(z)) \equiv \text{even}(z)\}$ is complex. For this, notice in particular that $\text{even}(z)$ and $\text{odd}(z)$ satisfy the no-theory condition. \triangle

Example 14.56. Continuing Example 14.53, add the rules for len from Example 14.36 and for plen from Example 14.49. Then, the conjecture $\{\text{plen}(\text{stitch}(zs, zs)) \equiv \text{len}(zs), \text{plen}(\text{stitch}'(zs, zs)) \equiv \text{len}(zs)\}$ is complex. \triangle

Theorem 14.57. *Using the strategy $\text{Expand}^* \cdot \text{Simplify}^* \cdot (\text{Theory}'_1 + \text{Theory}'_2)^!$, where Expand is applied once to each atomic conjecture of the set, it is decidable whether a complex conjecture is an inductive theorem.*

Example 14.58. All of the following conjectures are complex, but neither simple nor simple nested:

$$\begin{aligned} &\{\text{not}(\text{even}(x)) \equiv \text{odd}(x), \text{not}(\text{odd}(x)) \equiv \text{even}(x)\} \\ &\text{min}(x, y) \equiv \text{max}(x, y) \\ &\{\text{plen}(\text{stitch}(xs, xs)) = \text{len}(xs), \text{plen}(\text{stitch}'(xs, xs)) = \text{len}(xs)\} \\ &\text{plen}(\text{zip}(xs, xs)) \equiv \text{len}(xs) \end{aligned}$$

According to Theorems 14.50 and 14.57, it can be decided whether these conjectures are inductive theorems. \triangle

14.5 Implementation

The inductive proof method based on the inference systems \mathcal{I} and \mathcal{I}' has been implemented in the tool **Sail2**, the successor of **Sail** [65]. The implementation of the inference rules is straightforward. In order to check for termination of $\mathcal{R} \cup H$ in the

side condition of **Expand**, the implementation of the methods for proving termination of CERSs developed in this dissertation in the termination tool **AProVE** is used.

Furthermore, functions for checking whether a conjecture is simple, simple nested, or complex have been implemented in **Sail2**. In order to perform these checks as efficiently as possible, the following information is pre-computed while parsing the \mathbb{Z} -CERS:

1. The set $\text{ImpEq}(\mathcal{G})$ is computed for each LIAC-based set of functions \mathcal{G} . This requires calls to the external tool **CVC3** [24] in order to check for LIAC-satisfiability and LIAC-validity.
2. Information on the compatibility between function symbols is computed. This is done using rewriting with the \mathbb{Z} -CERS, and in order to ensure that this rewriting process is terminating it is first checked whether the \mathbb{Z} -CERS is terminating.
3. If all rules of the \mathbb{Z} -CERS are unconstrained, it is determined which defined functions satisfy the no-theory condition. This check is done using Theorem 14.35, Theorem 14.40, and Theorem 14.42 and might require further calls to **CVC3**.

Notice that definition schemes and call schemes are currently not computed at parse time. This would be possible, however, by considering all possible patterns of variables in Definition 14.44 and Definition 14.51.

14.6 Summary

This chapter has identified several classes of conjectures whose inductive validity can be decided using (an extension of) the inference system \mathcal{I} by the canonical strategy $\text{Expand}^* \cdot \text{Simplify}^* \cdot (\text{Theory}_1 + \text{Theory}_2)^!$. Decidability of inductive validity is obtained

by restricting the shape of the rewrite rules and requiring certain compatibility properties of the defined functions.

The classes of decidable conjectures introduced in this chapter are significant generalizations of the decidable conjectures considered in previous work [111, 80, 82, 104]. First, the rewrite rules that are admitted are less restricted since calls to auxiliary functions and even mutual recursive definitions are allowed. Additionally, the decidable conjectures may contain non-linear basic subterms. Finally, the tool **Sail2** provides the first implementation of decidable induction since the methods for checking whether a conjecture is decidable presented in [111, 80, 82, 104] have not been implemented. The implementation in **Sail2** has been successfully evaluated on a collection of examples. This evaluation confirms that checking whether the inductive validity of a conjecture is decidable is indeed much faster than attempting to prove or disprove it, cf. Chapter 15.

Chapter 15

Conclusions and Evaluation

This dissertation has presented an integration of natural numbers or integers and collection data structures such as sets or multisets into the term rewriting framework, resulting in *constrained equational rewrite systems (CERSs)*. In order to take full advantage of the pre-defined semantics of natural numbers or integers, the rewrite rules of a CERS are equipped with constraints from quantifier-free Presburger arithmetic and the rewrite relation of a CERS utilizes a decision procedure in order to check for validity of the instantiated constraints. Collection data structures are integrated in a way that closely corresponds to their intuitive semantics, and this approach results in a rewrite relation that is often terminating when a naive integration results in a non-terminating rewrite relation.

The main interest in this dissertation has been the development of methods for automatically proving termination of CERSs. To this extent, a dependency pair framework for CERSs has been developed in Chapter 5. Next, several termination techniques that can be applied within the dependency pair framework have been developed in Chapters 6–8. Implementation methods for the more complex termination techniques have been discussed in Chapter 9, and an empirical evaluation shows

that an implementation based on these methods in the termination tool AProVE is very efficient and powerful (cf. Section 15.1).

The class of CERSs has been extended in two orthogonal ways:

1. Conditional CERSs add the capability of having rewrite rules that are equipped with conditions in addition to constraints. In contrast to constraints, these conditions contain user-defined functions, which requires that the truth of the conditions has to be established by recursively rewriting them.

The inherently recursive nature of conditional rewriting requires a different notion of terminating, namely *operational termination*. It has been shown in Chapter 10 that operational termination of conditional CERSs can be reduced to termination of unconditional CERSs by a simple syntactic transformation. Thus, the methods for proving termination of CERSs can also be used for showing operational termination of conditional CERSs.

2. Using CERSs together with context-sensitive rewrite strategies provides a fine-grained control over the rewrite relation. This is often necessary in order to model the semantics of real-life programming languages more closely. For instance, context-sensitive rewriting is directly supported by Maude and makes it possible to model lazy evaluation as used in Haskell.

Using context-sensitive rewrite strategies makes reasoning about termination challenging. An adaptation of the method developed for CERSs to the context-sensitive case has been presented in Chapters 11 and 12.

These two orthogonal extensions can of course be combined as well.

In order to show the usefulness of CERSs in the context of showing partial correctness, an inductive theorem proving method for CERSs has been investigated in Chapter 13. This method provides a high degree of automation since it tightly couples inductive reasoning with a decision procedure.

Inductive reasoning does not provide a decision procedure in general since proof attempts may fail or diverge. Since guaranteed yes/no answers are needed in many cases, Chapter 14 has investigated cases where inductive reasoning *is* a decision procedure.

15.1 Empirical Evaluation

In order to assess the practical contributions of this dissertation, all methods presented in Chapters 5–14 have been implemented and evaluated.

15.1.1 Termination Analysis

The termination analysis techniques for CERSs and CS-CERSs as presented in Chapters 5–12 have been implemented in the termination tool AProVE [84], resulting in AProVE-CERS. In order to assess the power and efficiency of the approach, the implementation has been tested on a collection of 150 examples (110 non-context-sensitive examples and 40 context-sensitive examples). The detailed results can be found in Appendices B.1 and B.2. Furthermore, all examples, the detailed results, and all termination proofs produced by AProVE-CERS are available at <http://www.cs.unm.edu/~spf/tdps/>.

The examples were taken from various sources, including the *Termination Problem Data Base* [162], and have been suitably adapted to make use of built-in integers and collection data structures. Several of the examples have been obtained from functional Maude modules [43]. Of the non-context-sensitive examples, a total of 41 were obtained from imperative programs [44, 45, 145, 36, 37, 49, 50] using the translation presented in Chapter 4. Nine examples were obtained from conditional CERSs using the transformation presented in Chapter 10.

With a timeout of 60 seconds for each example, AProVE-CERS succeeds in proving termination of 140 examples (93.3%), taking an average time of about 2 seconds for one example. In particular, it succeeds on 36 (87.8%) of the examples obtained from imperative programs and on all nine examples obtained from conditional CERSs.

| Average Time | Success Rate |
|--------------|--------------|
| 2.15 sec | 93.33% |

An empirical comparison with AProVE-Integer based on the methods presented in [75, 143] has been conducted on a subset of 80 examples where the methods of [75, 143] are applicable (i.e., examples that use neither context-sensitive strategies nor collection data structures).¹ Out of these 80 examples, AProVE-CERS succeeds on 73, while AProVE-Integer succeeds on 72. There are examples that can only be handled by AProVE-CERS but not by AProVE-Integer, and vice versa. On examples that can be handled by both AProVE-CERS and AProVE-Integer, the system AProVE-CERS that is based on this dissertation is much faster than AProVE-Integer, on average by a factor of three (in the most extreme case, AProVE-CERS succeeds in 0.1 sec while AProVE-Integer needs 52.7 sec in order to prove termination).

| AProVE-CERS | | AProVE-Integer | |
|--------------|--------------|----------------|--------------|
| Average Time | Success Rate | Average Time | Success Rate |
| 3.86 sec | 91.25% | 10.36 sec | 90.00% |

15.1.2 Inductive Theorem Proving

The inductive theorem proving method based on the inference systems \mathcal{I} and \mathcal{I}' as presented in Chapters 13 and 14 has been implemented in the tool Sail2, the

¹Notice, however, that AProVE-Integer is applicable to examples where AProVE-CERS is not applicable since AProVE-Integer supports multiplication of the built-in integers, whereas AProVE-CERS is restricted to addition and subtraction.

Chapter 15. Conclusions and Evaluation

successor of *Sail* [65]. Furthermore, functions for checking whether a conjecture is simple, simple nested, or complex as defined in Chapter 14 have been implemented in *Sail2* as well.

The implementation has been tested on 28 examples of decidable conjectures, and the time spend for checking whether a conjecture is indeed decidable as well as the time needed for (dis-)proving it have been recorded. Recall that a proof attempt requires a call to *AProVE-CERS* in order to determine whether the \mathbb{Z} -CERS together with the oriented conjectures is terminating. Also recall that a proof attempt requires calls the external SMT-solver *CVC3* in order to check for *LIAC*-validity and $\mathcal{Th}_{\mathbb{Z}}$ -validity. The following table contains average times in milliseconds. The detailed results can be found in Appendix B.3. Furthermore, all examples, the detailed results and all proofs produced by *Sail2* are available at <http://www.cs.unm.edu/~spf/sail2/>.

| Checking | (Dis-)Proving | (Dis-)Proving w/o Time in CVC3 | Termination |
|------------|---------------|--------------------------------|--------------|
| 0.092 msec | 14.981 msec | 0.180 msec | 127.421 msec |

Notice that the most time-consuming part is the termination check using *AProVE-CERS*. Furthermore, checking whether a conjecture is a member of the class of decidable conjectures is orders of magnitude faster than deciding whether it is an inductive theorem. Notice that most of the time for the proof attempt is spent within the external tool *CVC3*. While the total time for the proof attempt can probably be shortened significantly by implementing a decision procedure for *LIAC*-validity and $\mathcal{Th}_{\mathbb{Z}}$ -validity from scratch since this would eliminates the overhead of calling an external tool, the remaining parts of the proof attempt still require nearly twice as much time as checking whether a conjecture is a member of the class of conjectures whose inductive validity is decidable.

15.2 Future Work

Several ideas on how the techniques and methods presented in this dissertation might be extended have already been mentioned in Chapters 3–14. These low-level ideas will not be repeated here. Instead, some high-level ideas and problems requiring future work will be outlined.

Having built-in integers and collection data structures is very useful for modeling programs written in `Maude` or `OCaml`, recall the examples in Chapter 1. Currently, the translation from `Maude` or `OCaml` into a CERS has to be performed by hand. While this is relatively straightforward for `Maude` since `Maude` is based on the term rewriting approach, it is more complicated for `OCaml`. With moderate implementation effort, an automatic translation should be possible. Indeed, the tool `MTT` [60] provides a translation from `Maude` modules into ordinary TRSs for the purpose of proving termination. CERSs seem to be a better candidate for this than ordinary TRSs, however, since CERSs support the equational attributes used in `Maude`.

Translating imperative programs into CERSs is highly non-trivial. Chapter 4 has shown how a simple class of imperative programs can be translated into CERSs. The class of imperative programs considered there could be extended relatively easily by allowing user-defined functions. Supporting dynamic data structures such as arrays or pointer-based lists or trees is more challenging. Initial progress has been reported in [155, 138], where it has been shown that a fragment of `Java` can be translated into ordinary TRSs. It should be possible to modify that translation to produce CERSs that can take advantage of the built-in integers.

The class of CERSs itself could also be extended. Notice that CERSs only support addition and subtraction of the built-in integers, but not multiplication. This is in contrast to the recent work in [75], which supports multiplication but does not support collection data structures and context-sensitive rewrite strategies. In prin-

ciple, CERSs can use integers with multiplication as the built-in theory. But then the problem of determining validity of constraints becomes undecidable in general, thus resulting in an undecidable rewrite relation. Notice, however, that the rewrite relation stays decidable on ground terms since validity of ground constraints can be determined by evaluating them. This is similar to [75], where the operations on integers are also restricted to the ground case. Some of the DP processors presented in this dissertation rely on the assumption that validity of (non-ground) constraints is decidable. While this is no longer true for integers with multiplication, there are nonetheless sound but incomplete methods for this which could be used.

With the increased popularity of automatic termination tools, the issue of trust becomes more and more important. Given a complex termination proof generated by one of these systems, can it be ensured that the proof is indeed correct? Formally verifying correctness of automatic termination tools is currently not possible due to their immense complexity. A promising approach to increase the trust in automatic termination tools has gained a lot of attention lately: *proof certification*. In proof certification, an automatically generated termination proof is checked for correctness by an interactive proof assistant for higher-order logic such as Coq [27] or Isabelle/HOL [135]. This approach is followed by CoLoR [28], A3PAT [47], and CeTA [163] for ordinary TRSs. Extending one of these systems to support CERSs and the methods developed in this dissertation is a worthwhile task.

Appendix A

Proofs

A.1 Proofs from Chapter 2

Proof of Lemma 2.21. Let $C[f(s^*)] \sim_{\mathcal{E}} t$, i.e., there exist terms t_0, \dots, t_n with $n \geq 0$ such that $C[f(s^*)] = t_0 \vdash_{\mathcal{E}} t_1 \vdash_{\mathcal{E}} \dots \vdash_{\mathcal{E}} t_n = t$. The claim is proved by induction on n . If $n = 0$ then $C[f(s^*)] = t$ and the claim is obvious.

If $n > 0$, the inductive assumption implies $t_{n-1} = C''[f(s''^*)]$ with $C'' \sim_{\mathcal{E}} C$ and $s''^* \sim_{\mathcal{E}} s^*$. Since $t_{n-1} \vdash_{\mathcal{E}} t_n$, there exists an equation $u \approx v$ (or $v \approx u$) in \mathcal{E} such that $t_{n-1}|_p = u\sigma$ and $t_n = t_{n-1}|_p[v\sigma]$ for some position p and some substitution σ . Let q be the position with $t_{n-1}|_q = f(s''^*)$, i.e., $C''|_q = \square$. Now, a case analysis on the relationship between the positions p and q is performed.

Case 1: $p \parallel q$. Then, $t_n = t_{n-1}|_p[v\sigma] = (C''[f(s''^*)])|_p[v\sigma] = (C''[v\sigma])|_p[f(s''^*)]$ with $C''[v\sigma] \sim_{\mathcal{E}} C''[u\sigma] = C''$.

Case 2: $p = q.q'$ for some position $q' \neq \Lambda$. Then it is the case that $t_n = t_{n-1}|_p[v\sigma] = (C''[f(s''^*)])|_p[v\sigma] = C''[f(s''^*)]|_p[v\sigma] = C''[f(s''^*)]|_{q.q'}[v\sigma] = C''[f(s''^*)]|_{q'}[v\sigma]$. Since $q' \neq \Lambda$, the position q' can be written as $q' = i.q''$ for some $i \in \mathbb{N}^+$ and some position q'' . Then $s'_j = s''_j$ if $i \neq j$ and

Appendix A. Proofs

$$s'_i = s''_i[v\sigma]_{q''} \sim_{\mathcal{E}} s''_i[u\sigma]_{q''} = s'', \text{ i.e., } s'^* \sim_{\mathcal{E}} s''^*.$$

Case 3: $q = p.p'$ for some position p' (possibly $p' = \Lambda$). Since $f \notin \mathcal{F}(\mathcal{E})$, the position p' can be written as $p' = p'_1.p'_2$ such that $u|_{p'_1}$ is a variable x and $x\sigma|_{p'_2} = f(s''^*)$. Since the equation $u \approx v$ (or $v \approx u$) is i.u.v., there exists a unique position p''_1 in v such that $v|_{p''_1} = x$. This implies $v\sigma|_{p''_1 p'_2} = x\sigma|_{p'_2} = f(s''^*)$. Define the substitution σ' by $\sigma'(y) = \sigma(y)$ for $y \neq x$ and $\sigma'(x) = \sigma(x)[\square]_{p'_2}$. Let $C' = (t_{n-1}[v\sigma]_p)[\square]_{pp'_1 p'_2} = t_{n-1}[v\sigma[\square]_{p''_1 p'_2}]_p = t_{n-1}[v\sigma']_p \sim_{\mathcal{E}} t_{n-1}[u\sigma']_p = C''$. Thus, $t_n = t_{n-1}[v\sigma]_p = C'[f(s''^*)]$ and the claim follows. \square

Proof of Lemma 2.22. It suffices to show that $\vdash_{\mathcal{E}_1} \circ \vdash_{\mathcal{E}_2} \subseteq \vdash_{\mathcal{E}_2} \circ \vdash_{\mathcal{E}_1}$. Thus, let $s \vdash_{\mathcal{E}_1} t \vdash_{\mathcal{E}_2} u$, i.e., there exist positions $p_1 \in \mathcal{Pos}(s)$ and $p_2 \in \mathcal{Pos}(t)$, equations $u_1 \approx v_1$ (or $v_1 \approx u_1$) in \mathcal{E}_1 and $u_2 \approx v_2$ (or $v_2 \approx u_2$) in \mathcal{E}_2 , and substitutions σ_1 and σ_2 such that

1. $s|_{p_1} = u_1\sigma_1$ and $t = s[v_1\sigma_1]_{p_1}$, and
2. $t|_{p_2} = u_2\sigma_2$ and $u = t[v_2\sigma_2]_{p_2}$.

$s \vdash_{\mathcal{E}_2} \circ \vdash_{\mathcal{E}_1} u$ is shown by a case analysis on the relationship between p_1 and p_2 .

Case 1: $p_1 \parallel p_2$. In this case $s \vdash_{\mathcal{E}_2} \circ \vdash_{\mathcal{E}_1} u$ is immediate.

Case 2: $p_2 = p_1.q$ for some position q (possibly $q = \Lambda$). Since v_1 does not contain symbols from \mathcal{E}_2 , there exists a position $q_1 \in \mathcal{Pos}(v_1)$ such that $v_1|_{q_1} = x$ is a variable and $q = q_1.q_2$ for some position q_2 . Define the substitution σ'_1 to behave like σ_1 , with the exception that $\sigma'_1(x) = \sigma_1(x)[v_2\sigma_2]_{q_2}$. Then $u_1\sigma_1 \vdash_{\mathcal{E}_2} u_1\sigma'_1$ since \mathcal{E}_1 is i.u.v. and thus $s|_{p_1} = u_1\sigma_1 \vdash_{\mathcal{E}_2} u_1\sigma'_1 \vdash_{\mathcal{E}_1} v_1\sigma'_1 = t|_{p_1}$, i.e., $s \vdash_{\mathcal{E}_2} \circ \vdash_{\mathcal{E}_1} t$.

Case 3: $p_1 = p_2.q$ for some position q . Analogous to the previous case. \square

Proof of Lemma 2.28. First, an easy induction shows that $\sim_{\mathcal{E}}$ commutes over $\rightarrow_{\mathcal{E} \setminus \mathcal{R}}^*$, i.e., $\sim_{\mathcal{E}} \circ \rightarrow_{\mathcal{E} \setminus \mathcal{R}}^* \subseteq \rightarrow_{\mathcal{E} \setminus \mathcal{R}}^* \circ \sim_{\mathcal{E}}$. Using this, $s \rightarrow_{\mathcal{E} \setminus \mathcal{R}}^! \hat{s}$ implies $t \rightarrow_{\mathcal{E} \setminus \mathcal{R}}^* t' \sim_{\mathcal{E}} \hat{s}$ for some t' since $t \sim_{\mathcal{E}} s$ and $s \rightarrow_{\mathcal{E} \setminus \mathcal{R}}^* \hat{s}$. If t' is reducible by $\rightarrow_{\mathcal{E} \setminus \mathcal{R}}$, then \hat{s} is reducible

Appendix A. Proofs

by $\rightarrow_{\mathcal{E}\setminus\mathcal{R}}$ as well since $\sim_{\mathcal{E}}$ commutes over $\rightarrow_{\mathcal{E}\setminus\mathcal{R}}$. Therefore, t' is irreducible by $\rightarrow_{\mathcal{E}\setminus\mathcal{R}}$ since \widehat{s} is irreducible by $\rightarrow_{\mathcal{E}\setminus\mathcal{R}}$. Thus $t \rightarrow_{\mathcal{E}\setminus\mathcal{S}}^! t'$ and $t \rightarrow_{\mathcal{E}\setminus\mathcal{S}}^! \widehat{t}$, and the confluence of $\rightarrow_{\mathcal{E}\setminus\mathcal{R}}$ implies $t' \sim_{\mathcal{E}} \widehat{t}$. Therefore $\widehat{s} \sim_{\mathcal{E}} \widehat{t}$. \square

Proof of Lemma 2.30. Let $s \rightarrow_{\mathcal{R}/\mathcal{E}} t$, i.e., $s \sim_{\mathcal{E}} \circ \rightarrow_{\mathcal{R}} \circ \sim_{\mathcal{E}} t$. Since $\rightarrow_{\mathcal{R}} \subseteq \rightarrow_{\mathcal{E}\setminus\mathcal{R}}$, the strong \mathcal{E} -coherence of $\rightarrow_{\mathcal{E}\setminus\mathcal{R}}$ implies $s \rightarrow_{\mathcal{E}\setminus\mathcal{R}} \circ \sim_{\mathcal{E}} t$. \square

Proof of Lemma 2.31. The proof heavily relies on results from [101, 99]. First, notice that strong \mathcal{E}_i -coherence for $i = 1, 2$ implies that $\rightarrow_{\mathcal{E}_i\setminus\mathcal{R}_i}$ is coherent modulo \mathcal{E}_i (using the terminology of [101]). Using [101, Theorem 5], $\rightarrow_{\mathcal{E}_i\setminus\mathcal{R}_i}$ is Church-Rosser modulo \mathcal{E}_i .¹ By [99, Theorem 3], Church-Rosser is a modular property, i.e., $\rightarrow_{\mathcal{E}_1\cup\mathcal{E}_2\setminus\mathcal{R}_1\cup\mathcal{R}_2}$ is Church-Rosser modulo $\mathcal{E}_1\cup\mathcal{E}_2$. Again by [101, Theorem 5] and since $\rightarrow_{\mathcal{E}_1\cup\mathcal{E}_2\setminus\mathcal{R}_1\cup\mathcal{R}_2}$ is terminating by assumption, $\rightarrow_{\mathcal{E}_1\cup\mathcal{E}_2\setminus\mathcal{R}_1\cup\mathcal{R}_2}$ is $\mathcal{E}_1\cup\mathcal{E}_2$ -convergent and coherent modulo $\mathcal{E}_1\cup\mathcal{E}_2$. Thus, it remains to be shown that $\rightarrow_{\mathcal{E}_1\cup\mathcal{E}_2\setminus\mathcal{R}_1\cup\mathcal{R}_2}$ is strongly $\mathcal{E}_1\cup\mathcal{E}_2$ -coherent.

For this, it is shown that $\vdash_{\mathcal{E}_1\cup\mathcal{E}_2} \circ \rightarrow_{\mathcal{E}_1\cup\mathcal{E}_2\setminus\mathcal{R}_1\cup\mathcal{R}_2} \subseteq \rightarrow_{\mathcal{E}_1\cup\mathcal{E}_2\setminus\mathcal{R}_1\cup\mathcal{R}_2} \circ \vdash_{\mathcal{E}_1\cup\mathcal{E}_2}^{\equiv}$. Thus, let $s \vdash_{\mathcal{E}_1\cup\mathcal{E}_2} t \rightarrow_{\mathcal{E}_1\cup\mathcal{E}_2\setminus\mathcal{R}_1\cup\mathcal{R}_2} u$, i.e., there exist positions $p_1 \in \mathcal{P}os(s)$ and $p_2 \in \mathcal{P}os(t)$, an equation $u_1 \approx v_1$ (or $v_1 \approx u_1$) in $\mathcal{E}_1\cup\mathcal{E}_2$, a rule $l_2 \rightarrow t_2 \in \mathcal{R}_1\cup\mathcal{R}_2$, and substitutions σ_1 and σ_2 such that

1. $s|_{p_1} = u_1\sigma_1$ and $t = s[v_1\sigma_1]_{p_1}$, and
2. $t|_{p_2} \sim_{\mathcal{E}_1\cup\mathcal{E}_2} l_2\sigma_2$ and $u = t[r_2\sigma_2]_{p_2}$.

Perform a case analysis on the relationship between p_1 and p_2 .

Case 1: $p_1 \parallel p_2$. In this case $s \rightarrow_{\mathcal{E}_1\cup\mathcal{E}_2\setminus\mathcal{R}_1\cup\mathcal{R}_2} \circ \vdash_{\mathcal{E}_1\cup\mathcal{E}_2} u$ is immediate.

Case 2: $p_1 = p_2.q$ for some position $q \neq \Lambda$. Then $s|_{p_2} \vdash_{\mathcal{E}_1\cup\mathcal{E}_2} t|_{p_2} \sim_{\mathcal{E}_1\cup\mathcal{E}_2} l_2\sigma_2$, i.e., $s \rightarrow_{\mathcal{E}_1\cup\mathcal{E}_2\setminus\mathcal{R}_1\cup\mathcal{R}_2} u$.

Case 3.1: $p_2 = p_1.q$ for some position q (possibly $q = \Lambda$) and $s \vdash_{\mathcal{E}_i} t \rightarrow_{\mathcal{E}_1\cup\mathcal{E}_2\setminus\mathcal{R}_{1-i}} u$.

¹I.e., $\leftrightarrow_{\mathcal{R}_i\cup\mathcal{E}_i}^* \subseteq \rightarrow_{\mathcal{E}_i\setminus\mathcal{R}_i} \circ \sim_{\mathcal{E}_i} \circ \leftarrow_{\mathcal{E}_i\setminus\mathcal{R}_i}$.

Appendix A. Proofs

Since $\mathcal{E}_1 \cup \mathcal{E}_2$ is collapse-free and v_1 does not contain symbols from \mathcal{E}_2 or \mathcal{R}_2 , there exists a position $q_1 \in \mathcal{Pos}(v_1)$ such that $v_1|_{q_1} = x$ is a variable and $q = q_1.q_2$ for some position q_2 . Define the substitution σ'_1 to behave like σ_1 , with the exception that $\sigma'_1(x) = \sigma_1(x)[r_2\sigma_2]_{q_2}$. Then $u_1\sigma_1 \rightarrow_{\mathcal{E}_1 \cup \mathcal{E}_2 \setminus \mathcal{R}_{1-i}} u_1\sigma'_1$ and thus $s|_{p_1} = u_1\sigma_1 \rightarrow_{\mathcal{E}_1 \cup \mathcal{E}_2 \setminus \mathcal{R}_{1-i}} u_1\sigma'_1 \vdash_{\mathcal{E}_i} v_1\sigma'_1 = t|_{p_1}$, i.e., $s \rightarrow_{\mathcal{E}_1 \cup \mathcal{E}_2 \setminus \mathcal{R}_{1-i}} \circ \vdash_{\mathcal{E}_i} t$.

Case 3.2: $p_2 = p_1.q$ for some position q (possibly $q = \Lambda$) and $s \vdash_{\mathcal{E}_i} t \rightarrow_{\mathcal{E}_1 \cup \mathcal{E}_2 \setminus \mathcal{R}_i} u$. Then $s \vdash_{\mathcal{E}_i} \circ \overset{\geq p_2}{\sim}_{\mathcal{E}_1 \cup \mathcal{E}_2} t[l_2\sigma_2]_{p_2} \rightarrow_{\mathcal{R}_i} t[r_2\sigma_2]_{p_2} = u$.² Using Lemma 2.22, $s \vdash_{\mathcal{E}_i} \circ \overset{\geq p_2}{\sim}_{\mathcal{E}_{1-i}} \circ \overset{\geq p_2}{\sim}_{\mathcal{E}_i} t[l_2\sigma_2]_{p_2} \rightarrow_{\mathcal{R}_i} t[r_2\sigma_2]_{p_2}$, i.e., $s \vdash_{\mathcal{E}_i} \circ \overset{\geq p_2}{\sim}_{\mathcal{E}_{1-i}} \circ \rightarrow_{\mathcal{E}_i \setminus \mathcal{R}_i} u$. Applying Lemma 2.22 again yields $s \sim_{\mathcal{E}_{1-i}} \circ \sim_{\mathcal{E}_i} \circ \rightarrow_{\mathcal{E}_i \setminus \mathcal{R}_i} u$. Now the strong \mathcal{E}_i -coherence of $\rightarrow_{\mathcal{E}_i \setminus \mathcal{R}_i}$ implies $s \sim_{\mathcal{E}_{1-i}} \circ \rightarrow_{\mathcal{E}_i \setminus \mathcal{R}_i} \circ \sim_{\mathcal{E}_i} u$ and thus $s \sim_{\mathcal{E}_{1-i}} \circ \rightarrow_{\mathcal{E}_1 \cup \mathcal{E}_2 \setminus \mathcal{R}_i} \circ \sim_{\mathcal{E}_1 \cup \mathcal{E}_2} u$. Using induction on the number of $\vdash_{\mathcal{E}_{1-i}}$ -steps in $\sim_{\mathcal{E}_{1-i}}$ and cases 1, 2 and 3.1 then implies $s \rightarrow_{\mathcal{E}_1 \cup \mathcal{E}_2 \setminus \mathcal{R}_i} \circ \sim_{\mathcal{E}_1 \cup \mathcal{E}_2} u$. \square

A.2 Proofs from Chapter 3

Proof of Lemma 3.20.

1. Since \mathcal{E} is size-preserving, the \mathcal{E} -equivalence classes are finite. Furthermore, the \mathcal{E} -equivalence class of a given term s can be effectively computed using the equations. In order to decide whether $s \sim_{\mathcal{E}} t$ holds true it then suffices to check whether t is in the \mathcal{E} -equivalence class of s .
2. It needs to be decided whether there exist a rewrite rule $l \rightarrow r \in \mathcal{S}$, a position $p \in \mathcal{Pos}(s)$, and a substitution σ such that $s|_p \sim_{\mathcal{E}} l\sigma$. Since \mathcal{S} and $\mathcal{Pos}(s)$ are finite, it suffices to consider a single rule and a single position, without loss of generality consider $p = \Lambda$. Thus, it needs to be decided whether there

²Here, the superscript $\geq p_2$ in $\overset{\geq p_2}{\sim}_{\mathcal{E}_1 \cup \mathcal{E}_2}$ denotes that equations are only applied at positions below p_2 .

Appendix A. Proofs

exists a substitution σ such that $s \sim_{\mathcal{E}} l\sigma$. In order to check this, it suffices to check whether there exists a term s' in the \mathcal{E} -equivalence class of s such that $s' = l\sigma$ for some substitution σ . But this is just syntactic matching, which is well-known to be decidable. Once such a substitution σ has been found it is easily possible to compute a term t with $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}} t$.

3. It needs to be decided whether there exist a constrained rewrite rule $l \rightarrow r[\![\varphi]\!] \in \mathcal{R}$, a position $p \in \mathcal{Pos}(s)$, and a \mathcal{Th} -based substitution σ such that $s|_p \xrightarrow{\geq \Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \overset{\geq \Lambda}{\sim}_{\mathcal{E}} l\sigma$ and $\varphi\sigma$ is \mathcal{Th} -valid. Since \mathcal{R} and $\mathcal{Pos}(s)$ are finite, it suffices to consider a single rule and a single position, without loss of generality consider $p = \Lambda$. Thus, it needs to be decided whether there exists a substitution σ such that $s \xrightarrow{\geq \Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \overset{\geq \Lambda}{\sim}_{\mathcal{E}} l\sigma$. First, notice that a term s' with $s \xrightarrow{\geq \Lambda!}_{\mathcal{E} \setminus \mathcal{S}} s'$ can be computed by Lemma 3.20.2. It then suffices to check whether there exists a term s'' in the \mathcal{E} -equivalence class of s' such that $s'' = l\sigma$ and $\varphi\sigma$ is \mathcal{Th} -valid for some \mathcal{Th} -based substitution σ . Candidate substitutions σ can be computed using syntactic matching, and it is easy to check whether such a substitution σ is \mathcal{Th} -based. \mathcal{Th} -validity of $\varphi\sigma$ can then be decided due to the assumption on \mathcal{Th} . Once such a substitution σ has been found it is easily possible to compute a term t with $s \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} t$. \square

Proof of Lemma 3.26.

1. Let $s' \sim_{\mathcal{E}} s \xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} t$. This means that $s = C[f(u^*)]$ for some context C with $f(u^*) \xrightarrow{\geq \Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \overset{\geq \Lambda}{\sim}_{\mathcal{E}} l\sigma$ for some constrained rewrite rule $l \rightarrow r[\![\varphi]\!] \in \mathcal{R}$ and some \mathcal{Th} -based substitution σ such that $\varphi\sigma$ is \mathcal{Th} -valid, all proper subterms of $f(u^*)$ are irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{Q}}$, and $t = C[r\sigma]$. Since $s \sim_{\mathcal{E}} s'$ and all equations in \mathcal{E} are i.u.v. and do not contain symbols from $\mathcal{D}(\mathcal{R})$, an application of Lemma 2.21 implies $s' = C'[f(u^*)]$ for some context C' with $C' \sim_{\mathcal{E}} C$ and $u^* \sim_{\mathcal{E}} u^*$. Therefore, $f(u^*) \xrightarrow{\geq \Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \overset{\geq \Lambda}{\sim}_{\mathcal{E}} l\sigma$ by Lemma 2.28 and σ can be used to rewrite $s' = C'[f(u^*)]$ to $t' = C'[r\sigma] \sim_{\mathcal{E}} C[r\sigma] = t$.

Appendix A. Proofs

2. Let $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}} t \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} u$, i.e., there exist positions $p_1 \in \mathcal{Pos}(s)$ and $p_2 \in \mathcal{Pos}(t)$, rules $l_1 \rightarrow r_1 \in \mathcal{S}$ and $l_2 \rightarrow r_2[\varphi_2] \in \mathcal{R}$, a substitution σ_1 , and a \mathcal{Th} -based substitution σ_2 such that

- (a) $s|_{p_1} \sim_{\mathcal{E}} l_1 \sigma_1$ and $t = s[r_1 \sigma_1]_{p_1}$, and
- (b) $t|_{p_2} \xrightarrow{>\Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \gtrsim_{\mathcal{E}}^{\Lambda} l_2 \sigma_2$, the instantiated \mathcal{Th} -constraint $\varphi_2 \sigma_2$ is \mathcal{Th} -valid, all proper subterms of $l \sigma_2$ are irreducible by $\xrightarrow{S}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{Q}}$, and $u = t[r_2 \sigma_2]_{p_2}$.

Next, a case analysis on the relationship between p_1 and p_2 is performed.

Case 1: $p_1 \parallel p_2$. In this case $s \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} \circ \rightarrow_{\mathcal{E} \setminus \mathcal{S}} u$ is immediate.

Case 2: $p_1 = p_2.q$ for some position $q \neq \Lambda$. In this case $s|_{p_2} = f(s^*)$, $t|_{p_2} = f(t^*)$, and $s^* \rightarrow_{\mathcal{E} \setminus \mathcal{S}} t^*$. Therefore, $f(s^*) \xrightarrow{>\Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \gtrsim_{\mathcal{E}}^{\Lambda} l_2 \sigma_2$ since $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ is \mathcal{E} -convergent and $f(t^*) \xrightarrow{>\Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \gtrsim_{\mathcal{E}}^{\Lambda} l_2 \sigma_2$. Thus, $s \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} s[r_2 \sigma_2]_{p_2} = t[r_2 \sigma_2]_{p_2} = u$.

Case 3: $p_2 = p_1.q$ for some position q , possibly $q = \Lambda$. Since r_1 does not contain symbols from $\mathcal{D}(\mathcal{R})$, there exists a position $q_1 \in \mathcal{Pos}(r_1)$ such that $r_1|_{q_1} = x$ is a variable and $q = q_1.q_2$ for some position q_2 . Define the substitution σ'_1 to behave like σ_1 , with the exception that $\sigma'_1(x) = \sigma_1(x)[r_2 \sigma_2]_{q_2}$. Then $l_1 \sigma_1 \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^+ l_1 \sigma'_1$ and thus $s|_{p_1} \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^+ \circ \sim_{\mathcal{E}} l_1 \sigma'_1$ by Lemma 3.26.1 since $s|_{p_1} \sim_{\mathcal{E}} l_1 \sigma_1$. Thus, $s|_{p_1} \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^+ \circ \rightarrow_{\mathcal{E} \setminus \mathcal{S}} r_1 \sigma'$. Since r_1 is linear, $s[r_1 \sigma'_1]_{p_1} = t[r_1 \sigma'_1]_{p_1} = u$ and thus $s \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^+ \circ \rightarrow_{\mathcal{E} \setminus \mathcal{S}} u$. \square

Proof of Corollary 3.27.

1. Assume that s starts an infinite $\xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reduction

$$s \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} s_1 \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} s_2 \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} s_3 \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} \dots$$

Using Lemma 3.26.1 this implies

$$t \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} t_1 \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} t_2 \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} t_3 \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} \dots$$

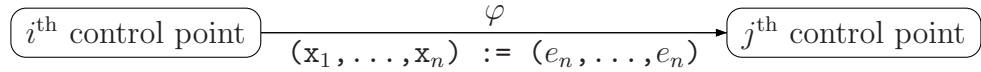
Appendix A. Proofs

where $s_i \sim_{\mathcal{E}} t_i$, i.e., t starts an infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}}$ -reduction as well. The other direction is shown the same way.

2. Let $s \rightarrow_{\mathcal{E}\setminus\mathcal{S}} t$ and assume that t starts an infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}}$ -reduction. Using $\rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}} \subseteq \xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}} \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ from Lemma 3.26.2 repeatedly produces an infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}}$ -reduction starting with s . \square

A.3 Proofs from Chapter 4

Proof sketch of Theorem 4.3. That the translation produces a CERS is immediate by inspection. For the second statement, consider the control flow graph associated with P , where each control point produces a node in the control flow graph. As in [145], the edges in the control flow graph are labeled by the parallel assignment executed during that transition in the program and by the condition obtained from the `while`-loop, `if`-, or `assume`-statement (if applicable). A typical transition thus has the following form:



It now suffices to notice that the translation produces the rewrite rule

$$\text{eval}_i(x_1, \dots, x_n) \rightarrow \text{eval}_j(e_1, \dots, e_n)[[\varphi]]$$

corresponding to this transition. \square

A.4 Proofs from Chapter 5

Proof of Theorem 5.6. Let $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be an RCERS.

Appendix A. Proofs

“ \Rightarrow ”: Assume that there exists a term t which starts an infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reduction. Then t contains a subterm $f_1(u_1^*)$ where $f_1(u_1^*)$ starts an infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reduction, but none of the terms in u_1^* starts an infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reduction. Consider an infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reduction starting with $f_1(u_1^*)$. First, the arguments u_1^* are reduced with $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ to terms v_1^* , and then a rewrite rule is applied to $f_1(v_1^*)$ at the root position, i.e., there exist a rule $l_1 \rightarrow r_1 \llbracket \varphi_1 \rrbracket$ in \mathcal{R} and a \mathcal{Th} -based substitution σ_1 such that $f_1(v_1^*) \xrightarrow{\geq \Lambda}_\mathcal{E} f_1(\widehat{v}_1^*) \gtrsim_\mathcal{E} l_1 \sigma_1$, the instantiated constraint $\varphi_1 \sigma$ is \mathcal{Th} -valid, and all proper subterms of $l_1 \sigma_1$ are irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{Q}}$. The reduction then yields $r_1 \sigma_1$ and the infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reduction continues with $r_1 \sigma_1$, i.e., the term $r_1 \sigma_1$ starts an infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reduction as well. So far the reduction of $f_1(u_1^*)$ has the following form:

$$f_1(u_1^*) \xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^* f_1(v_1^*) \xrightarrow{\geq \Lambda}_\mathcal{E} f_1(\widehat{v}_1^*) \gtrsim_\mathcal{E} l_1 \sigma_1 \rightarrow_{\mathcal{R}} r_1 \sigma_1$$

Here, the $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -steps in $f_1(u_1^*) \xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^* f_1(v_1^*)$ are applied strictly below the root position. By the definition of $\gtrsim_\mathcal{E}$, $l_1 = f_1(w_1^*)$ and $\widehat{v}_1^* \sim_\mathcal{E} w_1^* \sigma_1$, where the terms in $w_1^* \sigma_1$ are irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{Q}}$. Furthermore, the terms in $w_1^* \sigma_1$ do not start infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reductions by Corollary 3.27 since the terms in v_1^* do not start infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reductions.

Hence, for all variables x occurring in $f_1(w_1^*)$, the term $x \sigma_1$ does not start an infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reduction. Since $r_1 \sigma_1$ starts an infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reduction, there is a subterm $f_2(u_2^*)$ in r_1 such that $f_2(u_2^*) \sigma_1$ starts an infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reduction, whereas the terms in $u_2^* \sigma_1$ do not start infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reductions.

The first dependency pair in the infinite $(\text{DP}(\mathcal{R}), \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain that is going to be constructed is $f_1^\sharp(w_1^*) \rightarrow f_2^\sharp(u_2^*) \llbracket \varphi_1 \rrbracket$, obtained from the rewrite rule $l_1 \rightarrow r_1 \llbracket \varphi_1 \rrbracket$. The remaining dependency pairs of the infinite $(\text{DP}(\mathcal{R}), \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain are determined the same way: let $f_{i-1}^\sharp(w_{i-1}^*) \rightarrow f_i^\sharp(u_i^*) \llbracket \varphi_{i-1} \rrbracket$ be a dependency pair such that $f_i(u_i^*) \sigma_{i-1}$ starts an infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reduction

Appendix A. Proofs

but the terms in $u_i^* \sigma_{i-1}$ do not start infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reductions. Again, $u_i^* \sigma_{i-1}$ is first reduce using $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ to obtain v_i^* , and then a rewrite rule $f_i(w_i^*) \rightarrow r_i \llbracket \varphi_i \rrbracket$ is applied to $f(v_i^*)$ at the root position where $r_i \sigma_i$ starts an infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reduction for a \mathcal{Th} -based substitution σ_i with $\widehat{v}_i^* \sim_{\mathcal{E}} w_i^* \sigma_i$. As above, r_i contains a subterm $f_{i+1}(u_{i+1}^*)$ such that $f_{i+1}(u_{i+1}^*) \sigma_i$ starts an infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reduction, whereas the terms in $u_{i+1}^* \sigma_i$ do not start infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reductions. This produces the i^{th} dependency pair $f_i^\sharp(w_i^*) \rightarrow f_{i+1}^\sharp(u_{i+1}^*) \llbracket \varphi_i \rrbracket$. In this way, the infinite sequence

$$f_1^\sharp(w_1^*) \rightarrow f_2^\sharp(u_2^*) \llbracket \varphi_1 \rrbracket, f_2^\sharp(w_2^*) \rightarrow f_3^\sharp(u_3^*) \llbracket \varphi_2 \rrbracket, f_3^\sharp(w_3^*) \rightarrow f_4^\sharp(u_4^*) \llbracket \varphi_3 \rrbracket, \dots$$

is obtained and it remains to be shown that it is a minimal $(\text{DP}(\mathcal{R}), \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain. For this, notice that $f_{i+1}^\sharp(u_{i+1}^*) \sigma_i \xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} f_{i+1}^\sharp(v_{i+1}^*) \xrightarrow{>\Lambda}_{\mathcal{E} \setminus \mathcal{S}} f_{i+1}^\sharp(\widehat{v}_{i+1}^*) \gtrsim_{\mathcal{E}}^{\Lambda} f_{i+1}^\sharp(w_{i+1}^*) \sigma_{i+1}$ and $\varphi_i \sigma_i$ is \mathcal{Th} -valid for all $i \geq 1$. Furthermore, $f_i^\sharp(w_i^*) \sigma_i$ is irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{Q}}$ and $\xrightarrow{>\Lambda}_{\mathcal{E} \setminus \mathcal{S}}$. Since it is assumed that different (occurrences of) dependency pairs are variable-disjoint, it is possible to obtain a single \mathcal{Th} -based substitution $\sigma = \sigma_1 \cup \sigma_2 \cup \dots$ with $f_{i+1}^\sharp(u_{i+1}^*) \sigma \xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} \circ \xrightarrow{>\Lambda}_{\mathcal{E} \setminus \mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} f_{i+1}^\sharp(w_{i+1}^*) \sigma$ such that $\varphi_i \sigma$ is \mathcal{Th} -valid and $f_i^\sharp(w_i^*) \sigma$ is irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{Q}}$ and $\xrightarrow{>\Lambda}_{\mathcal{E} \setminus \mathcal{S}}$ for all $i \geq 1$. The chain is minimal by construction.

“ \Leftarrow ”: Assume that there exists an infinite minimal $(\text{DP}(\mathcal{R}), \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain

$$f_1^\sharp(w_1^*) \rightarrow f_2^\sharp(u_2^*) \llbracket \varphi_1 \rrbracket, f_2^\sharp(w_2^*) \rightarrow f_3^\sharp(u_3^*) \llbracket \varphi_2 \rrbracket, \dots$$

Hence, there is a \mathcal{Th} -based substitution σ such that

$$\begin{aligned} f_2^\sharp(u_2^*) \sigma &\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} \circ \xrightarrow{>\Lambda}_{\mathcal{E} \setminus \mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} f_2^\sharp(w_2^*) \sigma \\ f_3^\sharp(u_2^*) \sigma &\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} \circ \xrightarrow{>\Lambda}_{\mathcal{E} \setminus \mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} f_3^\sharp(w_3^*) \sigma \\ &\vdots \end{aligned}$$

where $f_i^\sharp(w_i^*) \sigma$ is irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{Q}}$ and $\varphi_i \sigma$ is \mathcal{Th} -valid for all $i \geq 1$.

Appendix A. Proofs

Notice that every dependency pair $f_i^\sharp(w_i^*) \rightarrow f_{i+1}^\sharp(u_{i+1})\llbracket\varphi_i\rrbracket$ corresponds to a rule $f_i(w_i^*) \rightarrow C_i[f_{i+1}(u_{i+1}^*)]\llbracket\varphi_i\rrbracket \in \mathcal{R}$ for some context C_i . Therefore, the infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}$ -reduction

$$\begin{array}{ccc} f_1(w_1^*)\sigma & \xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} & C_1[f_2(u_2^*)]\sigma \\ & \xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}}^* \circ \sim_{\mathcal{E}} & C_1[f_2(w_2^*)]\sigma \\ & \xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} & C_1[C_2[f_3(u_3^*)]]\sigma \\ & \vdots & \end{array}$$

is obtained and $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}$ is thus not terminating. \square

A.5 Proofs from Chapter 6

Proof of Theorem 6.1. Let $s_1 \rightarrow t_1\llbracket\varphi_1\rrbracket, s_2 \rightarrow t_2\llbracket\varphi_2\rrbracket, \dots$ be an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain. Thus, there exists a Th -based substitution σ such that $\varphi_1\sigma, \varphi_2\sigma, \dots$ are Th -valid. In particular, $\varphi_1, \varphi_2, \dots$ are Th -satisfiable and the dependency pairs in the chain thus cannot be from \mathcal{P}' .

Similarly, a reduction $s_i\sigma \xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}^* \circ \xrightarrow{>\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \circ \xrightarrow{\geq\Lambda}_{\mathcal{E}} t_{i+1}\sigma$ can only use rules $l \rightarrow r\llbracket\varphi\rrbracket$ for which φ is Th -satisfiable, i.e., rules from \mathcal{R}' cannot be applied.

Finally, a term is irreducible by $\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{Q}}$ iff it is irreducible by $\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{Q}-\mathcal{Q}'}$. Thus, there also exists an infinite minimal $(\mathcal{P} - \mathcal{P}', \mathcal{Q} - \mathcal{Q}', \mathcal{R} - \mathcal{R}', \mathcal{S}, \mathcal{E})$ -chain. \square

Proof of Theorem 6.3. Let $s_1 \rightarrow t_1\llbracket\varphi_1\rrbracket, s_2 \rightarrow t_2\llbracket\varphi_2\rrbracket, \dots$ be an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain using the Th -based substitution σ . First, assume that the infinite minimal chain contains a dependency pair $s \rightarrow t\llbracket\varphi\rrbracket$ from \mathcal{P}' . Since s is reducible by $\xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}}$, there exists a rule $l \rightarrow r \in \mathcal{S}$ such that $s|_p \sim_{\mathcal{E}} l\tau$ for some non-root position $p \in \mathcal{Pos}(s)$ and some substitution τ . Because $s\sigma|_p = s|_p\sigma \sim_{\mathcal{E}} l\tau\sigma$, the term $s\sigma$ is reducible by $\xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}}$, contradicting Definition 5.3.

Appendix A. Proofs

Similarly, assume that some reduction $s_i\sigma \xrightarrow{S, Q}^*_{Th\|\mathcal{E}\setminus\mathcal{R}} u \xrightarrow{>\Lambda}_!_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} t_{i+1}\sigma$ uses a rule $l \rightarrow r[\varphi]$ from \mathcal{R}' , i.e., let $u \xrightarrow{S, Q}_{Th\|\mathcal{E}\setminus\mathcal{R}} v$ for some terms u, v using the rule $l \rightarrow r[\varphi]$. Therefore, $u|_p \xrightarrow{>\Lambda}_!_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} l\tau$ for some non-root position $p \in \mathcal{Pos}(u)$ and some Th -based substitution τ . Since $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ is strongly \mathcal{E} -coherent by Definition 3.12, this means that $l\tau$ is irreducible by $\xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}}$. Now, since l is reducible by $\xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}}$, there exists a rule $l' \rightarrow r' \in \mathcal{S}$ such that $l|_{p'} \sim_{\mathcal{E}} l'\mu$ for some non-root position $p' \in \mathcal{Pos}(l)$ and some substitution μ . Since $l\tau|_{p'} = l|_{p'}\tau \sim_{\mathcal{E}} l'\mu\tau$, the term $l\tau$ is reducible by $\xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}}$, contradicting the fact that $l\tau$ is irreducible by $\xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}}$.

Finally, a term is irreducible by $\xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{Q}}$ iff it is irreducible by $\xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{Q}-\mathcal{Q}'}$. Thus, there also exists an infinite minimal $(\mathcal{P} - \mathcal{P}', \mathcal{Q} - \mathcal{Q}', \mathcal{R} - \mathcal{R}', \mathcal{S}, \mathcal{E})$ -chain. \square

Proof of Theorem 6.7. After a finite number of dependency pairs in the beginning, any infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain only contains pairs from some SCC. Hence, every infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain gives rise to an infinite minimal $(\mathcal{P}_i, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain for some $1 \leq i \leq n$. \square

Proof of Theorem 6.11. It needs to be shown that there exists a substitution μ that is Th -based for $\mathcal{V}(s_1) \cup \mathcal{V}(s_2)$ such that $\text{CAP}(t_1)\mu \xrightarrow{>\Lambda}_!_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} s_2\mu$, the constraints $\varphi_1\mu$ and $\varphi_2\mu$ are Th -valid, and $s_1\mu$ and $s_2\mu$ are irreducible by $\xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{Q}}$ and $\xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}}$ whenever $s_1 \rightarrow t_1[\varphi_1]$, $s_2 \rightarrow t_2[\varphi_2]$ is a $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain, i.e.,

$$\begin{aligned}
 & t_1\sigma \xrightarrow{S, Q}^*_{Th\|\mathcal{E}\setminus\mathcal{R}} u \xrightarrow{>\Lambda}_!_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} s_2\sigma \text{ for a } Th\text{-based substitution } \sigma \text{ such that} \\
 & \varphi_1\sigma \text{ and } \varphi_2\sigma \text{ are } Th\text{-valid and } s_1\sigma \text{ and } s_2\sigma \text{ are irreducible by } \xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{Q}} \text{ and} \\
 (\dagger) \quad & \xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}} \text{ implies } \text{CAP}(t_1)\mu \xrightarrow{>\Lambda}_!_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} s_2\mu \text{ for some substitution } \mu \text{ that is} \\
 & Th\text{-based for } \mathcal{V}(s_1) \cup \mathcal{V}(s_2) \text{ such that } \varphi_1\mu \text{ and } \varphi_2\mu \text{ are } Th\text{-valid and } s_1\mu \text{ and} \\
 & s_2\mu \text{ are irreducible by } \xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{Q}} \text{ and } \xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}}.
 \end{aligned}$$

In order to show (\dagger) , it is first shown that for all terms t and all substitutions μ that are Th -based for $\mathcal{V}(t)$ and where $\mu(x)$ is irreducible by $\xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{Q}}$ for all $x \in \mathcal{V}(t)$,

Appendix A. Proofs

$\text{CAP}(t)\mu \xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} u$ implies that there exists a substitution τ that is \mathcal{Th} -
 (‡) based for $\mathcal{V}(t)$ such that $u = \text{CAP}(t)\tau$, where μ and τ differ at most for the
 fresh variables introduced by $\text{CAP}(t)$.

The property (‡) is shown by induction on t . If $\text{CAP}(t) \in \mathcal{V}$, then it is a fresh variable y of sort `univ` since μ is \mathcal{Th} -based for $\mathcal{V}(t)$ and $\mu(x)$ is irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{Q}}$ for all $x \in \mathcal{V}(t)$. Then, letting $\tau = \{y \mapsto u\}$ establishes (‡).

Otherwise, $t = f(t_1, \dots, t_n)$, $\text{CAP}(t) = f(\text{CAP}(t_1), \dots, \text{CAP}(t_n))$, and there is no rule $l \rightarrow r \llbracket \varphi \rrbracket \in \mathcal{R}$ such that $f(\text{CAP}(t_1), \dots, \text{CAP}(t_n))\vartheta \xrightarrow{>\Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} l\vartheta$ for a substitution ϑ that is \mathcal{Th} -based for $\mathcal{V}(f(t_1, \dots, t_n)) \cup \mathcal{V}(l)$ such that $\varphi\vartheta$ is \mathcal{Th} -valid and all proper subterms of $l\vartheta$ are irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{Q}}$. First, it is shown that the reduction $f(\text{CAP}(t_1), \dots, \text{CAP}(t_n))\mu \xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} u$ cannot take place at the root position. If the reduction takes place at the root position, then there exist a rule $l \rightarrow r \llbracket \varphi \rrbracket \in \mathcal{R}$ and a \mathcal{Th} -based substitution ρ such that $f(\text{CAP}(t_1), \dots, \text{CAP}(t_n))\mu \xrightarrow{>\Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} l\rho$, the instantiated constraint $\varphi\rho$ is \mathcal{Th} -valid, and all proper subterms of $l\rho$ are irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{Q}}$. Since it can be assumed that l is variable-disjoint from $f(\text{CAP}(t_1), \dots, \text{CAP}(t_n))$, define the substitution $\vartheta = \mu \cup \rho$ which is \mathcal{Th} -based for $\mathcal{V}(f(t_1, \dots, t_n)) \cup \mathcal{V}(l)$. Since the instantiated constraint $\varphi\vartheta$ is \mathcal{Th} -valid and all proper subterms of $l\vartheta$ are irreducible by $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{Q}}$ this is a contradiction to the assumption. Hence, the $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -step does not take place at the root position, i.e., there exists an $1 \leq i \leq n$ with $\text{CAP}(t_i)\mu \xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} u_i$ and $u = f(\text{CAP}(t_1)\mu, \dots, u_i, \dots, \text{CAP}(t_n)\mu)$. By the inductive assumption, this yields a substitution δ that is \mathcal{Th} -based for $\mathcal{V}(t_i)$ such that $u_i = \text{CAP}(t_i)\delta$. Since the fresh variables introduced by $\text{CAP}(t_i)$ are disjoint from the fresh variables introduced by $\text{CAP}(t_j)$ for $1 \leq j \neq i \leq n$ and since μ and δ differ at most for the fresh variables introduced by $\text{CAP}(t_i)$, define the substitution τ with $\tau(x) = \delta(x)$ if x is a fresh variable introduced by $\text{CAP}(t_i)$ and $\tau(x) = \mu(x)$ otherwise. Then τ is \mathcal{Th} -based for $\mathcal{V}(t)$ and

$$u = f(\text{CAP}(t_1)\mu, \dots, \text{CAP}(t_i)\delta, \dots, \text{CAP}(t_n)\mu)$$

Appendix A. Proofs

$$\begin{aligned}
&= f(\text{CAP}(t_1)\tau, \dots, \text{CAP}(t_i)\tau, \dots, \text{CAP}(t_n)\tau) \\
&= \text{CAP}(t)\tau
\end{aligned}$$

By (‡) and an induction on the number of $\xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -steps, $\text{CAP}(t)\mu \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^* u$ for a substitution μ that is \mathcal{Th} -based for $\mathcal{V}(t)$ implies $u = \text{CAP}(t)\delta$ for some substitution δ that is \mathcal{Th} -based for $\mathcal{V}(t)$ such that μ and δ differ at most for the fresh variables introduced by $\text{CAP}(t)$. Since $t_1 = \text{CAP}(t_1)\sigma'$ for some substitution σ' that only instantiates fresh variables introduced by $\text{CAP}(t)$, in particular $t_1\sigma = \text{CAP}(t_1)\sigma'\sigma \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^* u$ implies $u = \text{CAP}(t_1)\delta$ for some substitution δ that is \mathcal{Th} -based for $\mathcal{V}(t_1)$ such that $\sigma'\sigma$ and δ differ at most for the fresh variables introduced by $\text{CAP}(t_1)$. Thus $\text{CAP}(t_1)\delta = u \xrightarrow{>\Lambda}_!_{\mathcal{E} \setminus \mathcal{S}} \circ \xrightarrow{>\Lambda}_{\mathcal{E}} s_2\sigma$ since $u \xrightarrow{>\Lambda}_!_{\mathcal{E} \setminus \mathcal{S}} \circ \xrightarrow{>\Lambda}_{\mathcal{E}} s_2\sigma$. Now define μ by $\mu(x) = \delta(x)$ if x is a fresh variable introduced by $\text{CAP}(t_1)$ and $\mu(x) = \sigma(x)$ otherwise. Notice that μ is \mathcal{Th} -based for $\mathcal{V}(s_1) \cup \mathcal{V}(s_2)$. Then $\text{CAP}(t_1)\mu = u \xrightarrow{>\Lambda}_!_{\mathcal{E} \setminus \mathcal{S}} \circ \xrightarrow{>\Lambda}_{\mathcal{E}} s_2\sigma = s_2\mu$. Since $s_1\mu = s_1\sigma$ and $s_2\mu = s_2\sigma$, the terms $s_1\mu$ and $s_2\mu$ are irreducible by $\xrightarrow{S}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{Q}}$ and $\xrightarrow{>\Lambda}_{\mathcal{E} \setminus \mathcal{S}}$ by Definition 5.3. Also, $\varphi_1\mu = \varphi_1\sigma$ and $\varphi_2\mu = \varphi_2\sigma$ are \mathcal{Th} -valid. \square

Proof of Theorem 6.12. In the second case soundness is obvious. Otherwise, assume that there is an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain $s_1 \rightarrow t_1[\varphi_1], s_2 \rightarrow t_2[\varphi_2], \dots$ using the \mathcal{Th} -based substitution σ . It needs to be shown that every occurrence of (a variable renamed version of) $s \rightarrow t[\varphi]$ in this chain can be replaced by $s \rightarrow t'[\varphi]$. Thus, let $t\sigma \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^* u \xrightarrow{>\Lambda}_!_{\mathcal{E} \setminus \mathcal{S}} v \xrightarrow{>\Lambda}_{\mathcal{E}} s_i\sigma$ for some $i > 1$.

First, consider the case where no $\xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -step takes place above the position p considered in Theorem 6.12. Then $t|_p\sigma = \text{CAP}(t|_p)\tau\sigma \xrightarrow{S, \mathcal{Q}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^* u|_p$. From (‡) in the proof of Theorem 6.11, $u|_p = \text{CAP}(t|_p)\mu$ for some substitution μ that is \mathcal{Th} -based for $\mathcal{V}(t|_p)$, i.e., the reductions take place in $\tau\sigma$. Let $\text{CAP}(t|_p) \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \hat{t}$, i.e., there is a rule $l \rightarrow r \in \mathcal{S}$, a position $q \in \text{Pos}(\text{CAP}(t|_p))$, and a substitution ρ such that $\text{CAP}(t|_p)|_q \sim_{\mathcal{E}} l\rho$ and $\hat{t} = t|_p[r\rho]_q$. Then $u|_{pq} = u|_p|_q = \text{CAP}(t|_p)\mu|_q = \text{CAP}(t|_p)|_q\mu \sim_{\mathcal{E}} l\rho\mu$, i.e., $u|_p = \text{CAP}(t|_p)\mu \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \hat{t}\mu$. Thus, $u[\hat{t}\mu]_p \xrightarrow{>\Lambda}_!_{\mathcal{E} \setminus \mathcal{S}} \circ \xrightarrow{>\Lambda}_{\mathcal{E}} v$ since $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ is \mathcal{E} -

Appendix A. Proofs

convergent. Hence, $t'\sigma = t[\widehat{t\tau}]_p\sigma = t\sigma[\widehat{t\tau}\sigma]_p \xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}^* u[\widehat{t\mu}]_p \xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} s_i\sigma$ and $s \rightarrow t[[\varphi]]$ in the above chain can be replaced by $s \rightarrow t'[[\varphi]]$.

If an $\xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}$ -step takes place above the position p , then the proof is similar but the rule $l \rightarrow r[[\varphi]]$ from \mathcal{R} that is used in the first reduction occurring above p takes the place of the next dependency pair $s_i \rightarrow t_i[[\varphi_i]]$. \square

Proof of Theorem 6.15. In the second case soundness is obvious. Otherwise, it needs to be shown that every occurrence of (a variable renamed version of) $s \rightarrow t[[\varphi]]$ and the dependency pair following it in an infinite chain can be replaced by some dependency pair from \mathcal{P}' . Thus, assume some infinite chain contains $\dots, s \rightarrow t[[\varphi]], s' \rightarrow t'[[\varphi']], v \rightarrow w[[\psi]], \dots$

Let the infinite chain use the substitution σ , i.e., $t\sigma \xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}^* \circ \xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} s'\sigma$, $t'\sigma \xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}^* \circ \xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} v\sigma$, and $\varphi\sigma$ and $\varphi'\sigma$ are Th -valid. Since σ is Th -based, the condition on t implies $t\sigma \xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} s'\sigma$, where furthermore only rules from $\mathcal{S}_{\text{base}} = \{l \rightarrow r \in \mathcal{S} \mid \text{sort}(l) = \text{base}\}$ and equations from $\mathcal{E}_{\text{base}} = \{u \approx v \in \mathcal{E} \mid \text{sort}(u) = \text{base}\}$ are used. Since $s'\tau = t$, this implies $s'\tau\sigma \xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} s'\sigma$, and the condition on s' implies that all reductions take place within the substitution, i.e., $x\tau\sigma \rightarrow_{\mathcal{E}\setminus\mathcal{S}}^! \circ \sim_{\mathcal{E}} x\sigma$ for all variables $x \in \mathcal{V}(s')$. But then $t'\tau\sigma \xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}} \circ \sim_{\mathcal{E}} t'\sigma$ as well and therefore $t'\tau\sigma \xrightarrow{S, \mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}^* \circ \xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} v\sigma$ by Lemmas 3.26 and 2.28. Furthermore, $\varphi'\tau\sigma$ is still Th -valid since application of rules from $\mathcal{S}_{\text{base}}$ and equations from $\mathcal{E}_{\text{base}}$ does not influence Th -validity. \square

Proof of Theorem 6.17. In the second case soundness is obvious. Otherwise, it needs to be shown that every infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain can be converted into an infinite minimal $(\pi(\mathcal{P}), \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain. Thus, assume that $s_1 \rightarrow t_1[[\varphi_1]], s_2 \rightarrow t_2[[\varphi_2]], \dots$ is an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain using the $Th_{\mathbb{Z}}$ -based substitution σ , i.e., $t_i\sigma \xrightarrow{S, \mathcal{Q}}_{Th_{\mathbb{Z}}\|\mathcal{E}\setminus\mathcal{R}}^* \circ \xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} s_{i+1}\sigma$ and $\varphi_i\sigma$ is $Th_{\mathbb{Z}}$ -valid for all $i \geq 1$. Then $\pi(t_i\sigma) \xrightarrow{S, \mathcal{Q}}_{Th_{\mathbb{Z}}\|\mathcal{E}\setminus\mathcal{R}}^* \circ \xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} \pi(s_{i+1}\sigma)$ since all reductions take

Appendix A. Proofs

place below the root. The minimality of the $(\pi(\mathcal{P}), \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain obtained this way immediately follows from the minimality of the initial $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain. \square

Proof of Lemma 6.23. Let $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be an RCERS such that \mathcal{E} is size-preserving.

1. Since \mathcal{E} is size-preserving, the \mathcal{E} -equivalence classes are finite. Furthermore, the \mathcal{E} -equivalence class of a given term s can be effectively computed using the equations. In order to decide whether $s \triangleright_{\mathcal{E}} t$ it then suffices to check whether there exist a term s' in the \mathcal{E} -equivalence class of s and a term t' in the \mathcal{E} -equivalence class of t such that $s' \triangleright t'$. For this, recall that the syntactic subterm relation is easily decidable. $s \succeq_{\mathcal{E}} t$ is decided in the same way.
2. Well-foundedness of $\triangleright_{\mathcal{E}}$ is immediate once it has been shown that $s \triangleright_{\mathcal{E}} t$ implies $|s| > |t|$. Thus, let $s \triangleright_{\mathcal{E}} t$, i.e., $s \sim_{\mathcal{E}} s' \triangleright t' \sim_{\mathcal{E}} t$ for some terms s', t' .

First, it is shown that $s \sim_{\mathcal{E}} s'$ implies $|s| = |s'|$. For this, an induction on n in $s \vdash_{\mathcal{E}}^n s'$ is performed. If $n = 0$, then the claim is obvious. Otherwise, $s \vdash_{\mathcal{E}}^{n-1} s'' \vdash_{\mathcal{E}} s'$, and the inductive assumption implies $|s| = |s''|$. Now $s'' \vdash_{\mathcal{E}} s'$ implies that there exists an equation $u \approx v$ (or $v \approx u$) in \mathcal{E} such that $s'' = C[u\sigma]$ and $s' = C[v\sigma]$ for some context C and some substitution σ and it thus suffices to show $|u\sigma| = |v\sigma|$. But $|u\sigma| = |v\sigma|$ is an immediate consequence of the assumption that the equation $u \approx v$ (or $v \approx u$) is size-preserving and i.u.v.

Now $|s| > |t|$ is easily obtained since $s' \triangleright t'$ implies $|s'| > |t'|$.

3. For $\triangleright_{\mathcal{E}}$, let $s \triangleright_{\mathcal{E}} t$, i.e., $s \sim_{\mathcal{E}} s' \triangleright t' \sim_{\mathcal{E}} t$ for some terms s', t' . Then $s\sigma \sim_{\mathcal{E}} s'\sigma \triangleright t'\sigma \sim_{\mathcal{E}} t\sigma$ since both $\sim_{\mathcal{E}}$ and \triangleright are stable, i.e., $s\sigma \triangleright_{\mathcal{E}} t\sigma$. Now stability of $\succeq_{\mathcal{E}}$ is obvious since both $\triangleright_{\mathcal{E}}$ and $\sim_{\mathcal{E}}$ are stable.
4. Let $s \triangleright_{\mathcal{E}} t$, i.e., $s \sim_{\mathcal{E}} s' \triangleright t' \sim_{\mathcal{E}} t$, and let $s' \sim_{\mathcal{E}} s$ and $t' \sim_{\mathcal{E}} t$. Thus, $s' \sim_{\mathcal{E}} s' \triangleright t' \sim_{\mathcal{E}} t$, i.e., $s' \triangleright_{\mathcal{E}} t'$. Since $\succeq_{\mathcal{E}} = \triangleright_{\mathcal{E}} \cup \sim_{\mathcal{E}}$ the claim for $\succeq_{\mathcal{E}}$ is now immediate. \square

Proof of Theorem 6.24. In the second case soundness is obvious. Otherwise,

Appendix A. Proofs

it needs to be shown that every infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chains contains only finitely many dependency pairs from \mathcal{P}' . Thus, consider an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain $s_1 \rightarrow t_1[\varphi_1], s_2 \rightarrow t_2[\varphi_2], \dots$ using the Th -based substitution σ and apply the simple projection π to it.

Consider the instantiation $s_i\sigma \rightarrow t_i\sigma[\varphi_i\sigma]$ of the i^{th} dependency pair in this chain. Clearly, $\pi(s_i\sigma) = \pi(s_i)\sigma$ and $\pi(t_i\sigma) = \pi(t_i)\sigma$. Since $\pi(s_i) \triangleright_{\mathcal{E}} \pi(t_i)$ by assumption, $\pi(s_i\sigma) \triangleright_{\mathcal{E}} \pi(t_i\sigma)$ by Lemma 6.23.3. The sequence $t_i\sigma \xrightarrow{S, Q}_{\mathit{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^* \circ \xrightarrow{> \Lambda}_{\mathcal{E} \setminus \mathcal{S}} \circ \xrightarrow{\gtrsim \Lambda}_{\mathcal{E}}$ $s_{i+i}\sigma$ gives rise to the (possibly shorter) sequence $\pi(t_i\sigma) \xrightarrow{S, Q}_{\mathit{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^* \circ \rightarrow_{\mathcal{E} \setminus \mathcal{S}}^* \circ \sim_{\mathcal{E}}$ $\pi(s_{i+i}\sigma)$ since all steps take place below the root. Thus, $t_i\sigma \xrightarrow{S, Q}_{\mathit{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^* \circ \xrightarrow{> \Lambda}_{\mathcal{E} \setminus \mathcal{S}} \circ \xrightarrow{\gtrsim \Lambda}_{\mathcal{E}}$ $s_{i+1}\sigma \rightarrow_{\text{DP}(\mathcal{R})} t_{i+1}\sigma$ gives rise to $\pi(t_i\sigma) \xrightarrow{S, Q}_{\mathit{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^* \circ \rightarrow_{\mathcal{E} \setminus \mathcal{S}}^* \sim_{\mathcal{E}} \pi(s_{i+1}\sigma) \triangleright_{\mathcal{E}} \pi(t_{i+1}\sigma)$. Hence, the infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain gives rise to an infinite $\xrightarrow{S, Q}_{\mathit{Th} \parallel \mathcal{E} \setminus \mathcal{R}} \cup \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \cup \triangleright_{\mathcal{E}} \cup \sim_{\mathcal{E}}$ -sequence starting with $\pi(t_1\sigma)$. Now perform a case analysis.

Case 1: The infinite sequence contains only finitely many $\xrightarrow{S, Q}_{\mathit{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -steps and only finitely many $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ -steps. Then, there exists an infinite $\triangleright_{\mathcal{E}} \cup \sim_{\mathcal{E}}$ -sequence. This sequence cannot contain infinitely many $\triangleright_{\mathcal{E}}$ -steps stemming from dependency pairs in \mathcal{P}' since otherwise Lemma 6.23.4 yields an infinite $\triangleright_{\mathcal{E}}$ -sequence, contradicting the well-foundedness of $\triangleright_{\mathcal{E}}$ (Lemma 6.23.2).

Case 2: The infinite sequence contains only finitely many $\xrightarrow{S, Q}_{\mathit{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -steps but infinitely many $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ -steps. Recall the inclusion $\sim_{\mathcal{E}} \circ \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \subseteq \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \circ \sim_{\mathcal{E}}$ that follows from the strong \mathcal{E} -coherence of $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ (Definition 3.12). Using this and the easily seen inclusion $\triangleright \circ \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \subseteq \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \circ \triangleright$, it is furthermore straightforward to show that

$$\begin{aligned}
 \triangleright_{\mathcal{E}} \circ \rightarrow_{\mathcal{E} \setminus \mathcal{S}} &= \sim_{\mathcal{E}} \circ \triangleright \circ \sim_{\mathcal{E}} \circ \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \\
 &\subseteq \sim_{\mathcal{E}} \circ \triangleright \circ \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \circ \sim_{\mathcal{E}} \\
 &\subseteq \sim_{\mathcal{E}} \circ \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \circ \triangleright \circ \sim_{\mathcal{E}} \\
 &\subseteq \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \circ \sim_{\mathcal{E}} \circ \triangleright \circ \sim_{\mathcal{E}} \\
 &= \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \circ \triangleright_{\mathcal{E}}
 \end{aligned}$$

Appendix A. Proofs

By making repeated use of these inclusions, an infinite $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ -sequence is obtained, contradicting the assumption that $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ is terminating.

Case 3: The infinite sequence contains infinitely many $\xrightarrow{\mathcal{S},\mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}$ -steps. Recall Lemma 3.26 and the inclusions $\sim_{\mathcal{E}} \circ \xrightarrow{\mathcal{S},\mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} \subseteq \xrightarrow{\mathcal{S},\mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} \circ \sim_{\mathcal{E}}$ and $\rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \xrightarrow{\mathcal{S},\mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} \subseteq \xrightarrow{\mathcal{S},\mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}}$. Using the first of these inclusions and the easily seen inclusion $\triangleright \circ \xrightarrow{\mathcal{S},\mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} \subseteq \xrightarrow{\mathcal{S},\mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} \circ \triangleright$, it is furthermore straightforward to show that

$$\begin{aligned} \triangleright_{\mathcal{E}} \circ \xrightarrow{\mathcal{S},\mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} &= \sim_{\mathcal{E}} \circ \triangleright \circ \sim_{\mathcal{E}} \circ \xrightarrow{\mathcal{S},\mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} \\ &\subseteq \sim_{\mathcal{E}} \circ \triangleright \circ \xrightarrow{\mathcal{S},\mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} \circ \sim_{\mathcal{E}} \\ &\subseteq \sim_{\mathcal{E}} \circ \xrightarrow{\mathcal{S},\mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} \circ \triangleright \circ \sim_{\mathcal{E}} \\ &\subseteq \xrightarrow{\mathcal{S},\mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} \circ \sim_{\mathcal{E}} \circ \triangleright \circ \sim_{\mathcal{E}} \\ &= \xrightarrow{\mathcal{S},\mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}} \circ \triangleright_{\mathcal{E}} \end{aligned}$$

By making repeated use of these inclusions, an infinite $\xrightarrow{\mathcal{S},\mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}$ -sequence starting with the term $\pi(t_1\sigma)$ is obtained. But this contradicts the minimality of the infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain since this implies that $t_1\sigma$ also starts an infinite $\xrightarrow{\mathcal{S},\mathcal{Q}}_{Th\|\mathcal{E}\setminus\mathcal{R}}$ reduction. \square

A.6 Proofs from Chapter 7

Proof of Theorem 7.6. It needs to be shown that $\succsim_{\mathcal{P}ol}$ is reflexive, transitive, monotonic, and stable. Furthermore, it needs to be shown that $\succ_{\mathcal{P}ol}$ is well-founded and stable and that $\succ_{\mathcal{P}ol}$ is compatible with $\succsim_{\mathcal{P}ol}$.

$\succsim_{\mathcal{P}ol}$ is reflexive: $[s\sigma]_{\mathcal{P}ol} \geq [s\sigma]_{\mathcal{P}ol}$ is obvious for any ground substitution σ .

$\succsim_{\mathcal{P}ol}$ is transitive: Let $s \succsim_{\mathcal{P}ol} t \succsim_{\mathcal{P}ol} u$ and let σ be a ground substitution. Then $[s\sigma]_{\mathcal{P}ol} \geq [t\sigma]_{\mathcal{P}ol} \geq [u\sigma]_{\mathcal{P}ol}$ by the assumption, and the transitivity of \geq on \mathbb{N} implies

Appendix A. Proofs

$[s\sigma]_{\mathcal{P}ol} \geq [u\sigma]_{\mathcal{P}ol}$. Since this applies for all ground substitutions, $s \succsim_{\mathcal{P}ol} u$.

$\succsim_{\mathcal{P}ol}$ is monotonic: Let $s \succsim_{\mathcal{P}ol} t$ and let $s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_n$ be terms. Then

$$\begin{aligned} & [f(s_1, \dots, s_{i-1}, s, s_{i+1}, \dots, s_n)\sigma]_{\mathcal{P}ol} \\ = & \mathcal{P}ol(f)([s_1\sigma]_{\mathcal{P}ol}, \dots, [s_{i-1}\sigma]_{\mathcal{P}ol}, [s\sigma]_{\mathcal{P}ol}, [s_{i+1}\sigma]_{\mathcal{P}ol}, \dots, [s_n\sigma]_{\mathcal{P}ol}) \end{aligned}$$

and

$$\begin{aligned} & [f(s_1, \dots, s_{i-1}, t, s_{i+1}, \dots, s_n)\sigma]_{\mathcal{P}ol} \\ = & \mathcal{P}ol(f)([s_1\sigma]_{\mathcal{P}ol}, \dots, [s_{i-1}\sigma]_{\mathcal{P}ol}, [t\sigma]_{\mathcal{P}ol}, [s_{i+1}\sigma]_{\mathcal{P}ol}, \dots, [s_n\sigma]_{\mathcal{P}ol}) \end{aligned}$$

$[s\sigma]_{\mathcal{P}ol} \geq [t\sigma]_{\mathcal{P}ol}$ follows from $s \succsim_{\mathcal{P}ol} t$ and thus $[f(s_1, \dots, s_{i-1}, s, s_{i+1}, \dots, s_n)\sigma]_{\mathcal{P}ol} \geq [f(s_1, \dots, s_{i-1}, t, s_{i+1}, \dots, s_n)\sigma]_{\mathcal{P}ol}$ since $\mathcal{P}ol(f) \in \mathbb{N}[x_1, \dots, x_n]$ is weakly increasing in x_i . Since this applies for all ground substitutions, $f(s_1, \dots, s_{i-1}, s, s_{i+1}, \dots, s_n) \succsim_{\mathcal{P}ol} f(s_1, \dots, s_{i-1}, t, s_{i+1}, \dots, s_n)$.

$\succsim_{\mathcal{P}ol}$ is stable: Let $s \succsim_{\mathcal{P}ol} t$ and let τ be a substitution. Then $[s\tau\sigma]_{\mathcal{P}ol} \geq [t\tau\sigma]_{\mathcal{P}ol}$ for all ground substitutions σ , i.e., $s\tau \succsim_{\mathcal{P}ol} t\tau$.

$\succ_{\mathcal{P}ol}$ is well-founded: For a contradiction, assume that $s_1 \succ_{\mathcal{P}ol} s_2 \succ_{\mathcal{P}ol} \dots$ is an infinite descending sequence of terms. Let σ be an arbitrary ground substitution. Then $[s_i\sigma]_{\mathcal{P}ol} > [s_{i+1}\sigma]_{\mathcal{P}ol}$ for all $i \geq 1$. But this is clearly impossible since $[s_i\sigma]_{\mathcal{P}ol} \in \mathbb{N}$ for all $i \geq 1$.

$\succ_{\mathcal{P}ol}$ is compatible with $\succsim_{\mathcal{P}ol}$: Let $s \succsim_{\mathcal{P}ol} t \succ_{\mathcal{P}ol} u \succsim_{\mathcal{P}ol} v$, i.e., $[s\sigma]_{\mathcal{P}ol} \geq [t\sigma]_{\mathcal{P}ol} > [u\sigma]_{\mathcal{P}ol} \geq [v\sigma]_{\mathcal{P}ol}$ for all ground substitutions σ . But then $[s\sigma]_{\mathcal{P}ol} > [v\sigma]_{\mathcal{P}ol}$ for all ground substitutions σ as well and therefore $s \succ_{\mathcal{P}ol} v$. \square

Proof of Theorem 7.8. In the second case soundness is obvious. Otherwise, it needs to be shown that every infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain contains only finitely many dependency pairs from \mathcal{P}' . Thus, let $s_1 \rightarrow t_1[[\varphi_1]], s_2 \rightarrow t_2[[\varphi_2]], \dots$ be an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain using the $\mathcal{T}h$ -based substitution σ . This

Appendix A. Proofs

means that $t_i\sigma \xrightarrow{S, \mathcal{Q}}^*_{Th \parallel \mathcal{E} \setminus \mathcal{R}} \circ \xrightarrow{> \Lambda}_!_{\mathcal{E} \setminus \mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} s_{i+1}\sigma$. Then $t_i\sigma = f^{\#}(t_{i,1}\sigma, \dots, t_{i,n}\sigma)$ and $s_{i+1}\sigma = f^{\#}(s_{i+1,1}\sigma, \dots, s_{i+1,n}\sigma)$ for some $f^{\#}$, where $t_{i,j}\sigma \xrightarrow{S, \mathcal{Q}}^*_{Th \parallel \mathcal{E} \setminus \mathcal{R}} \circ \xrightarrow{> \Lambda}_!_{\mathcal{E} \setminus \mathcal{S}} \circ \sim_{\mathcal{E}} s_{i+1,j}\sigma$ for all $1 \leq j \leq n$. It is shown that $t_i\sigma \gtrsim s_{i+1}\sigma$ for all $i \geq 1$.

For this, it is first shown that $w \vdash_{\mathcal{E}} w'$ implies $w \sim w'$ for all $w, w' \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{Th}, \mathcal{V})$. If $w \vdash_{\mathcal{E}} w'$, then there exist an equation $u \approx v$ (or $v \approx u$) in \mathcal{E} , a position $p \in \mathcal{Pos}(w)$, and a substitution σ such that $w|_p = u\sigma$ and $w' = w[v\sigma]_p$. Notice that $u\sigma \sim v\sigma$ by assumption, and monotonicity of \gtrsim implies $w \sim w'$. Using this result and similar reasoning, $w \rightarrow_{\mathcal{E} \setminus \mathcal{S}} w'$ implies $w \gtrsim w'$.

Next, it is shown that $w \xrightarrow{S, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}} w'$ implies $w \gtrsim w'$ for all $w, w' \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{Th}, \mathcal{V})$. If $w \xrightarrow{S, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}} w'$, then there exist a rule $l \rightarrow r[\varphi]$ in \mathcal{R} , a position $p \in \mathcal{Pos}(w)$, and a Th -based substitution σ such that $w|_p \xrightarrow{> \Lambda}_!_{\mathcal{E} \setminus \mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} l\sigma$, the constraint $\varphi\sigma$ is Th -valid, and $w' = w[r\sigma]_p$. Using the results from above, $w|_p \gtrsim l\sigma$, and monotonicity of \gtrsim implies $w \gtrsim w[l\sigma]_p$. Notice that $l\sigma \gtrsim r\sigma$ by assumption, and monotonicity of \gtrsim gives $w[l\sigma]_p \gtrsim w[r\sigma]_p$, i.e., $w \gtrsim w'$.

Thus, $t_{i,j}\sigma \gtrsim s_{i+1,j}\sigma$ for all $1 \leq j \leq n$ and monotonicity of \gtrsim implies $t_i\sigma \gtrsim s_{i+1}\sigma$ for all $i \geq 1$. Furthermore, $s_i\sigma \gtrsim t_i\sigma$ or $s_i\sigma \succ t_i\sigma$ for all $i \geq 1$ since $s_i \gtrsim t_i$ for all $s_i \rightarrow t_i[\varphi_i] \in \mathcal{P} - \mathcal{P}'$ and $s_i \succ t_i$ for all $s_i \rightarrow t_i[\varphi_i] \in \mathcal{P}'$. Hence, the infinite minimal chain gives rise to

$$s_1\sigma \bowtie_1 t_1\sigma \gtrsim s_2\sigma \bowtie_2 t_2\sigma \gtrsim \dots$$

where $\bowtie_i \in \{\gtrsim, \succ\}$. If the above infinite minimal chain contains infinitely many dependency pairs from \mathcal{P}' , then $\bowtie_i = \succ$ for infinitely many i . In this case, the compatibility of \succ with \gtrsim produces an infinite \succ chain, contradicting the well-foundedness of \succ . Thus, only finitely many dependency pairs from \mathcal{P}' occur in the above infinite minimal chain. \square

Proof of Theorem 7.16. Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be a DP problem and let \mathcal{Pol} be a $Th_{\mathbb{N}}$ -polynomial interpretation such that $\mathcal{Pol}(f^{\#})$ is weakly increasing in all x_i with

Appendix A. Proofs

$i \in \mathcal{RedPos}(f^\#, \mathcal{P})$.

It needs to be shown that $\succsim_{\mathcal{Pol}}$ is \mathcal{F} -monotonic and $f^\#$ -monotonic at position i for all $f^\# \in \mathcal{F}^\#$ and all $i \in \mathcal{RedPos}(f^\#, \mathcal{P})$. Furthermore, it needs to be shown that $\sim_{\mathcal{Pol}}$ is $f^\#$ -monotonic at position i whenever $i \notin \mathcal{RedPos}(f^\#, \mathcal{P})$, that $\succ_{\mathcal{Pol}}$ is well-founded, and that $\succ_{\mathcal{Pol}}$ is compatible with $\succsim_{\mathcal{Pol}}$.

$\succsim_{\mathcal{Pol}}$ is reflexive, transitive, and \mathcal{F} -monotonic: This is shown as for ordinary polynomial interpretations in the proof of Theorem 7.6.

$\succsim_{\mathcal{Pol}}$ is $f^\#$ -monotonic at position i for all $i \in \mathcal{RedPos}(f^\#, \mathcal{P})$: This is shown similarly to the proof that $\succsim_{\mathcal{Pol}}$ is monotonic in the proof of Theorem 7.6. Notice that $\mathcal{Pol}(f^\#)$ is assumed to be weakly increasing in x_i for all $i \in \mathcal{RedPos}(f^\#, \mathcal{P})$.

$\sim_{\mathcal{Pol}}$ is $f^\#$ -monotonic at position i for all $i \notin \mathcal{RedPos}(f^\#, \mathcal{P})$: Let $s \sim_{\mathcal{Pol}} t$ for terms $s, t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{Th}_{\mathbb{N}}}, \mathcal{V})$ and let $s_1, \dots, s_{i-1}, s_{i+1}, s_n \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{Th}_{\mathbb{N}}}, \mathcal{V})$. Then

$$\begin{aligned} & [f(s_1, \dots, s_{i-1}, s, s_{i+1}, \dots, s_n)\sigma]_{\mathcal{Pol}} \\ = & \mathcal{Pol}(f)([s_1\sigma]_{\mathcal{Pol}}, \dots, [s_{i-1}\sigma]_{\mathcal{Pol}}, [s\sigma]_{\mathcal{Pol}}, [s_{i+1}\sigma]_{\mathcal{Pol}}, \dots, [s_n\sigma]_{\mathcal{Pol}}) \end{aligned}$$

and

$$\begin{aligned} & [f(s_1, \dots, s_{i-1}, t, s_{i+1}, \dots, s_n)\sigma]_{\mathcal{Pol}} \\ = & \mathcal{Pol}(f)([s_1\sigma]_{\mathcal{Pol}}, \dots, [s_{i-1}\sigma]_{\mathcal{Pol}}, [t\sigma]_{\mathcal{Pol}}, [s_{i+1}\sigma]_{\mathcal{Pol}}, \dots, [s_n\sigma]_{\mathcal{Pol}}) \end{aligned}$$

$[s\sigma]_{\mathcal{Pol}} = [t\sigma]_{\mathcal{Pol}}$ follows from $s \sim_{\mathcal{Pol}} t$ and thus $[f(s_1, \dots, s_{i-1}, s, s_{i+1}, \dots, s_n)\sigma]_{\mathcal{Pol}} = [f(s_1, \dots, s_{i-1}, t, s_{i+1}, \dots, s_n)\sigma]_{\mathcal{Pol}}$. Since this reasoning applies for all ground substitutions, $f(s_1, \dots, s_{i-1}, s, s_{i+1}, \dots, s_n) \sim_{\mathcal{Pol}} f(s_1, \dots, s_{i-1}, t, s_{i+1}, \dots, s_n)$.

$\succ_{\mathcal{Pol}}$ is well-founded: For a contradiction, assume that $s_1 \succ_{\mathcal{Pol}} s_2 \succ_{\mathcal{Pol}} \dots$ is an infinite descending sequence of terms. Let σ be an arbitrary ground substitution. Then $[s_i\sigma]_{\mathcal{Pol}} > [s_{i+1}\sigma]_{\mathcal{Pol}}$ and $[s_i\sigma]_{\mathcal{Pol}} \geq c_{\mathcal{Pol}}$ for all $i \geq 1$. But this is clearly impossible.

Appendix A. Proofs

$\succ_{\mathcal{P}ol}$ is compatible with $\succsim_{\mathcal{P}ol}$: In order to show $\succsim_{\mathcal{P}ol} \circ \succ_{\mathcal{P}ol} \circ \succsim_{\mathcal{P}ol} \subseteq \succ_{\mathcal{P}ol}$, let $s \succsim_{\mathcal{P}ol} t \succ_{\mathcal{P}ol} u \succsim_{\mathcal{P}ol} v$, i.e., $[s\sigma]_{\mathcal{P}ol} \geq [t\sigma]_{\mathcal{P}ol} > [u\sigma]_{\mathcal{P}ol} \geq [v\sigma]_{\mathcal{P}ol}$ and $[t\sigma]_{\mathcal{P}ol} \geq c_{\mathcal{P}ol}$ for all ground substitutions σ . But then $[s\sigma]_{\mathcal{P}ol} > [v\sigma]_{\mathcal{P}ol}$ and $[s\sigma]_{\mathcal{P}ol} \geq c_{\mathcal{P}ol}$ for all ground substitutions σ as well and therefore $s \succ_{\mathcal{P}ol} v$. \square

Proof of Theorem 7.18. In the second case soundness is obvious. Otherwise, it needs to be shown that every infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain contains only finitely many dependency pairs from \mathcal{P}' . Thus, let $s_1 \rightarrow t_1[\varphi_1], s_2 \rightarrow t_2[\varphi_2], \dots$ be an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain using the $\mathcal{Th}_{\mathbb{N}}$ -based substitution σ . This means that $t_i\sigma \xrightarrow{\mathcal{S}, \mathcal{Q}}^*_{\mathcal{Th}_{\mathbb{N}} \parallel \mathcal{E} \setminus \mathcal{R}} \circ \xrightarrow{\geq \Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \circ \succsim_{\mathcal{E}}^{\Lambda} s_{i+1}\sigma$ and $\varphi_i\sigma$ is $\mathcal{Th}_{\mathbb{N}}$ -valid for all $i \geq 1$. Now $t_i\sigma = f^{\sharp}(t_{i,1}\sigma, \dots, t_{i,n}\sigma)$ and $s_{i+1}\sigma = f^{\sharp}(s_{i+1,1}\sigma, \dots, s_{i+1,n}\sigma)$ for some f^{\sharp} , where $t_{i,j}\sigma \xrightarrow{\mathcal{S}, \mathcal{Q}}^*_{\mathcal{Th}_{\mathbb{N}} \parallel \mathcal{E} \setminus \mathcal{R}} \circ \rightarrow^!_{\mathcal{E} \setminus \mathcal{S}} \circ \sim_{\mathcal{E}} s_{i+1,j}\sigma$ for all $1 \leq j \leq n$. Notice that this implies $t_{i,j}\sigma \rightarrow^*_{\mathcal{E} \setminus \mathcal{S}_{\text{base}}} \circ \sim_{\mathcal{E}} s_{i+1}\sigma$ if $j \notin \text{RedPos}(f^{\sharp}, \mathcal{P})$. Next, it is shown that $t_i\sigma \succsim s_{i+1}\sigma$ for all $i \geq 1$.

Similarly to the proof of Theorem 7.8, $t_{i,j}\sigma \succsim s_{i+1,j}\sigma$ if $j \in \text{RedPos}(f^{\sharp}, \mathcal{P})$ and $t_{i,j}\sigma \sim s_{i+1,j}\sigma$ if $j \notin \text{RedPos}(f^{\sharp}, \mathcal{P})$. Since \succsim is \mathcal{F} -monotonic and f^{\sharp} -monotonic at the relevant positions, and since $\succsim \cap \succsim^{-1}$ is f^{\sharp} -monotonic at the relevant positions, $t_i\sigma \succsim s_{i+1}\sigma$ is obtained and the proof continues as the proof of Theorem 7.8, where $s_i\sigma \succsim t_i\sigma$ for all $s_i \rightarrow t_i[\varphi_i] \in \mathcal{P} - \mathcal{P}'$ follows from $s_i[\varphi_i] \succsim t_i[\varphi_i]$ and the $\mathcal{Th}_{\mathbb{N}}$ -validity of $\varphi_i\sigma$, and similarly $s_i\sigma \succ t_i\sigma$ for all $s_i \rightarrow t_i[\varphi_i] \in \mathcal{P}'$. \square

Proof of Lemma 7.25. The claim is proved by induction on s . If s is a constant c , then $[s]_{\mathcal{P}ol} = \mathcal{P}ol(c) \in \mathbb{N}$. Otherwise, $s = f(s_1, \dots, s_n)$ and the inductive assumption implies $[s_i]_{\mathcal{P}ol} \geq 0$ for all s_i of sort `univ`. Now $[s]_{\mathcal{P}ol} = \mathcal{P}ol(f)([s_1]_{\mathcal{P}ol}, \dots, [s_n]_{\mathcal{P}ol}) \geq 0$ since $\mathcal{P}ol(f) \in \mathbb{N}[x_1, \dots, x_n]$ where $\mathcal{P}ol(f)$ only depends on a variable x_i if the i^{th} argument of f has sort `univ`. \square

Proof of Theorem 7.26. Similar to the proof of Theorem 7.16. \square

Proof of Theorem 7.28. In the second case soundness is obvious. Otherwise, it

Appendix A. Proofs

thus $s\sigma \xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{U}_{\mathcal{R}}(s_i)} t$. Furthermore, $\mathcal{U}_{\mathcal{R}}(s_i) \subseteq \mathcal{R}(f) \cup \bigcup_{r' \rightarrow r'[\varphi'] \in \mathcal{R}(f)} \mathcal{U}_{\mathcal{R}'}(r') \cup \mathcal{U}_{\mathcal{R}'}(s_i) \subseteq \mathcal{U}_{\mathcal{R}}(s)$. This implies $s\sigma \xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{U}_{\mathcal{R}}(s)} t$. Additionally, the inductive assumption implies that there is some term u_i and some normal substitution μ_i such that $t_i = u_i\mu_i$. Let u'_i result from u_i by replacing variables x by corresponding fresh variables x' . Define μ by $\mu(x') = \mu_i(x)$ for these fresh variables and $\mu(x) = \sigma(x)$ for all other variables. Then $t = u\mu$ for $u = f(s_1, \dots, u'_i, \dots, s_n)$. Obviously $\mathcal{U}_{\mathcal{R}}(u'_i) = \mathcal{U}_{\mathcal{R}}(u_i)$ and the inductive assumption implies $\mathcal{U}_{\mathcal{R}}(u'_i) \subseteq \mathcal{U}_{\mathcal{R}}(s_i)$. As shown above, $\mathcal{U}_{\mathcal{R}}(s_i) \subseteq \mathcal{U}_{\mathcal{R}}(s)$ and thus $\mathcal{U}_{\mathcal{R}}(u'_i) \subseteq \mathcal{U}_{\mathcal{R}}(s)$. Thus, $\mathcal{U}_{\mathcal{R}}(u) \subseteq \mathcal{U}_{\mathcal{R}}(s)$ since $\mathcal{U}_{\mathcal{R}}(u)$ differs from $\mathcal{U}_{\mathcal{R}}(s)$ only by containing $\mathcal{U}_{\mathcal{R}'}(u'_i)$ instead of $\mathcal{U}_{\mathcal{R}'}(s_i)$ and since $\mathcal{U}_{\mathcal{R}'}(u'_i) \subseteq \mathcal{U}_{\mathcal{R}}(u_i) \subseteq \mathcal{U}_{\mathcal{R}}(s)$.

2. The claim immediately follows from the previous part. \square

Proof of Theorem 8.7. The proof is similar to the proofs of Theorems 7.6, 7.16, and 7.26. \square

Proof of Theorem 8.8. In the second case soundness is obvious. Otherwise, let $s_1 \rightarrow t_1[\varphi_1], s_2 \rightarrow t_2[\varphi_2], \dots$ be an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain using the Th -based substitution σ . This means that $t_i\sigma \xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{R}}^* \circ \xrightarrow{>\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} s_{i+1}\sigma$ and $\varphi_i\sigma$ is Th -valid for all $i \geq 1$. It is now shown that rules from \mathcal{R}' are used for only finitely many i , i.e., that $t_i\sigma \xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{R}-\mathcal{R}'}^* \circ \xrightarrow{>\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} s_{i+1}\sigma$ for all $i \geq n$ for some $n \geq 1$.

For a contradiction, assume that rules from \mathcal{R}' are used for infinitely many i . By extending the proofs of Theorems 7.8, 7.18, and 7.28 and by making use of the assumption that (\gtrsim, \succ) is monotonic,

$$s_1\sigma \bowtie_1 t_1\sigma (\gtrsim \cup \succ) s_2\sigma \bowtie_2 t_2\sigma (\gtrsim \cup \succ) \dots$$

where $\bowtie_j \in \{\gtrsim, \succ\}$ and $t_i\sigma \succ s_{i+1}\sigma$ if the reduction $t_i\sigma \xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{R}}^* \circ \xrightarrow{>\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} s_{i+1}\sigma$ uses a rule from \mathcal{R}' . As before, $\bowtie_j = \succ$ for only finitely many j , i.e., there is an

Appendix A. Proofs

infinite tail

$$s_k\sigma \succsim t_k\sigma (\succsim \cup \succ) s_{k+1}\sigma \succsim t_{k+1}\sigma (\succsim \cup \succ) \dots$$

If this infinite tail uses rules from \mathcal{R}' infinitely often, then $t_i\sigma \succ s_{i+1}\sigma$ for infinitely many i and the compatibility of \succ with \succsim produces an infinite \succ chain, contradicting the well-foundedness of \succ . Thus, rules from \mathcal{R}' are used for only finitely many i and there thus exists an infinite minimal $(\mathcal{P} - \mathcal{P}', \mathcal{Q}, \mathcal{R} - \mathcal{R}', \mathcal{S}, \mathcal{E})$ -chain. \square

Before proving Lemma 8.14, some auxiliary results need to be obtained. Lemma 8.14 will be shown by well-founded induction using the following relation.

Definition A.1 (\heartsuit). *Let s and t be terms. Then $s \heartsuit t$ iff $s \rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \sim_{\mathcal{E}} \cup \xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}} \circ \sim_{\mathcal{E}} \cup \triangleright_{\mathcal{E}} t$.*

In order to use \heartsuit for inductive proofs, the following properties are needed.

Lemma A.2. *Let s be a terminating term.*

1. *If $s \heartsuit t$, then t is terminating.*
2. *\heartsuit is well-founded on terminating terms.*

Proof. Let s be terminating.

1. If $s \rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \sim_{\mathcal{E}} t$ or $s \xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}} \circ \sim_{\mathcal{E}} t$, then t is terminating by Corollary 3.27. If $s \triangleright_{\mathcal{E}} t$ then t is terminating due to the inclusion $\triangleright_{\mathcal{E}} \circ \xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}} \subseteq \xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}} \circ \triangleright_{\mathcal{E}}$ from the proof of Theorem 6.24.
2. Assume that \heartsuit is not well-founded on terminating terms. Then, there exists an infinite $\rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \sim_{\mathcal{E}} \cup \xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}} \circ \sim_{\mathcal{E}} \cup \triangleright_{\mathcal{E}}$ -sequence containing only terminating terms.

If this sequence contains only finitely many $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}} \circ \sim_{\mathcal{E}}$ -steps, then there is an infinite $\rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \sim_{\mathcal{E}} \cup \triangleright_{\mathcal{E}}$ -sequence. Since $\triangleright_{\mathcal{E}}$ is well-founded, this sequence contains infinitely many $\rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \sim_{\mathcal{E}}$ -steps. Using the inclusions $\triangleright_{\mathcal{E}} \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}}$

Appendix A. Proofs

$\subseteq \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \circ \triangleright_{\mathcal{E}}$ from the proof of Theorem 6.24 and $\sim_{\mathcal{E}} \circ \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \subseteq \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \circ \sim_{\mathcal{E}}$ from Definition 3.12, an infinite $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ -sequence is obtained, contradicting the assumption that $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ is terminating.

Otherwise, the sequence contains infinitely many $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}} \circ \sim_{\mathcal{E}}$ -steps. Using the inclusions $\sim_{\mathcal{E}} \circ \xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}} \subseteq \xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}} \circ \sim_{\mathcal{E}}$ and $\rightarrow_{\mathcal{E} \setminus \mathcal{S}} \circ \xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}} \subseteq \xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}} \circ \rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ from Lemma 3.26 and $\triangleright_{\mathcal{E}} \circ \xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}} \subseteq \xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}} \circ \triangleright_{\mathcal{E}}$ from the proof of Theorem 6.24, an infinite $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}}$ -sequence starting with a terminating term is obtained, which is clearly impossible. \square

The first property will be used freely in the following. Next, it is shown that $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ and $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}}$ are finitely branching if \mathcal{E} is size-preserving.

Lemma A.3. *Let $(\mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be an RCERS such that \mathcal{E} is size-preserving. Then $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ and $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}}$ are finitely branching.*

Proof. First, the property is shown for $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$. Since a term has only finitely many positions and since \mathcal{S} is a finite set of rules, it suffices to show that $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ is finitely branching if only one position p and one rule $l \rightarrow r \in \mathcal{S}$ is considered. Without loss of generality assume $p = \Lambda$, i.e., $s \sim_{\mathcal{E}} l\sigma$. Since \mathcal{E} is size-preserving, the \mathcal{E} -equivalence classes are finite and there are thus only finitely many substitutions σ for \mathcal{E} -matching that differ on $\mathcal{V}(l)$. Therefore, $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ is finitely branching.

The proof for $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}}$ is similar, where it again suffices to consider only the root position and a single rule. Thus, $s \xrightarrow{\geq \Lambda}!_{\mathcal{E} \setminus \mathcal{S}} s' \approx_{\mathcal{E}}^{\geq \Lambda} l\sigma$, where σ is Th -based. Since $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ is \mathcal{E} -convergent and since \mathcal{E} is size-preserving there are only finitely many possible terms s' . As above, there are only finitely many substitutions σ for each s' , which implies that $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{Th \parallel \mathcal{E} \setminus \mathcal{R}}$ is finitely branching. \square

Proof of Lemma 8.14. The claim is shown by induction on \heartsuit . If $t \in \mathcal{V}$ then $\mathcal{I}(t) = t$ and nothing needs to be shown. If $t = f(t_1, \dots, t_n)$ with $f \in \Delta$ then $\mathcal{I}(t) = f(\mathcal{I}(t_1), \dots, \mathcal{I}(t_n))$ and the inductive assumption implies that $\mathcal{I}(t_i)$ is a finite

Appendix A. Proofs

term for all $1 \leq i \leq n$. But then $\mathcal{I}(t)$ is clearly a finite term as well. Finally, assume $\text{root}(t) \notin \Delta$. Then the sets $\mathcal{R}ed_{\mathcal{S}}(t)$, $\mathcal{R}ed_{\mathcal{R}}(t)$, and $\mathcal{E}q_{\mathcal{E}}(t)$ are finite since $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ and $\xrightarrow{\mathcal{S}, \mathcal{Q}}_{\mathcal{T}h \parallel \mathcal{E} \setminus \mathcal{R}}$ are finitely branching by Lemma A.3 and \mathcal{E} is size-preserving, which implies that the \mathcal{E} -equivalence classes are finite. By the inductive assumption, $\mathcal{I}(t')$ is a finite term for any $\mathcal{I}(t') \in \mathcal{R}ed_{\mathcal{S}}(t) \cup \mathcal{R}ed_{\mathcal{R}}(t)$ and $\mathcal{I}(t_i)$ is a finite term for any $g(\mathcal{I}(t_1), \dots, \mathcal{I}(t_n)) \in \mathcal{E}q_{\mathcal{E}}(t)$ and all $1 \leq i \leq n$. Thus, $\mathcal{I}(t)$ is a finite term as well. \square

Proof of Lemma 8.16. Let $t_1 <_{\mathcal{T}} \dots <_{\mathcal{T}} t_n$ where all t_i have sort s . Then it is easy to see that $\text{Comp}_s(\{t_1, \dots, t_n\}) = \Pi_s(t_1, \dots, \Pi_s(t_i, \dots, \Pi_s(t_n, \perp_s) \dots) \dots) \rightarrow_{\mathcal{R}_{\Pi}}^* \Pi_s(t_i, \dots, \Pi_s(t_n, \perp_s) \dots) \rightarrow_{\mathcal{R}_{\Pi}} t_i$ for any $1 \leq i \leq n$. \square

Proof of Lemma 8.17. Let $s, t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{T}h}, \mathcal{V})$ and let σ be a $\mathcal{T}h$ -based substitution such that s , t , and σ are terminating.

1. $\mathcal{I}(s\sigma) = s\mathcal{I}(\sigma)$ is proved by induction on s . If $s \in \mathcal{V}$ then this is immediate by the definition of $\mathcal{I}(\sigma)$. Otherwise, $s = f(s_1, \dots, s_n)$ with $f \in \Delta$. But then

$$\begin{aligned} \mathcal{I}(s\sigma) &= \mathcal{I}(f(s_1\sigma, \dots, s_n\sigma)) \\ &= f(\mathcal{I}(s_1\sigma), \dots, \mathcal{I}(s_n\sigma)) \\ &= f(s_1\mathcal{I}(\sigma), \dots, s_n\mathcal{I}(\sigma)) \\ &= s\mathcal{I}(\sigma) \end{aligned}$$

by the inductive assumption.

2. $\mathcal{I}(s\sigma) \rightarrow_{\mathcal{R}_{\Pi}}^* s\mathcal{I}(\sigma)$ is proved by induction on s . If $s \in \mathcal{V}$ then this is immediate by the definition of $\mathcal{I}(\sigma)$. Otherwise, $s = f(s_1, \dots, s_n)$. If $f \in \Delta$ then $\mathcal{I}(s\sigma) = f(\mathcal{I}(s_1\sigma), \dots, \mathcal{I}(s_n\sigma)) \rightarrow_{\mathcal{R}_{\Pi}}^* f(s_1\mathcal{I}(\sigma), \dots, s_n\mathcal{I}(\sigma)) = s\mathcal{I}(\sigma)$ by the inductive assumption. If $f \notin \Delta$ then $\mathcal{I}(s\sigma) = \text{Comp}_{\text{sort}(s\sigma)}(\mathcal{R}ed_{\mathcal{S}}(s\sigma) \cup \mathcal{R}ed_{\mathcal{R}}(s\sigma) \cup \mathcal{E}q_{\mathcal{E}}(s\sigma))$. Notice that $f(\mathcal{I}(s_1\sigma), \dots, \mathcal{I}(s_n\sigma)) \in \mathcal{E}q_{\mathcal{E}}(s\sigma)$ and therefore $\mathcal{I}(s\sigma) \xrightarrow{+}_{\mathcal{R}_{\Pi}} f(\mathcal{I}(s_1\sigma), \dots, \mathcal{I}(s_n\sigma)) \rightarrow_{\mathcal{R}_{\Pi}}^* f(s_1\mathcal{I}(\sigma), \dots, s_n\mathcal{I}(\sigma))$ by Lemma 8.16 and the inductive assumption.

Appendix A. Proofs

3. It suffices to show that $s \vdash_{\mathcal{E}} t$ implies $\mathcal{I}(s) \vdash_{\mathcal{E}(\Delta)}^{\bar{=}} \mathcal{I}(t)$ since the statement then follows by induction on the number of $\vdash_{\mathcal{E}}$ -steps in $s \sim_{\mathcal{E}} t$.

Thus, let $s \vdash_{\mathcal{E}} t$ and perform an induction on the position p where the step takes places. If $\text{root}(s) \notin \Delta$ then $\text{root}(t) \notin \Delta$ as well by the definition of Δ . Since $s \sim_{\mathcal{E}} t$, Definition 3.12 implies that whenever $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}} s'$, then $t \rightarrow_{\mathcal{E} \setminus \mathcal{S}} t'$ for some $t' \sim_{\mathcal{E}} s'$. Thus, $\mathcal{R}ed_{\mathcal{S}}(s) \subseteq \mathcal{R}ed_{\mathcal{S}}(t)$. Similarly, Lemma 3.26.1 implies $\mathcal{R}ed_{\mathcal{R}}(s) \subseteq \mathcal{R}ed_{\mathcal{R}}(t)$. Finally, if $s \sim_{\mathcal{E}} g(t_1, \dots, t_n)$, then $t \sim_{\mathcal{E}} g(t_1, \dots, t_n)$, which immediately implies $\mathcal{E}q_{\mathcal{E}}(s) = \mathcal{E}q_{\mathcal{E}}(t)$. Using these properties, $\mathcal{R}ed_{\mathcal{S}}(s) \cup \mathcal{R}ed_{\mathcal{R}}(s) \cup \mathcal{E}q_{\mathcal{E}}(s) \subseteq \mathcal{R}ed_{\mathcal{S}}(t) \cup \mathcal{R}ed_{\mathcal{R}}(t) \cup \mathcal{E}q_{\mathcal{E}}(t)$. Since the same reasoning can be applied with s and t interchanged,

$$\begin{aligned} & \text{Comp}_{\text{sort}(s)}(\mathcal{R}ed_{\mathcal{S}}(s) \cup \mathcal{R}ed_{\mathcal{R}}(s) \cup \mathcal{E}q_{\mathcal{E}}(s)) \\ &= \text{Comp}_{\text{sort}(t)}(\mathcal{R}ed_{\mathcal{S}}(t) \cup \mathcal{R}ed_{\mathcal{R}}(t) \cup \mathcal{E}q_{\mathcal{E}}(t)) \end{aligned}$$

and thus $\mathcal{I}(s) = \mathcal{I}(t)$.

Otherwise, $\text{root}(s) \in \Delta$. If $p = \Lambda$, then there exist an equation $u \approx v$ (or $v \approx u$) in \mathcal{E} and a substitution σ such that $s = u\sigma$ and $t = v\sigma$. By the definition of Δ , $u, v \in \mathcal{T}(\Delta, \mathcal{V})$. Hence, $\mathcal{I}(s) = \mathcal{I}(u\sigma) = u\mathcal{I}(\sigma) \vdash_{\mathcal{E}(\Delta)} v\mathcal{I}(\sigma) = \mathcal{I}(v\sigma) = \mathcal{I}(t)$ by 1. If $\text{root}(s) \in \Delta$ and $p \neq \Lambda$, then $s = f(s_1, \dots, s_i, \dots, s_n)$ and $t = f(s_1, \dots, t_i, \dots, s_n)$ where $s_i \vdash_{\mathcal{E}} t_i$. Now the definition of \mathcal{I} gives $\mathcal{I}(s) = f(\mathcal{I}(s_1), \dots, \mathcal{I}(s_i), \dots, \mathcal{I}(s_n))$ and $\mathcal{I}(t) = f(\mathcal{I}(s_1), \dots, \mathcal{I}(t_i), \dots, \mathcal{I}(s_n))$. Thus $\mathcal{I}(s) \vdash_{\mathcal{E}(\Delta)}^{\bar{=}} \mathcal{I}(t)$ since $\mathcal{I}(s_i) \vdash_{\mathcal{E}(\Delta)}^{\bar{=}} \mathcal{I}(t_i)$ by the inductive assumption.

4. It suffices to show that $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}} t$ implies $\mathcal{I}(s) \rightsquigarrow_1 \mathcal{I}(t)$ since the statement then follows by induction on the number of $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ -steps in $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}}^* t$.

Thus, let $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}} t$ and perform an induction on the position p where the reduction takes places. If $\text{root}(s) \notin \Delta$, then $\mathcal{I}(t) \in \mathcal{R}ed_{\mathcal{S}}(s)$, which implies $\mathcal{I}(s) \rightarrow_{\mathcal{R}_{\Pi}}^+ \mathcal{I}(t)$ by Lemma 8.16.

If $\text{root}(s) \in \Delta$, first consider the case $p = \Lambda$. Then, there exist a rule $l \rightarrow r \in \mathcal{S}$ and a substitution σ such that $s \sim_{\mathcal{E}} l\sigma \rightarrow_{\mathcal{S}} r\sigma = t$. Since $\text{root}(s) \in \Delta$, the

Appendix A. Proofs

definition of Δ implies that $\text{root}(l) \in \Delta$, $l \rightarrow r \in \mathcal{S}(\Delta)$, and $r \in \mathcal{T}(\Delta, \mathcal{V})$. Using 1, 2, and 3, $\mathcal{I}(s) \sim_{\mathcal{E}(\Delta)} \mathcal{I}(l\sigma) \xrightarrow{*}_{\mathcal{R}_{\Pi}} l\mathcal{I}(\sigma) \xrightarrow{\mathcal{S}(\Delta)} r\mathcal{I}(\sigma) = \mathcal{I}(r\sigma) = \mathcal{I}(t)$, and thus $\mathcal{I}(s) \rightsquigarrow_1 \mathcal{I}(t)$. If $\text{root}(s) \in \Delta$ and $p \neq \Lambda$, then $s = f(s_1, \dots, s_i, \dots, s_n)$ and $t = f(s_1, \dots, t_i, \dots, s_n)$, where $s_i \rightarrow_{\mathcal{E} \setminus \mathcal{S}} t_i$. Now $\mathcal{I}(s_i) \rightsquigarrow_1 \mathcal{I}(t_i)$ by the inductive assumption and therefore $\mathcal{I}(s) = f(\mathcal{I}(s_1), \dots, \mathcal{I}(s_i), \dots, \mathcal{I}(s_n)) \rightsquigarrow_1 f(\mathcal{I}(s_1), \dots, \mathcal{I}(t_i), \dots, \mathcal{I}(s_n)) = \mathcal{I}(t)$.

5. It suffices to show that $s \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} t$ implies $\mathcal{I}(s) \rightsquigarrow_2 \mathcal{I}(t)$. Then the statement follows by induction on the number of $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ -steps in $s \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}^* t$.

Thus, let $s \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} t$ and perform an induction on the position p where the reduction takes places. If $\text{root}(s) \notin \Delta$, then $\mathcal{I}(t) \in \mathcal{Red}_{\mathcal{R}}(s)$, which implies $\mathcal{I}(s) \xrightarrow{+}_{\mathcal{R}_{\Pi}} \mathcal{I}(t)$ by Lemma 8.16.

If $\text{root}(s) \in \Delta$, first consider the case $p = \Lambda$. Then, there exist a rule $l \rightarrow r[\varphi] \in \mathcal{R}$ and a \mathcal{Th} -based substitution σ with $s = f(s^*) \xrightarrow{>\Lambda!}_{\mathcal{E} \setminus \mathcal{S}} \overset{\Lambda}{\sim}_{\mathcal{E}} l\sigma \xrightarrow{\mathcal{R}} r\sigma = t$ such that $\varphi\sigma$ is \mathcal{Th} -valid. Since $\text{root}(l) = \text{root}(s) = f$ and $f \in \Delta$, the definition of Δ implies that $l \rightarrow r[\varphi] \in \mathcal{R}(\Delta)$ and $r \in \mathcal{T}(\Delta, \mathcal{V})$. Using 1, 2, 3, and 4, $\mathcal{I}(s) \rightsquigarrow_1^* \circ \sim_{\mathcal{E}(\Delta)} \mathcal{I}(l\sigma) \xrightarrow{*}_{\mathcal{R}_{\Pi}} l\mathcal{I}(\sigma) \xrightarrow{\mathcal{R}(\Delta)} r\mathcal{I}(\sigma) = \mathcal{I}(r\sigma) = \mathcal{I}(t)$ where $\varphi\mathcal{I}(\sigma) = \varphi\sigma$ is \mathcal{Th} -valid, and thus $\mathcal{I}(s) \rightsquigarrow_2 \mathcal{I}(t)$. If $\text{root}(s) \in \Delta$ and $p \neq \Lambda$, then $s = f(s_1, \dots, s_i, \dots, s_n)$ and $t = f(s_1, \dots, t_i, \dots, s_n)$, where $s_i \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} t_i$. Now $\mathcal{I}(s_i) \rightsquigarrow_2 \mathcal{I}(t_i)$ by the inductive assumption and therefore $\mathcal{I}(s) = f(\mathcal{I}(s_1), \dots, \mathcal{I}(s_i), \dots, \mathcal{I}(s_n)) \rightsquigarrow_2 f(\mathcal{I}(s_1), \dots, \mathcal{I}(t_i), \dots, \mathcal{I}(s_n)) = \mathcal{I}(t)$.

6. Let $s\sigma \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} \circ \rightarrow_{\mathcal{E} \setminus \mathcal{S}}^! \circ \sim_{\mathcal{E}} t\sigma$. Using 3, 4, and 5, $\mathcal{I}(s\sigma) \rightsquigarrow_2^* \circ \rightsquigarrow_1^* \circ \sim_{\mathcal{E}(\Delta)} \mathcal{I}(t\sigma)$. Using 1 and 2 this implies $s\mathcal{I}(\sigma) \rightsquigarrow_2^* \circ \rightsquigarrow_1^* \circ \sim_{\mathcal{E}(\Delta)} \circ \xrightarrow{*}_{\mathcal{R}_{\Pi}} t\mathcal{I}(\sigma)$. \square

Proof of Theorem 8.18. In the second case soundness is obvious. Otherwise, it needs to be shown that every infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain contains only finitely many dependency pairs from \mathcal{P}' . This is done similarly to the proofs of previous theorems based on reduction pairs. Thus, let $s_1 \rightarrow t_1[\varphi_1], s_2 \rightarrow t_2[\varphi_2], \dots$ be an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain using the \mathcal{Th} -based substitution σ . There-

Appendix A. Proofs

fore, $t_i\sigma \xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{R}}^* \circ \xrightarrow{\geq\Delta}_!_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Delta} s_{i+1}\sigma$ and $\varphi_i\sigma$ is \mathcal{Th} -valid for all $i \geq 1$. Thus, $t_i\sigma = f^\sharp(t_{i,1}\sigma, \dots, t_{i,n}\sigma)$ and $s_{i+1}\sigma = f^\sharp(s_{i+1,1}\sigma, \dots, s_{i+1,n}\sigma)$ for some f^\sharp , where $t_{i,j}\sigma \xrightarrow{S}_{Th\|\mathcal{E}\setminus\mathcal{R}}^* \circ \rightarrow!_{\mathcal{E}\setminus\mathcal{S}} \circ \sim_{\mathcal{E}} s_{i+1,j}\sigma$ for all $1 \leq j \leq n$. Since $t_{i,j} \in \mathcal{T}(\Delta, \mathcal{V})$ by the definition of Δ , Lemma 8.17.6 yields $t_{i,j}\mathcal{I}(\sigma) \rightsquigarrow_2^* \circ \rightsquigarrow_1^* \circ \sim_{\mathcal{E}(\Delta)} \circ \rightarrow_{\mathcal{R}_\Pi}^* s_{i+1,j}\mathcal{I}(\sigma)$ for all $1 \leq j \leq n$.

As before, $t_i\mathcal{I}(\sigma) \gtrsim s_{i+1}\mathcal{I}(\sigma)$ for all $i \geq 1$ and the proof continues unchanged. \square

Proof of Theorem 8.20. Combine the proofs of Theorems 8.8 and 8.18. \square

A.8 Proofs from Chapter 9

Proof of Lemma 9.1. Notice that $s\mu \xrightarrow{\geq\Delta}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Delta} t\mu$ implies $s\mu \sim_{\mathcal{E}\cup\mathcal{S}} t\mu$. \square

Proof of Lemma 9.3. In order to show that $s\mu \xrightarrow{\geq\Delta}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Delta} t\mu$ implies $\text{CAP}_{\mathcal{S}}(s)\vartheta \gtrsim_{\mathcal{E}}^{\Delta} t\vartheta$ for some substitution ϑ , it is first shown that $\text{CAP}_{\mathcal{S}}(s)\mu \rightarrow_{\mathcal{E}\setminus\mathcal{S}} u$ implies $u = \text{CAP}_{\mathcal{S}}(s)\tau$ for some substitution τ such that μ and τ differ at most for the fresh variables introduced by $\text{CAP}_{\mathcal{S}}$. This is shown by induction on s . If $\text{CAP}_{\mathcal{S}}(s) \in \mathcal{V}$, then it is a fresh variable y and $\tau = \{y \mapsto u\}$ establishes the claim. Otherwise, $s = f(s_1, \dots, s_n)$, $\text{CAP}_{\mathcal{S}}(s) = f(\text{CAP}_{\mathcal{S}}(s_1), \dots, \text{CAP}_{\mathcal{S}}(s_n))$ and there is no rule $l \rightarrow r \in \mathcal{S}$ such that $f(\text{CAP}_{\mathcal{S}}(s_1), \dots, \text{CAP}_{\mathcal{S}}(s_n))$ and l are \mathcal{E} -unifiable. First, it is shown that the $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ -step in $\text{CAP}_{\mathcal{S}}(s)\mu \rightarrow_{\mathcal{E}\setminus\mathcal{S}} u$ cannot take place at the root position. If the reduction takes place at the root position, then there exist a rule $l \rightarrow r \in \mathcal{S}$ and a substitution ρ such that $f(\text{CAP}_{\mathcal{S}}(s_1), \dots, \text{CAP}_{\mathcal{S}}(s_n))\mu \sim_{\mathcal{E}} l\rho$. Since it can be assumed that l is variable-disjoint from $f(\text{CAP}_{\mathcal{S}}(s_1), \dots, \text{CAP}_{\mathcal{S}}(s_n))$, define the substitution $\vartheta = \mu \cup \rho$. Since then $f(\text{CAP}_{\mathcal{S}}(s_1), \dots, \text{CAP}_{\mathcal{S}}(s_n))\vartheta \sim_{\mathcal{E}} l\vartheta$, this gives a contradiction to the assumption. Thus, there exists an $1 \leq i \leq n$ such that $\text{CAP}_{\mathcal{S}}(s_i)\mu \rightarrow_{\mathcal{E}\setminus\mathcal{S}} u_i$ and $u = f(\text{CAP}_{\mathcal{S}}(s_1)\mu, \dots, u_i, \dots, \text{CAP}_{\mathcal{S}}(s_n)\mu)$. By the inductive assumption, there is a substitution δ such that $u_i = \text{CAP}_{\mathcal{S}}(s_i)\delta$. Since the fresh

Appendix A. Proofs

variables introduced by $\text{CAP}_{\mathcal{S}}(s_i)$ are disjoint from the fresh variables introduced by $\text{CAP}_{\mathcal{S}}(s_j)$ for $1 \leq j \neq i \leq n$ and since μ and δ differ at most for the fresh variables introduced by $\text{CAP}_{\mathcal{S}}(s_i)$, define the substitution τ with $\tau(x) = \delta(x)$ if x is a fresh variable introduced by $\text{CAP}_{\mathcal{S}}(s_i)$ and $\tau(x) = \mu(x)$ otherwise. Using the substitution defined this way, $u = f(\text{CAP}_{\mathcal{S}}(s_1)\mu, \dots, \text{CAP}_{\mathcal{S}}(s_i)\delta, \dots, \text{CAP}_{\mathcal{S}}(s_n)\mu) = f(\text{CAP}_{\mathcal{S}}(s_1)\tau, \dots, \text{CAP}_{\mathcal{S}}(s_i)\tau, \dots, \text{CAP}_{\mathcal{S}}(s_n)\tau) = \text{CAP}_{\mathcal{S}}(s)\tau$.

By induction on the number of the $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ -steps, $\text{CAP}_{\mathcal{S}}(s)\mu \rightarrow_{\mathcal{E} \setminus \mathcal{S}}^* u$ for a substitution μ implies $u = \text{CAP}_{\mathcal{S}}(s)\delta$ for some substitution δ such that μ and δ differ at most for the fresh variables introduced by $\text{CAP}_{\mathcal{S}}(s)$. Since $s = \text{CAP}_{\mathcal{S}}(s)\sigma'$ for some substitution σ' which only instantiates the fresh variables introduced by $\text{CAP}_{\mathcal{S}}(s)$, it is in particular true that $s\mu = \text{CAP}_{\mathcal{S}}(s)\sigma'\mu \rightarrow_{\mathcal{E} \setminus \mathcal{S}}^* u$ implies $u = \text{CAP}_{\mathcal{S}}(s)\delta$ for some substitution δ such that $\sigma'\mu$ and δ differ at most for the fresh variables introduced by $\text{CAP}_{\mathcal{S}}(s)$. Thus $\text{CAP}_{\mathcal{S}}(s)\delta = u \stackrel{\geq \Lambda}{\sim}_{\mathcal{E}} t\mu$ since $u \stackrel{\geq \Lambda}{\sim}_{\mathcal{E}} t\mu$. Now define the substitution η by $\eta(x) = \delta(x)$ if x is a fresh variable introduced by $\text{CAP}_{\mathcal{S}}(s)$ and $\eta(x) = \mu(x)$ otherwise. Then $\text{CAP}_{\mathcal{S}}(s)\eta = u \stackrel{\geq \Lambda}{\sim}_{\mathcal{E}} t\mu = t\eta$, i.e., $\text{CAP}_{\mathcal{S}}(s)$ and t are \mathcal{E} -unifiable. \square

A.9 Proofs from Chapter 10

Proof of Lemma 10.6. Assume $s' \sim_{\mathcal{E}} s \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} t$. It needs to be shown that $s' \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} t' \sim_{\mathcal{E}} t$ for some t' . Now $s \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} t$ implies $s = C[f(u^*)]$ for some context C and some $f \in \mathcal{D}(\mathcal{R})$, where $f(u^*) \xrightarrow{\geq \Lambda}!_{\mathcal{E} \setminus \mathcal{S}} \circ \stackrel{\geq \Lambda}{\sim}_{\mathcal{E}} l\sigma$ for some conditional constrained rewrite rule $s_1 \rightarrow^* t_1, \dots, s_n \rightarrow^* t_n \mid l \rightarrow r[\varphi] \in \mathcal{R}$ and some \mathcal{Th} -based substitution σ such that $\varphi\sigma$ is \mathcal{Th} -valid, $s_i\sigma \xrightarrow{\mathcal{S}}!_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} \circ \sim_{\mathcal{E}} t_i\sigma$ for all $1 \leq i \leq n$, and $t = C[r\sigma]$. Since $s' \sim_{\mathcal{E}} s$ and all equations in \mathcal{E} are i.u.v. and do not contain symbols from $\mathcal{D}(\mathcal{R})$, an application of Lemma 2.21 implies $s' = C'[f(u^*)]$ for some context C' with $C' \sim_{\mathcal{E}} C$ and $u^* \sim_{\mathcal{E}} u^*$. Therefore, $f(u^*) \xrightarrow{\geq \Lambda}!_{\mathcal{E} \setminus \mathcal{S}} \circ \stackrel{\geq \Lambda}{\sim}_{\mathcal{E}} l\sigma$ by Lemma 2.28. Thus, σ can be used to rewrite $s' = C'[f(u^*)]$ to $t' = C'[r\sigma] \sim_{\mathcal{E}} C[r\sigma] = t$. \square

Appendix A. Proofs

Proof of Lemma 10.13. Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CERS.

“ \Rightarrow ”: Assume that $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is not terminating and let $s_0 \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}} s_1 \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}} s_2 \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}} \dots$ be an infinite rewrite sequence. Construct an infinite proof tree $\{T_i\}_{i \geq 0}$ as

$$T_0 = s_0 \rightarrow^* t \quad \text{for some arbitrary term } t \quad (\text{an open goal})$$

$$T_{i+1} = \frac{\frac{\text{————— Repl}}{s_i \rightarrow s_{i+1}} \quad s_{i+1} \rightarrow^* t}{T_i} \text{Tran}$$

Here, T_i is extended at its (only) open goal $s_i \rightarrow^* t$. Therefore, $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is not operationally terminating.

“ \Leftarrow ”: Assume that $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is not operationally terminating. Thus, there exists an infinite proof tree $\{T_i\}_{i \geq 0}$. Since \mathcal{R} is unconditional, the shared head of the T_i has the form $s_0 \rightarrow^* t$ for some terms s_0 and t . Furthermore, the rule **Tran** is applied to this head. The leftmost subgoal thus generated is closed using **Repl** before the rightmost subgoal can be expanded further.

$$\frac{\frac{\text{————— Repl}}{s_0 \rightarrow s_1} \quad s_1 \rightarrow^* t}{s_0 \rightarrow^* t} \text{Tran}$$

Thus, $s_0 \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}} s_1$. Applying the same argument to the subgoal $s_1 \rightarrow^* t$ and continuing in the same fashion afterwards, the infinite rewrite sequence

$$s_0 \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}} s_1 \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}} s_2 \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}} \dots$$

is obtained and $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is thus not terminating. \square

The following properties are immediate consequences of Lemma 10.6. They are used freely in the proof of Lemma 10.16.

Appendix A. Proofs

Property A.4. Given a proof tree

$$\frac{T_1 \quad \cdots \quad T_n}{s \rightarrow t}$$

and a term $s' \sim_{\mathcal{E}} s$, the following is also a proof tree:

$$\frac{T_1 \quad \cdots \quad T_n}{s' \rightarrow t'}$$

Here, $t' \sim_{\mathcal{E}} t$ is given by Lemma 10.6. □

Property A.5. Let

$$\frac{\frac{\frac{T_1}{s_0 \rightarrow s_1} \quad \frac{s_1 \rightarrow s_2}{\vdots}}{s_1 \rightarrow^* t} \quad \frac{\frac{T_2}{s_{n-1} \rightarrow s_n} \quad \frac{\frac{T_n}{s_n \rightarrow^* t}}{\text{Refl}}}{\text{Tran}}}{s \rightarrow^* t} \text{Tran}$$

be a proof tree where $s_0 = s$ and $s_n \sim_{\mathcal{E}} t$. Given a term $s' \sim_{\mathcal{E}} s$,

$$\frac{\frac{\frac{T_1}{\tilde{s}_0 \rightarrow \tilde{s}_1} \quad \frac{\tilde{s}_1 \rightarrow \tilde{s}_2}{\vdots}}{\tilde{s}_1 \rightarrow^* t} \quad \frac{\frac{T_2}{\tilde{s}_{n-1} \rightarrow \tilde{s}_n} \quad \frac{\frac{T_n}{\tilde{s}_n \rightarrow^* t}}{\text{Refl}}}{\text{Tran}}}{s' \rightarrow^* t} \text{Tran}$$

is a proof tree with $\tilde{s}_0 = s'$. Here, for any u , the expression \tilde{u} denotes some term with $\tilde{u} \sim_{\mathcal{E}} u$. The terms \tilde{s}_i are given by Lemma 10.6. Notice that $\tilde{s}_n \sim_{\mathcal{E}} t$ since $\tilde{s}_n \sim_{\mathcal{E}} s_n$ and $s_n \sim_{\mathcal{E}} t$. □

Appendix A. Proofs

proof tree for $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ is obtained:

$$\frac{\frac{\text{Repl}}{s \rightarrow t} \quad \frac{\text{Refl}}{t \rightarrow^* t}}{s \rightarrow^* t} \text{Tran}$$

Otherwise, $\mathcal{U}(\mathcal{R})$ contains rules of the form (1), (2), and (3) from Definition 10.14. Using these, proof trees for $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ with the following head goals are constructed:

$$\begin{array}{lll} U_n^\rho(t_n, x_n^*)\sigma & \rightarrow^* & r\sigma \quad (G_n) \\ U_n^\rho(s_n, x_n^*)\sigma & \rightarrow^* & r\sigma \quad (H_n) \\ U_{n-1}^\rho(t_{n-1}, x_{n-1}^*)\sigma & \rightarrow^* & r\sigma \quad (G_{n-1}) \\ U_{n-1}^\rho(s_{n-1}, x_{n-1}^*)\sigma & \rightarrow^* & r\sigma \quad (H_{n-1}) \\ & \vdots & \\ U_1^\rho(t_1, x_1^*)\sigma & \rightarrow^* & r\sigma \quad (G_1) \\ U_1^\rho(s_1, x_1^*)\sigma & \rightarrow^* & r\sigma \quad (H_1) \\ s & \rightarrow^* & r\sigma \quad (K) \end{array}$$

For the following, notice that $\varphi\sigma$ is \mathcal{Th} -valid by assumption.

1. Proof tree for (G_n) :

Using rule (3) from Definition 10.14, the following proof tree is constructed:

$$\frac{\frac{\text{Repl}}{U_n^\rho(t_n, x_n^*)\sigma \rightarrow r\sigma} \quad \frac{\text{Refl}}{r\sigma \rightarrow^* r\sigma}}{U_n^\rho(t_n, x_n^*)\sigma \rightarrow^* r\sigma} \text{Tran}$$

2. Proof tree for (H_k) using a proof tree for (G_k) :

Assume that a proof tree T_k for the goal $(G_k) = U_k^\rho(t_k\sigma, x_k^*\sigma) \rightarrow^* r\sigma$ has already been constructed. By induction on the tree structure, it can furthermore be assumed that for the subtree

$$U_k = \frac{S_k}{s_k\sigma \rightarrow^* t_k\sigma}$$

Appendix A. Proofs

a transformed tree $\beta(U_k)$ of the form

$$\begin{array}{c}
 \frac{\frac{\frac{\frac{\frac{u_0 \rightarrow u_1}{T_k^1}}{u_1 \rightarrow u_2}}{u_1 \rightarrow^* t_k \sigma}}{u_1 \rightarrow^* t_k \sigma}}{s_k \sigma \rightarrow^* t_k \sigma} \\
 \text{Tran} \\
 \frac{\frac{\frac{\frac{\frac{u_{l-1} \rightarrow u_l}{T_k^2}}{u_{l-1} \rightarrow u_l}}{u_l \rightarrow^* t_k \sigma}}{u_l \rightarrow^* t_k \sigma}}{u_l \rightarrow^* t_k \sigma} \\
 \text{Tran} \\
 \frac{\frac{\frac{\frac{u_{l-1} \rightarrow u_l}{T_k^l}}{u_{l-1} \rightarrow u_l}}{u_l \rightarrow^* t_k \sigma}}{u_l \rightarrow^* t_k \sigma}}{u_l \rightarrow^* t_k \sigma} \\
 \text{Refl} \\
 \text{Tran} \\
 \text{Tran} \\
 \text{Tran}
 \end{array}$$

with $u_0 = s_k \sigma$ and $u_l \sim_{\mathcal{E}} t_k \sigma$ exists. Then, a proof tree for the goal

$U_k^\rho(s_k \sigma, x_k^* \sigma) \rightarrow^* r \sigma$ can be constructed as

$$\begin{array}{c}
 \frac{\frac{\frac{\frac{\frac{u'_0 \rightarrow u'_1}{T_k^1}}{u'_1 \rightarrow u'_2}}{u'_1 \rightarrow^* r \sigma}}{u'_1 \rightarrow^* r \sigma}}{u'_1 \rightarrow^* r \sigma} \\
 \text{Tran} \\
 \frac{\frac{\frac{\frac{\frac{u'_{l-1} \rightarrow u'_l}{T_k'^2}}{u'_{l-1} \rightarrow u'_l}}{u'_l \rightarrow^* r \sigma}}{u'_l \rightarrow^* r \sigma}}{u'_l \rightarrow^* r \sigma} \\
 \text{Tran} \\
 \frac{\frac{\frac{\frac{\frac{u'_{l-1} \rightarrow u'_l}{T_k'^l}}{u'_{l-1} \rightarrow u'_l}}{u'_l \rightarrow^* r \sigma}}{u'_l \rightarrow^* r \sigma}}{u'_l \rightarrow^* r \sigma}}{u'_l \rightarrow^* r \sigma} \\
 \text{Tran} \\
 \frac{\frac{\frac{\frac{\frac{u'_0 \rightarrow u'_1}{T_k^1}}{u'_1 \rightarrow u'_2}}{u'_1 \rightarrow^* r \sigma}}{u'_1 \rightarrow^* r \sigma}}{u'_1 \rightarrow^* r \sigma}}{u'_1 \rightarrow^* r \sigma} \\
 \text{Tran} \\
 \frac{\frac{\frac{\frac{\frac{u'_{l-1} \rightarrow u'_l}{T_k'^2}}{u'_{l-1} \rightarrow u'_l}}{u'_l \rightarrow^* r \sigma}}{u'_l \rightarrow^* r \sigma}}{u'_l \rightarrow^* r \sigma}}{u'_l \rightarrow^* r \sigma} \\
 \text{Tran} \\
 \frac{\frac{\frac{\frac{\frac{u'_{l-1} \rightarrow u'_l}{T_k'^l}}{u'_{l-1} \rightarrow u'_l}}{u'_l \rightarrow^* r \sigma}}{u'_l \rightarrow^* r \sigma}}{u'_l \rightarrow^* r \sigma}}{u'_l \rightarrow^* r \sigma} \\
 \text{Tran} \\
 U_k^\rho(s_k \sigma, x_k^* \sigma) \rightarrow^* r \sigma
 \end{array}$$

where $u'_i = U_k^\rho(u_i, x_k^* \sigma)$ and $T_k'^i$ ‘‘corresponds’’ to T_k^i .

3. Proof tree for (G_{k-1}) using a proof tree for (H_k) :

Assume that a proof tree T_k for the goal $(H_k) = U_k^\rho(s_k \sigma, x_k^* \sigma) \rightarrow^* r \sigma$ has already been constructed. Then, a proof tree for $U_{k-1}^\rho(t_{k-1} \sigma, x_{k-1}^* \sigma) \rightarrow^* r \sigma$ can be constructed as

$$\frac{\frac{U_{k-1}^\rho(t_{k-1} \sigma, x_{k-1}^* \sigma) \rightarrow U_k^\rho(s_k \sigma, x_k^* \sigma)}{\text{Repl}}}{U_{k-1}^\rho(t_{k-1} \sigma, x_{k-1}^* \sigma) \rightarrow^* r \sigma} T_k \text{ Tran}$$

where the Repl-step uses rule (2) from Definition 10.14.

4. Proof tree for (K) using a proof tree for (H_1) :

Appendix A. Proofs

Assume that a proof tree T_1 for the goal $(H_1) = U_1^\rho(s_1\sigma, x_1^*\sigma) \rightarrow^* r\sigma$ has already been constructed. Then, a proof tree for the goal $s \rightarrow^* t$ can be constructed as

$$\frac{\frac{}{s \rightarrow U_1^\rho(s_1\sigma, x_1^*\sigma)} \text{Repl} \quad T_1}{s \rightarrow^* t} \text{Tran}$$

where the **Repl**-step uses rule (1) from Definition 10.14.

As in case I, if the original proof tree is not closed, then the transformed tree is cut at some level. In either case, $\beta(T)$ is well-formed if T is well-formed and $\beta(T) \subset \beta(T')$ if $T \subset T'$. \square

Proof of Theorem 10.17. Assume that $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is not operationally terminating. Thus, there exists an infinite proof tree $\{T_i\}_{i \geq 0}$ for $(\mathcal{R}, \mathcal{S}, \mathcal{E})$. By Lemma 10.16, there exists an infinite sequence $\{\beta(T_i)\}_{i \geq 0}$ of proof trees for $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$. Additionally, $T_i \subset T_{i+1}$ implies $\beta(T_i) \subset \beta(T_{i+1})$ for all $i \geq 0$. Therefore, $\{\beta(T_i)\}_{i \geq 0}$ is an infinite proof tree for $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$, which is thus not operationally terminating. \square

Proof of Corollary 10.18. By Lemma 10.13, termination of $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ implies operational termination of $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ since $\mathcal{U}(\mathcal{R})$ is unconditional. Thus, $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is operationally terminating by Theorem 10.17. \square

A.10 Proofs from Chapter 11

Proof of Lemma 11.10.

1. Let $C[f(s^*)] \sim_{\mathcal{E}} t$, i.e., there exist terms t_0, \dots, t_n with $n \geq 0$ such that $C[f(s^*)] = t_0 \vdash_{\mathcal{E}} t_1 \vdash_{\mathcal{E}} \dots \vdash_{\mathcal{E}} t_n = t$. The claim is proved by induction on n . If $n = 0$ then $C[f(s^*)] = t$ and the claim is obvious.

Appendix A. Proofs

If $n > 0$, the inductive assumption implies $t_{n-1} = C''[f(s''^*)]$ with $C'' \sim_{\mathcal{E}} C$ and $s''^* \sim_{\mathcal{E}} s^*$, where additionally C'' is active. Since $t_{n-1} \vdash_{\mathcal{E}} t_n$, there exists an equation $u \approx v$ (or $v \approx u$) in \mathcal{E} such that $t_{n-1}|_p = u\sigma$ and $t_n = t_{n-1}[v\sigma]_p$ for some position p and some substitution σ . Let q be the position with $t_{n-1}|_q = f(s''^*)$, i.e., $C''|_q = \square$. Now perform a case analysis on the relationship between the positions p and q .

Case 1: $p \parallel q$. Then, $t_n = t_{n-1}[v\sigma]_p = (C''[f(s''^*)])[v\sigma]_p = (C''[v\sigma]_p)[f(s''^*)]$ with $C''[v\sigma]_p \sim_{\mathcal{E}} C''[u\sigma]_p = C''$ and $C''[v\sigma]_p$ is active.

Case 2: $p = q.q'$ for some position $q' \neq \Lambda$. In this case, $t_n = t_{n-1}[v\sigma]_p = (C''[f(s''^*)])[v\sigma]_{q.q'} = C''[f(s''^*)][v\sigma]_{q'}$. Since $q' \neq \Lambda$, the position q' can be written as $q' = i.q''$ for some $1 \leq i \leq \text{arity}(f)$ and some position q'' . Then $s'_j = s''_j$ if $i \neq j$ and $s'_i = s''_i[v\sigma]_{q''} \sim_{\mathcal{E}} s''_i[u\sigma]_{q''} = s''_i$, i.e., $s'^* \sim_{\mathcal{E}} s''^*$.

Case 3: $q = p.p'$ for some position p' (possibly $p' = \Lambda$). Since $f \notin \mathcal{F}(\mathcal{E})$, the position p' can be written as $p' = p'_1.p'_2$ such that $u|_{p'_1}$ is a variable x and $x\sigma|_{p'_2} = f(s''^*)$. Since the equation $u \approx v$ (or $v \approx u$) is i.u.v., there exists a unique position p''_1 in v such that $v|_{p''_1} = x$. This implies $v\sigma|_{p''_1.p'_2} = x\sigma|_{p'_2} = f(s''^*)$. Define the substitution σ' by $\sigma'(y) = \sigma(y)$ for $y \neq x$ and $\sigma'(x) = x\sigma|_{p''_1.p'_2}$. Let $C' = (t_{n-1}[v\sigma]_p)[\square]_{p.p''_1.p'_2} = t_{n-1}[v\sigma[\square]_{p''_1.p'_2}]_p = t_{n-1}[v\sigma']_p \sim_{\mathcal{E}} t_{n-1}[u\sigma']_p = C''$. Since q is active in C'' , the position p'_1 is active in u and Definition 11.7.2a implies that p''_1 is active in v . Therefore, C' is active as well. Thus, $t_n = t_{n-1}[v\sigma]_p = C'[f(s''^*)]$ and the claim follows.

2. Let $s' \sim_{\mathcal{E}} s \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel_{\mathcal{E}} \mathcal{R}, \mu} t$. This means that $s = C[f(u^*)]$ for some active context C with $f(u^*) \xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} l\sigma$ for some constrained rewrite rule $l \rightarrow r[\varphi] \in \mathcal{R}$ and some \mathcal{Th} -based substitution σ such that $\varphi\sigma$ is \mathcal{Th} -valid and $t = C[r\sigma]$. Since $s \sim_{\mathcal{E}} s'$, Lemma 11.10.1 implies $s' = C'[f(u^*)]$ for some active context C' with $C' \sim_{\mathcal{E}} C$ and $u^* \sim_{\mathcal{E}} u^*$. Therefore, $f(u^*) \xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} l\sigma$ by Lemma 2.28 and the substitution σ can be used to rewrite $s' = C'[f(u^*)]$ to $t' = C'[r\sigma] \sim_{\mathcal{E}}$

Appendix A. Proofs

$$C[r\sigma] = t.$$

3. Let $s \rightarrow_{\mathcal{E}\setminus\mathcal{S}} t \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} u$, i.e., there exist positions $p_1 \in \mathcal{Pos}(s)$ and $p_2 \in \mathcal{Pos}^\mu(t)$, rules $l_1 \rightarrow r_1 \in \mathcal{S}$ and $l_2 \rightarrow r_2[\varphi_2] \in \mathcal{R}$, a substitution σ_1 , and a \mathcal{Th} -based substitution σ_2 such that

- (a) $s|_{p_1} \sim_{\mathcal{E}} l_1\sigma_1$ and $t = s[r_1\sigma_1]_{p_1}$, and
- (b) $t|_{p_2} \xrightarrow{\geq\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \gtrsim_{\mathcal{E}}^{\Lambda} l_2\sigma_2$, the instantiated \mathcal{Th} -constraint $\varphi_2\sigma_2$ is \mathcal{Th} -valid, and $u = t[r_2\sigma_2]_{p_2}$.

Next, a case analysis on the relationship between p_1 and p_2 is performed.

Case 1: $p_1 \parallel p_2$. In this case $s \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}} u$ is immediate.

Case 2: $p_1 = p_2.q$ for some position $q \neq \Lambda$. In this case $s|_{p_2} = f(s^*)$, $t|_{p_2} = f(t^*)$, and $s^* \rightarrow_{\mathcal{E}\setminus\mathcal{S}} t^*$. Therefore, $f(s^*) \xrightarrow{\geq\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \gtrsim_{\mathcal{E}}^{\Lambda} l_2\sigma_2$ since $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ is \mathcal{E} -convergent and strongly \mathcal{E} -coherent and $f(t^*) \xrightarrow{\geq\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \gtrsim_{\mathcal{E}}^{\Lambda} l_2\sigma_2$. Thus, $s \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} s[r_2\sigma_2]_{p_2} = t[r_2\sigma_2]_{p_2} = u$.

Case 3: $p_2 = p_1.q$ for some position q , possibly $q = \Lambda$. Since r_1 does not contain symbols from $\mathcal{D}(\mathcal{R})$, there exists a position $q_1 \in \mathcal{Pos}^\mu(r_1)$ such that $r_1|_{q_1} = x$ is a variable and $q = q_1.q_2$ for some position q_2 . Define the substitution σ'_1 to behave like σ_1 , with the exception that $\sigma'_1(x) = \sigma_1(x)[r_2\sigma_2]_{q_2}$. Then $l_1\sigma_1 \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} l_1\sigma'_1$ using Definition 11.7.3a and thus $s|_{p_1} \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \sim_{\mathcal{E}} l_1\sigma'_1$ by Lemma 11.10.2 since $s|_{p_1} \sim_{\mathcal{E}} l_1\sigma_1$. Thus, $s|_{p_1} \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}} r_1\sigma'$. Since r_1 is linear, $s[r_1\sigma'_1]_{p_1} = t[r_1\sigma'_1]_{p_1} = u$ and thus $s \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}} u$. \square

Proof of Corollary 11.11.

1. Assume that s starts an infinite $\xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction

$$s \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} s_1 \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} s_2 \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} s_3 \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} \dots$$

Using Lemma 11.10.2 this implies

$$t \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} t_1 \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} t_2 \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} t_3 \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} \dots$$

Appendix A. Proofs

where $s_i \sim_{\mathcal{E}} t_i$, i.e., t starts an infinite $\xrightarrow{S}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction as well. The other direction is shown the same way.

2. Let $s \rightarrow_{\mathcal{E}\setminus\mathcal{S}} t$ and assume that t starts an infinite $\xrightarrow{S}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction. Using the inclusion $\rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \xrightarrow{S}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu} \subseteq \xrightarrow{S}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ from Lemma 11.10.3 repeatedly produces an infinite $\xrightarrow{S}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction starting with s . \square

Proof of Lemma 11.22.

1. If $t \sim_{\mathcal{E}} t'$ and $t \in \mathcal{M}_{(\mathcal{R},\mathcal{S},\mathcal{E},\mu)}^{\infty}$, then t' starts an infinite $\xrightarrow{S}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction by Corollary 11.11.1. Notice that $\text{root}(t) \in \mathcal{D}(\mathcal{R})$, which implies $t \gtrsim_{\mathcal{E}}^{\Lambda} t'$, i.e., $t = f(t_1, \dots, t_n)$ and $t' = f(t'_1, \dots, t'_n)$ where $t_i \sim_{\mathcal{E}} t'_i$ for all $1 \leq i \leq n$. Assume $t' \triangleright_{\mu} s$ such that s starts an infinite $\xrightarrow{S}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction. Then $t'_i \triangleright_{\mu} s$ for some $1 \leq i \leq n$ and t'_i starts an infinite $\xrightarrow{S}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction as well. But then t_i starts an infinite $\xrightarrow{S}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction by Corollary 11.11.1, contradicting $t \in \mathcal{M}_{(\mathcal{R},\mathcal{S},\mathcal{E},\mu)}^{\infty}$.
2. By the assumption, t' starts an infinite $\xrightarrow{S}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction. Notice that $\text{root}(t) \in \mathcal{D}(\mathcal{R})$ and thus $t \xrightarrow{\geq \Lambda}_{\mathcal{E}\setminus\mathcal{S}}^* t'$, i.e., $t = f(t_1, \dots, t_n)$ and $t' = f(t'_1, \dots, t'_n)$ where $t_i \rightarrow_{\mathcal{E}\setminus\mathcal{S}}^* t'_i$ for all $1 \leq i \leq n$. Assume $t' \triangleright_{\mu} s$ such that s starts an infinite $\xrightarrow{S}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction. Then $t'_i \triangleright_{\mu} s$ for some $1 \leq i \leq n$ and t'_i starts an infinite $\xrightarrow{S}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction as well. But then t_i starts an infinite $\xrightarrow{S}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu}$ -reduction by Corollary 11.11.2, contradicting $t \in \mathcal{M}_{(\mathcal{R},\mathcal{S},\mathcal{E},\mu)}^{\infty}$. \square

Proof of Lemma 11.23.

1. If $C[t] \vdash_{\mathcal{E}} s$ at a position $p \in \mathcal{Pos}(C)$, then there exist a substitution σ and an equation $u \approx v$ (or $v \approx u$) in \mathcal{E} such that $C[t]_p = u\sigma$ and $s = C[t][v\sigma]_p$. Now, the proof proceeds with an induction on p .

If $p = \Lambda$, then, since $\text{root}(t) \in \mathcal{D}(\mathcal{R})$, there is a variable $x \in \mathcal{V}(u)$ with $u \triangleright_{\mu} x$ and $x\sigma \triangleright_{\mu} t$. Now Definition 11.7.2a implies $v \triangleright_{\mu} x$. Let q be the position of x in v and let q' be the position of t in $x\sigma$. Then $s = v\sigma = C'[t]$ with $C' = v\sigma[\square]_{q,q'}$,

Appendix A. Proofs

where C' is a hiding context.

If $p \neq \Lambda$, then $C[t] = f(t_1, \dots, t_{i-1}, C_1[t], t_{i+1}, \dots, t_n)$ for a hiding context C_1 . If p is inside one of the t_j , then $s = f(t'_1, \dots, t'_{i-1}, C_1[t], t'_{i+1}, \dots, t'_n) = C'[t]$ where C' is clearly a hiding context. Otherwise, $C_1[t] \vdash_{\mathcal{E}} s'$ and $s = f(t_1, \dots, t_{i-1}, s', t_{i+1}, \dots, t_n)$. The inductive assumption implies $s' = C'_1[t]$ for a hiding context C'_1 , which clearly implies $s = C'[t]$ for a hiding context C' .

2. If $C[t] \rightarrow_{\mathcal{S}} s$ at a position $p \in \text{Pos}(C)$, then there exists a substitution σ and a rule $l \rightarrow r \in \mathcal{S}$ such that $C[t]|_p = l\sigma$ and $s = C[t][r\sigma]_p$. Now, the proof proceeds with an induction on p .

If $p = \Lambda$, then, since $\text{root}(t) \in \mathcal{D}(\mathcal{R})$, there is a variable $x \in \mathcal{V}(l)$ with $l \succeq_{\mu} x$ and $x\sigma \succeq_{\mu} t$. By the assumption, $r \succeq_{\mu} x$. Let q be the position of x in r and let q' be the position of t in $x\sigma$. Then $s = r\sigma = C'[t]$ with $C' = r\sigma[\square]_{q,q'}$, where C' is a hiding context.

If $p \neq \Lambda$, then $C[t] = f(t_1, \dots, t_{i-1}, C_1[t], t_{i+1}, \dots, t_n)$ for a hiding context C_1 . If p is inside one of the t_j , then $s = f(t'_1, \dots, t'_{i-1}, C_1[t], t'_{i+1}, \dots, t'_n) = C'[t]$ where C' is clearly a hiding context. Otherwise, $C_1[t] \rightarrow_{\mathcal{S}} s'$ and $s = f(t_1, \dots, t_{i-1}, s', t_{i+1}, \dots, t_n)$. The inductive assumption implies $s' = C'_1[t]$ for a hiding context C'_1 , which clearly implies $s = C'[t]$ for a hiding context C' . \square

Proof of Lemma 11.25.1. Let $w \triangleright_{-\mu} s \succeq_{\mu} t$ with $t \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^{\infty}$. Thus $v \triangleright_{-\mu} s \succeq_{\mu} t$ since $v \succeq_{\mu} w \triangleright_{-\mu} s$. The claim is shown by performing a case analysis on whether s is a subterm of u .

1. $u \triangleright s$.

- (a) If $u \triangleright_{\mu} s$, then $u \triangleright_{\mu} s \succeq_{\mu} t$ contradicts $u \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^{\infty}$ since $u \triangleright_{\mu} t$ with $t \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^{\infty}$.

- (b) Otherwise, $u \triangleright_{-\mu} s$. Then $u \triangleright_{-\mu} s \succeq_{\mu} t$ and the claim follows from the hiding property of u .

Appendix A. Proofs

2. $u \not\preceq s$.

Then $u|_p = l\sigma$ and $v = u[r\sigma]_p$ for a position $p \in \text{Pos}(u)$, a substitution σ , and an equation $l \approx r$ (or $r \approx l$) in \mathcal{E} . Now $l\sigma \not\preceq s$ since $u \not\preceq s$ and, in particular, $\sigma(x) \not\preceq s$ for all $x \in \mathcal{V}(l)$. Also, u does not contain s as a subterm at a position \hat{p} with $p \parallel \hat{p}$.

(a) First, assume that $r\sigma \succeq_\mu s$. Since $\text{root}(t) \notin \mathcal{F}(\mathcal{E})$, $\sigma(x) \succeq_\mu t$ for some variable $x \in \mathcal{V}(r)$. Let q be the outermost position in v above s that satisfies $v|_q \succeq_\mu s$.

(i) First, assume that q is above p or $q = p$. Since $r\sigma \succeq_\mu s \succeq_\mu t$, Definition 11.7.2a implies $l\sigma \succeq_\mu t$, i.e., $u \triangleright_{-\mu} l\sigma \succeq_\mu t$. From the hiding property of u , $l\sigma = C[t]$ for a hiding context C and there exists an instance t' of a hidden term such that $t' \xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}}^* \circ \gtrsim_{\mathcal{E}}^\Lambda t$. Then $r\sigma = C'[t]$ where C' is a hiding context by Lemma 11.23.1. Now $r\sigma \succeq_\mu s$ implies $s = C''[t]$, where C'' is a hiding context.

(ii) Otherwise, $q = p.p'$ for a position $p' \neq \Lambda$ such that $r|_{p'} \notin \mathcal{V}$. Then $r|_{p'} \triangleright_\mu x$, and Definition 11.7.2b (or Definition 11.7.2c) implies $l \triangleright_{-\mu} l' \succeq_\mu x$ for some non-variable subterm l' of l . Therefore, $u \triangleright_{-\mu} l'\sigma \succeq_\mu t$ and the hiding property of u implies $l'\sigma = C[t]$ for a hiding context C and there exists an instance t' of a hidden term such that $t' \xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}}^* \circ \gtrsim_{\mathcal{E}}^\Lambda t$. Then, $r|_{p'}\sigma = r\sigma|_{p'} = C'[t]$ for some hiding context C' and $r\sigma|_{p'} \succeq_\mu s$ implies $s = C''[t]$, where C'' is a hiding context.

(b) If $r\sigma \not\preceq s$, then $s \triangleright r\sigma$, i.e., $s|_{q_1} = r\sigma$ for some position q_1 . In this case, $u \triangleright_{-\mu} s[l\sigma]_{q_1}$.

(i) If t occurs above q_1 , then $s|_{q_2} = t$ and $q_1 = q_2.q_3$ for some position $q_3 \neq \Lambda$. Thus, $u \triangleright_{-\mu} s[l\sigma]_{q_1} \triangleright_\mu t[l\sigma]_{q_3}$ and the hiding property of u gives $s[l\sigma]_{q_1} = C[t[l\sigma]_{q_3}]$ for a hiding context C and there exists an instance t' of a hidden term such that $t' \xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}}^* \circ \gtrsim_{\mathcal{E}}^\Lambda t[l\sigma]_{q_3}$. Thus $t' \xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}}^* \circ \gtrsim_{\mathcal{E}}^\Lambda t$ since $t[l\sigma]_{q_3} \gtrsim_{\mathcal{E}}^\Lambda t[r\sigma]_{q_3} = t$. Furthermore, $s = s[r\sigma]_{q_1} = C[t]$.

Appendix A. Proofs

- (ii) Otherwise, $\sigma(x) \triangleright_{\mu} t$ for some variable $x \in \mathcal{V}(r)$. In this case, $u \triangleright_{-\mu} s[l\sigma]_{q_1} \triangleright_{\mu} t$ and the hiding property of u implies $s[l\sigma]_{q_1} = C[t]$ for a hiding context C and there exists an instance t' of a hidden term such that $t' \xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}} \circ \overset{\geq \Lambda}{\sim}_{\mathcal{E}} t$. Thus, $s = s[r\sigma]_{q_1} = C'[t]$ for some context C' . The context C' is hiding by Lemma 11.23.1. \square

Proof of Lemma 11.25.2. If $u \rightarrow_{\mathcal{E} \setminus \mathcal{S}} v \triangleright_{\mu} w$, then $u \sim_{\mathcal{E}} u' \rightarrow_{\mathcal{S}} v \triangleright_{\mu} w$. By Lemma 11.22, $u' \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^{\infty}$, and Lemma 11.25.1 implies that u' has the hiding property. Let $w \triangleright_{-\mu} s \triangleright_{\mu} t$ with $t \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^{\infty}$. Thus $v \triangleright_{-\mu} s \triangleright_{\mu} t$ since $v \triangleright_{\mu} w \triangleright_{-\mu} s$. Perform a case analysis on whether s is a subterm of u' .

1. $u' \triangleright s$.

- (a) If $u' \triangleright_{\mu} s$, then $u' \triangleright_{\mu} s \triangleright_{\mu} t$ contradicts $u' \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^{\infty}$ since $u' \triangleright_{\mu} t$ with $t \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^{\infty}$.
- (b) Otherwise, $u' \triangleright_{-\mu} s$. Then $u' \triangleright_{-\mu} s \triangleright_{\mu} t$ and the claim follows from the hiding property of u' .

2. $u' \not\triangleright s$.

Then $u'|_p = l\sigma$ and $v = u'[r\sigma]_p$ for a position $p \in \mathcal{Pos}(u')$, a substitution σ , and a rule $l \rightarrow r \in \mathcal{S}$. Now $l\sigma \not\triangleright s$ since $u' \not\triangleright s$ and, in particular, $\sigma(x) \not\triangleright s$ for all $x \in \mathcal{V}(l)$. Also, u' does not contain s as a subterm at a position \hat{p} with $p \parallel \hat{p}$.

- (a) First, assume that $r\sigma \triangleright s$. Since $\text{root}(t) \notin \mathcal{F}(\mathcal{E})$, $\sigma(x) \triangleright_{\mu} t$ for some variable $x \in \mathcal{V}(r)$. Let q be the outermost position in v above s that satisfies $v|_q \triangleright_{\mu} s$.
- (i) First, assume that q is above p or $q = p$. Since $r\sigma \triangleright_{\mu} s \triangleright_{\mu} t$, Definition 11.7.3a implies $l\sigma \triangleright_{\mu} t$, i.e., $u \triangleright_{-\mu} l\sigma \triangleright_{\mu} t$. From the hiding property of u' , $l\sigma = C[t]$ for a hiding context C and there exists an instance t' of a hidden term such that $t' \xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}} \circ \overset{\geq \Lambda}{\sim}_{\mathcal{E}} t$. Then $r\sigma = C'[t]$ where C' is a hiding context by Lemma 11.23.2. Now $r\sigma \triangleright_{\mu} s$ implies $s = C''[t]$, where C'' is a hiding context.

Appendix A. Proofs

- (ii) Otherwise, $q = p.p'$ for a position $p' \neq \Lambda$ such that $r|_{p'} \notin \mathcal{V}$. Then $r|_{p'} \triangleright_{\mu} x$, and Definition 11.7.3b implies $l \triangleright_{-\mu} l' \succeq_{\mu} x$ for some non-variable subterm l' of l . Therefore, $u' \triangleright_{-\mu} l' \sigma \succeq_{\mu} t$ and the hiding property of u' implies $l' \sigma = C[t]$ for a hiding context C and there exists an instance t' of a hidden term such that $t' \xrightarrow{\succ_{\mathcal{E} \setminus \mathcal{S}}^{\Lambda}} * \circ \overset{\succ_{\mathcal{E}}}{\sim} t$. Then, $r|_{p'} \sigma = r\sigma|_{p'} = C'[t]$ for some hiding context C' and $r\sigma|_{p'} \succeq_{\mu} s$ implies $s = C''[t]$, where C'' is a hiding context.
- (b) If $r\sigma \not\preceq s$, then $s \triangleright r\sigma$, i.e., $s|_{q_1} = r\sigma$ for some position q_1 . In this case, $u' \triangleright_{-\mu} s[l\sigma]_{q_1}$.
- (i) If t occurs above q_1 , then $s|_{q_2} = t$ and $q_1 = q_2.q_3$ for some position $q_3 \neq \Lambda$. Thus, $u' \triangleright_{-\mu} s[l\sigma]_{q_1} \triangleright_{\mu} t[l\sigma]_{q_3}$. Now the hiding property of u' gives $s[l\sigma]_{q_1} = C[t[l\sigma]_{q_3}]$ for a hiding context C and there exists an instance t' of a hidden term such that $t' \xrightarrow{\succ_{\mathcal{E} \setminus \mathcal{S}}^{\Lambda}} * \circ \overset{\succ_{\mathcal{E}}}{\sim} t[l\sigma]_{q_3}$. Thus $t' \xrightarrow{\succ_{\mathcal{E} \setminus \mathcal{S}}^{\Lambda}} * \circ \overset{\succ_{\mathcal{E}}}{\sim} t$ since $t[l\sigma]_{q_3} \xrightarrow{\succ_{\mathcal{S}}^{\Lambda}} t[r\sigma]_{q_3} = t$ and $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ is strongly \mathcal{E} -coherent by Definition 3.12. Furthermore, $s = s[r\sigma]_{q_1} = C[t]$.
- (ii) Otherwise, $\sigma(x) \triangleright_{\mu} t$ for some variable $x \in \mathcal{V}(r)$. In this case, $u' \triangleright_{-\mu} s[l\sigma]_{q_1} \succeq_{\mu} t$ and the hiding property of u' implies $s[l\sigma]_{q_1} = C[t]$ for a hiding context C and there exists an instance t' of a hidden term such that $t' \xrightarrow{\succ_{\mathcal{E} \setminus \mathcal{S}}^{\Lambda}} * \circ \overset{\succ_{\mathcal{E}}}{\sim} t$. Thus, $s = s[r\sigma]_{q_1} = C'[t]$ for some context C' . The context C' is hiding by Lemma 11.23.2. \square

Proof of Lemma 11.25.3. If $u \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}, \mu} v \succeq_{\mu} w$, then $u = C[u']$ for an active context C such that $u' \xrightarrow{\succ_{\mathcal{E} \setminus \mathcal{S}}^{\Lambda}} * \circ \overset{\succ_{\mathcal{E}}}{\sim} l\sigma \rightarrow_{\mathcal{R}} r\sigma$ for a constrained rewrite rule $l \rightarrow r[\varphi] \in \mathcal{R}$ and a \mathcal{Th} -based substitution σ . Since $C[l\sigma]$ starts an infinite $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}, \mu}$ -reduction, Lemma 11.22 implies $C[l\sigma] \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^{\infty}$. Also, Lemmas 11.25.1 and 11.25.2 imply that $C[l\sigma]$ has the hiding property. Let $w \triangleright_{-\mu} s \succeq_{\mu} t$ with $t \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^{\infty}$. Then $v \triangleright_{-\mu} s \succeq_{\mu} t$ since $v \succeq_{\mu} w \triangleright_{-\mu} s$. Perform a case analysis on whether s is a subterm of $C[l\sigma]$.

Appendix A. Proofs

1. $C[l\sigma] \triangleright s$.

- (a) If $C[l\sigma] \triangleright_\mu s$, then $C[l\sigma] \triangleright_\mu s \not\subseteq_\mu t$ contradicts $C[l\sigma] \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^\infty$ since $C[l\sigma] \triangleright_\mu t$ with $t \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^\infty$.
- (b) Otherwise, $C[l\sigma] \triangleright_{\neg\mu} s$. Then $C[l\sigma] \triangleright_{\neg\mu} s \subseteq_\mu t$ and the claim follows from the hiding property of $C[l\sigma]$.

2. $C[l\sigma] \not\triangleright s$.

First, notice that $v = C[r\sigma]$. Thus, $C \not\triangleright s$ and $l\sigma \not\triangleright s$ since $C[l\sigma] \not\triangleright s$. Also, in particular, $\sigma(x) \not\triangleright s$ for all $x \in \mathcal{V}(l)$. Finally, the root of s in v cannot be above \square in C since these positions are active. Hence, $v \triangleright_{\neg\mu} s$ implies $r\sigma \triangleright_{\neg\mu} s$ such that s is an instance of a non-variable subterm of r occurring at a position from $\mathcal{Pos}^{-\mu}(r)$, i.e., $r \triangleright_{\neg\mu} r'$ with $r' \notin \mathcal{V}$ such that $r'\sigma = s$ and $s|_p = t$ for an active position $p \in \mathcal{Pos}^\mu(s)$.

- (a) First, assume $p \in \mathcal{Pos}^\mu(r')$ and $r'|_p \notin \mathcal{V}$. Then $r'|_p$ is a hidden term and $t = s|_p = r'\sigma|_p = r'|_p\sigma$ is an instance of a hidden term. Furthermore, $C' = r'[\square]_p$ is a hiding context, which implies that $C'\sigma$ is a hiding context as well. Thus, $s = r'\sigma = C'[r'|_p]\sigma = C'\sigma[r'|_p\sigma]$ and the claim follows.
- (b) Otherwise, $p = p_1.p_2$ such that $r'|_{p_1} = x$ for a variable $x \in \mathcal{V}$. In this case, $C' = r'[\square]_{p_1}$ is a hiding context. Now $l\sigma \triangleright t$ since $x\sigma \subseteq_\mu t$. If $l\sigma \triangleright_\mu t$, then $C[l\sigma] \subseteq_\mu l\sigma \triangleright_\mu x\sigma \subseteq_\mu t$ with $t \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^\infty$, contradicting $C[l\sigma] \in \mathcal{M}_{(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)}^\infty$. Thus, $l\sigma \triangleright_{\neg\mu} x\sigma$ and $C[l\sigma] \triangleright_{\neg\mu} x\sigma \subseteq_\mu t$ and the hiding property of $C[l\sigma]$ implies $x\sigma = \widehat{C}[t]$ for a hiding context \widehat{C} and there exists an instance t' of a hidden term such that $t' \xrightarrow{\geq \Lambda}_{\mathcal{E} \setminus \mathcal{S}} \circ \xrightarrow{\geq \Lambda}_{\mathcal{E}} t$. Let $C'' = C'\sigma[\widehat{C}]$. Then the context C'' is hiding and $s = r'\sigma = r'\sigma[x\sigma[t]_{p_2}]_{p_1} = C''\sigma[\widehat{C}[t]] = C''[t]$, thus establishing the claim. \square

Proof of Theorem 11.26. If $\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R}, \mu}$ is not terminating, then there exists a term t which starts an infinite $\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R}, \mu}$ -reduction such that every proper subterm of t (even at inactive positions) is terminating w.r.t. $\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R}, \mu}$. Thus, t trivially has the

Appendix A. Proofs

hiding property. Furthermore, there are terms t_i, s_i, \widehat{t}_i such that

$$t = \widehat{t}_1 \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu}^* t_1 \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} s_1 \succeq_\mu \widehat{t}_2 \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu}^* t_2 \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} s_2 \succeq_\mu \widehat{t}_3 \dots$$

where the $\xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu}$ -steps in $\widehat{t}_i \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu}^* t_i$ are applied strictly below the root position and the $\xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu}$ -step $t_i \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} s_i$ is applied at the root position. Furthermore, $\widehat{t}_i, t_i \in \mathcal{M}_{(\mathcal{R},\mathcal{S},\mathcal{E},\mu)}^\infty$ for all i .

First, it is shown that every \widehat{t}_i and every t_i has the hiding property. As shown above, $\widehat{t}_1 = t$ has the hiding property by assumption. Next, if \widehat{t}_i has the hiding property, then t_i has the hiding property by Lemma 11.25.3, and Lemma 11.25.3 furthermore implies that \widehat{t}_{i+1} has the hiding property if t_i has the hiding property.

Next, it is shown that $t_i^\# \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\text{DP}(\mathcal{R},\mu),\mu}^+ \widehat{t}_{i+1} \xrightarrow{\geq\Lambda}_{\mathcal{E}\setminus\mathcal{S}}^* \circ \gtrsim_{\mathcal{E}}^\Lambda \widehat{t}_{i+1}^\#$ for some \widehat{t}_{i+1} and that every term in this sequence is terminating w.r.t. $\xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu}$. Since $t_i \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} s_i \succeq_\mu \widehat{t}_{i+1}$ where the $\xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu}$ -step is applied at the root position, there exist a rule $l_i \rightarrow r_i[\![\varphi_i]\!] \in \mathcal{R}$, a \mathcal{Th} -based substitution σ , and a position $p_i \in \text{Pos}^\mu(s_i)$ such that $t_i \xrightarrow{\geq\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^\Lambda l_i\sigma$, $\varphi_i\sigma$ is \mathcal{Th} -valid, $s_i = r_i\sigma$, and $s_i|_{p_i} = \widehat{t}_{i+1}$. Perform a case analysis on the position p_i .

If $p_i \in \text{Pos}^\mu(r_i)$ and $r_i|_{p_i} \notin \mathcal{V}$, then there exists the dependency pair $l^\# \rightarrow (r_i|_{p_i})^\#[\![\varphi_i]\!] \in \text{DP}_\circ(\mathcal{R},\mu)$ since $\text{root}(r_i|_{p_i}) = \text{root}(\widehat{t}_{i+1}) \in \mathcal{D}(\mathcal{R})$. But this clearly implies $t_i^\# \xrightarrow{\geq\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^\Lambda l^\#\sigma$ and thus $t_i^\# \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\text{DP}(\mathcal{R},\mu),\mu} (r_i|_{p_i})^\#\sigma = \widehat{t}_{i+1}^\#$. Furthermore, $t_i^\#$ and $\widehat{t}_{i+1}^\#$ are terminating w.r.t. $\xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu}$ since $t_i, \widehat{t}_{i+1} \in \mathcal{M}_{(\mathcal{R},\mathcal{S},\mathcal{E},\mu)}^\infty$.

Otherwise, p_i is at or below the position of an active variable x_i in r_i . The variable x_i only occurs in inactive positions in l_i since $l_i\sigma \triangleright_\mu x_i\sigma \succeq_\mu \widehat{t}_{i+1}$ and $\widehat{t}_{i+1} \in \mathcal{M}_{(\mathcal{R},\mathcal{S},\mathcal{E},\mu)}^\infty$ implies that $l_i\sigma \notin \mathcal{M}_{(\mathcal{R},\mathcal{S},\mathcal{E},\mu)}^\infty$, which, by Lemma 11.22 contradicts $t_i \in \mathcal{M}_{(\mathcal{R},\mathcal{S},\mathcal{E},\mu)}^\infty$ because $t_i \xrightarrow{\geq\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^\Lambda l_i\sigma$. Since $l_i\sigma$ has the hiding property by Lemma 11.25 (because t_i has the hiding property) and $l_i\sigma \triangleright_{\neg\mu} x\sigma \succeq_\mu \widehat{t}_{i+1}$, Definition 11.24 implies $x\sigma = C[\widehat{t}_{i+1}]$ for a hiding context C and there exists an instance \widehat{t}'_{i+1} of a hidden term such that $\widehat{t}'_{i+1} \xrightarrow{\geq\Lambda}_{\mathcal{E}\setminus\mathcal{S}}^* \circ \gtrsim_{\mathcal{E}}^\Lambda \widehat{t}_{i+1}$. Notice that $C[\widehat{t}_{i+1}]$ is irreducible by

Appendix A. Proofs

$\xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}}$ since $l_i\sigma$ is irreducible by $\xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}}$. By letting s and s' be the appropriate sorts,

$$\begin{aligned}
t_i^\# &\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus DP(\mathcal{R},\mu),\mu} U_s(x_i\sigma) && \text{since } l_i^\# \rightarrow U_s(x_i)[\varphi_i] \in DP_u(\mathcal{R},\mu) \\
&= U_s(C[\widehat{t}_{i+1}]) && \text{where } C \text{ is hiding} \\
&\xrightarrow{\mathcal{S}^*}_{Th\|\mathcal{E}\setminus DP(\mathcal{R},\mu),\mu} U_{s'}(\widehat{t}_{i+1}) && \text{since } U_s(C[\widehat{t}_{i+1}]) \xrightarrow{\mathcal{S}^*}_{Th\|\mathcal{E}\setminus DP(\mathcal{R},\mu),\mu} U_{s'}(\widehat{t}_{i+1}) \\
&&& \text{for any hiding context } C \\
&\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus DP(\mathcal{R},\mu),\mu} \widehat{t}'_{i+1} && \text{using } U_{s'}(h) \rightarrow h^\#[\top] \text{ since } \widehat{t}'_{i+1} \gtrsim_{\mathcal{E}}^\Lambda \widehat{t}_{i+1} \\
&&& \text{and } \widehat{t}'_{i+1} \text{ is an instance of a hidden term } h \\
&\xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^\Lambda && \widehat{t}'_{i+1}
\end{aligned}$$

Notice that terms of the form $U_s(\dots)$ are terminating w.r.t. $\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu}$ since $\mu(U_s) = \emptyset$. Also, $t_i^\#$ and $\widehat{t}_{i+1}^\#$ are terminating w.r.t. $\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu}$ since $t_i, \widehat{t}_{i+1} \in \mathcal{M}_{(\mathcal{R},\mathcal{S},\mathcal{E},\mu)}^\infty$.

Using $t_i^\# \xrightarrow{\mathcal{S}^+}_{Th\|\mathcal{E}\setminus DP(\mathcal{R},\mu),\mu} \widehat{t}'_{i+1}^\# \xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^\Lambda \widehat{t}_{i+1}^\#$, the sequence

$$t = \widehat{t}_1 \xrightarrow{\mathcal{S}^*}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} t_1 \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} s_1 \triangleright_\mu \widehat{t}_2 \xrightarrow{\mathcal{S}^*}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} t_2 \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} s_2 \triangleright_\mu \widehat{t}_3 \dots$$

is transformed into

$$t^\# = \widehat{t}_1^\# \xrightarrow{\mathcal{S}^*}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} t_1^\# \xrightarrow{\mathcal{S}^+}_{Th\|\mathcal{E}\setminus DP(\mathcal{R},\mu),\mu} \widehat{t}'_2^\# \xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^\Lambda \widehat{t}_2^\# \xrightarrow{\mathcal{S}^*}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} t_2^\# \dots$$

Using Lemma 11.10 and the strong \mathcal{E} -coherence of $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$, this gives rise to an infinite minimal $(DP(\mathcal{R},\mu), \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -chain. \square

A.11 Proofs from Chapter 12

Proof of Theorem 12.3. It needs to be shown that there exists a substitution σ that is Th -based for $\mathcal{V}(s_1) \cup \mathcal{V}(s_2)$ such that $\text{CAP}(t_1)\sigma \xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^\Lambda s_2\sigma$, the terms $s_1\sigma$ and $s_2\sigma$ are irreducible by $\xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}}$, and $\varphi_1\sigma$ and $\varphi_2\sigma$ are Th -valid whenever $s_1 \rightarrow t_1[\varphi_1]$, $s_2 \rightarrow t_2[\varphi_2]$ is a $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -chain, i.e.,

Appendix A. Proofs

$t_1\sigma \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu}^* u \xrightarrow{>\Lambda}_!_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} s_2\sigma$ for a \mathcal{Th} -based substitution σ such that $\varphi_1\sigma$ and $\varphi_2\sigma$ are \mathcal{Th} -valid and $s_1\sigma$ and $s_2\sigma$ are normal forms w.r.t. $\xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}}$

(†) implies $\text{CAP}_{\mu}(t_1)\eta \xrightarrow{>\Lambda}_!_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} s_2\eta$ for some substitution η that is \mathcal{Th} -based for $\mathcal{V}(s_1) \cup \mathcal{V}(s_2)$ such that $s_1\eta$ and $s_2\eta$ are normal forms w.r.t. $\xrightarrow{>\Lambda}_{\mathcal{E}\setminus\mathcal{S}}$ and $\varphi_1\eta$ and $\varphi_2\eta$ are \mathcal{Th} -valid.

In order to show (†), it is first shown that for all terms t and all substitutions η that are \mathcal{Th} -based for $\mathcal{V}(t)$,

$\text{CAP}_{\mu}(t)\eta \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} u$ implies that there exists a substitution τ that is \mathcal{Th} -

(‡) based for $\mathcal{V}(t)$ such that $u = \text{CAP}_{\mu}(t)\tau$, where η and τ differ at most for the fresh variables introduced by $\text{CAP}_{\mu}(t)$.

The property (‡) is shown by induction on t . If $\text{CAP}_{\mu}(t) \in \mathcal{V}$, then it is a fresh variable y of sort `univ` since η is \mathcal{Th} -based for $\mathcal{V}(t)$. Letting $\tau = \{y \mapsto u\}$ establishes (‡). Otherwise, $t = f(t_1, \dots, t_n)$, $\text{CAP}_{\mu}(t) = f(t'_1, \dots, t'_n)$ and there is no rule $l \rightarrow r[\![\varphi]\!] \in \mathcal{R}$ such that $f(t'_1, \dots, t'_n)\vartheta \xrightarrow{>\Lambda}_!_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} l\vartheta$ for a substitution ϑ that is \mathcal{Th} -based for $\mathcal{V}(f(t_1, \dots, t_n)) \cup \mathcal{V}(l)$ where $\varphi\vartheta$ is \mathcal{Th} -valid. First, it is shown that the $\xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu}$ -step cannot take place at the root position. If the reduction takes place at the root position, then there exist a rule $l \rightarrow r[\![\varphi]\!] \in \mathcal{R}$ and a \mathcal{Th} -based substitution ρ such that $f(t'_1, \dots, t'_n)\eta \xrightarrow{>\Lambda}_!_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} l\rho$ and $\varphi\rho$ is \mathcal{Th} -valid. Since it can be assumed that l is variable-disjoint from $f(t'_1, \dots, t'_n)$, it is possible to define the substitution $\vartheta = \sigma \cup \rho$ where ϑ is \mathcal{Th} -based for $\mathcal{V}(f(t'_1, \dots, t'_n)) \cup \mathcal{V}(l)$. Since $\varphi\vartheta$ is \mathcal{Th} -valid this is a contradiction to the assumption. Hence, the $\xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu}$ -step takes place below the root position, i.e., there exists an $1 \leq i \leq n$ such that $t'_i\eta \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} u_i$ and $u = f(t'_1\eta, \dots, u_i, \dots, t'_n\eta)$. Furthermore, $i \in \mu(f)$, which implies $t'_i = \text{CAP}_{\mu}(t_i)$. By the inductive assumption, this yields a substitution δ that is \mathcal{Th} -based for $\mathcal{V}(t_i)$ such that $u_i = \text{CAP}_{\mu}(t_i)\delta = t'_i\delta$. Since the fresh variables introduced by $\text{CAP}_{\mu}(t_i)$ are disjoint from the fresh variables introduced by $\text{CAP}_{\mu}(t_j)$ for $1 \leq j \neq i \leq n$ and since

Appendix A. Proofs

η and δ differ at most for the fresh variables introduced by $\text{CAP}_\mu(t_i)$, it is possible to define the substitution τ with $\tau(x) = \delta(x)$ if x is a fresh variable introduced by $\text{CAP}_\mu(t_i)$ and $\tau(x) = \eta(x)$ otherwise. Then τ is \mathcal{Th} -based for $\mathcal{V}(t)$ and

$$\begin{aligned} u &= f(t'_1\eta, \dots, t'_i\delta, \dots, t'_n\eta) \\ &= f(t'_1\tau, \dots, t'_i\tau, \dots, t'_n\tau) \\ &= \text{CAP}_\mu(t)\tau \end{aligned}$$

By (‡) and an induction on the number of $\xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu}$ -steps, $\text{CAP}_\mu(t)\eta \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu}^* u$ for a substitution η that is \mathcal{Th} -based for $\mathcal{V}(t)$ implies $u = \text{CAP}_\mu(t)\delta$ for some substitution δ that is \mathcal{Th} -based for $\mathcal{V}(t)$ such that η and δ differ at most for fresh variables introduced by $\text{CAP}_\mu(t)$. Since $t_1 = \text{CAP}_\mu(t_1)\sigma'$ for some substitution σ' that only instantiates fresh variables introduced by $\text{CAP}_\mu(t_1)$, in particular $t_1\sigma = \text{CAP}_\mu(t_1)\sigma'\sigma \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu}^* u$ implies $u = \text{CAP}_\mu(t_1)\delta$ for some substitution δ that is \mathcal{Th} -based for $\mathcal{V}(t_1)$ such that $\sigma'\sigma$ and δ differ at most for the fresh variables introduced by $\text{CAP}_\mu(t_1)$. Thus $\text{CAP}_\mu(t_1)\delta = u \xrightarrow{\geq\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \circ \overset{\geq\Lambda}{\approx}_{\mathcal{E}} s_2\sigma$ since $u \xrightarrow{\geq\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \circ \overset{\geq\Lambda}{\approx}_{\mathcal{E}} s_2\sigma$. Now define the substitution ξ by $\xi(x) = \delta(x)$ if x is a fresh variable introduced by $\text{CAP}_\mu(t_1)$ and $\xi(x) = \sigma(x)$ otherwise. Notice that ξ is \mathcal{Th} -based for $\mathcal{V}(s_1) \cup \mathcal{V}(s_2)$. Then $\text{CAP}_\mu(t_1)\xi = u \xrightarrow{\geq\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \circ \overset{\geq\Lambda}{\approx}_{\mathcal{E}} s_2\sigma = s_2\xi$. Since $s_1\xi = s_1\sigma$ and $s_2\xi = s_2\sigma$, the terms $s_1\xi$ and $s_2\xi$ are normal forms w.r.t. $\xrightarrow{\geq\Lambda}_{\mathcal{E}\setminus\mathcal{S}}$ by Definition 11.20. Also, $\varphi_1\xi = \varphi_1\sigma$ and $\varphi_2\xi = \varphi_2\sigma$ are \mathcal{Th} -valid. \square

Proof of Theorem 12.4. After a finite number of dependency pairs in the beginning, any infinite minimal $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -chain only contains pairs from some SCC. Hence, every infinite minimal $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -chain gives rise to an infinite minimal $(\mathcal{P}_i, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -chain for some $1 \leq i \leq n$. \square

Proof of Lemma 12.8. Similar to the proof of Lemma 6.23. \square

Proof of Theorem 12.9. In the second case soundness is obvious. Otherwise, it needs to be shown that every infinite minimal $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -chain contains only finitely many dependency pairs from \mathcal{P}' . Thus, let $s_1 \rightarrow t_1\llbracket\varphi_1\rrbracket, s_2 \rightarrow t_2\llbracket\varphi_2\rrbracket, \dots$ be

Appendix A. Proofs

an infinite minimal $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -chain using the $\mathcal{T}h$ -based substitution σ and apply the simple projection π to it.

First, consider the instantiation $s_i\sigma \rightarrow t_i\sigma[\![\varphi_i\sigma]\!] of the i^{th} dependency pair. Clearly, $\pi(s_i\sigma) = \pi(s_i)\sigma$ and $\pi(t_i\sigma) = \pi(t_i)\sigma$. Since $\pi(s_i) \triangleright_{\mathcal{E},\mu} \pi(t_i)$ by assumption, $\pi(s_i\sigma) \triangleright_{\mathcal{E},\mu} \pi(t_i\sigma)$ by Lemma 12.8.3. From $t_i\sigma \xrightarrow{\mathcal{S}}^*_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu} \xrightarrow{>\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} s_{i+i}\sigma$, the (possibly shorter) sequence $\pi(t_i\sigma) \xrightarrow{\mathcal{S}}^*_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \rightarrow^*_{\mathcal{E}\setminus\mathcal{S}} \circ \sim_{\mathcal{E}} \pi(s_{i+i}\sigma)$ is obtained since all reductions take place below the root. Thus, $t_i\sigma \xrightarrow{\mathcal{S}}^*_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \xrightarrow{>\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \circ \gtrsim_{\mathcal{E}}^{\Lambda} s_{i+1}\sigma \rightarrow t_{i+1}\sigma$ implies $\pi(t_i\sigma) \xrightarrow{\mathcal{S}}^*_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \rightarrow^*_{\mathcal{E}\setminus\mathcal{S}} \circ \sim_{\mathcal{E}} \pi(s_{i+1}\sigma) \triangleright_{\mathcal{E},\mu} \pi(t_{i+1}\sigma)$.$

Therefore, the infinite minimal $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -chain is transformed into an infinite $\xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu} \cup \rightarrow_{\mathcal{E}\setminus\mathcal{S}} \cup \triangleright_{\mathcal{E},\mu} \cup \sim_{\mathcal{E}}$ sequence starting with $\pi(t_1\sigma)$. Now perform a case analysis.

Case 1: The infinite sequence contains only finitely many $\xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu}$ -steps and only finitely many $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ -steps. Then, there exists an infinite $\triangleright_{\mathcal{E},\mu} \cup \sim_{\mathcal{E}}$ sequence. This sequence cannot contain infinitely many $\triangleright_{\mathcal{E},\mu}$ -steps stemming from dependency pairs in \mathcal{P}' since otherwise Lemma 12.8.4 yields an infinite $\triangleright_{\mathcal{E},\mu}$ sequence, contradicting the well-foundedness of $\triangleright_{\mathcal{E},\mu}$ (Lemma 12.8.2).

Case 2: The infinite sequence contains only finitely many $\xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu}$ -steps but infinitely many $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ -steps. Recall that $\sim_{\mathcal{E}} \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}} \subseteq \rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \sim_{\mathcal{E}}$ since $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ is strongly \mathcal{E} -coherent. Using this and the easily seen inclusion $\triangleright_{\mu} \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}} \subseteq \rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \triangleright_{\mu}$, it is furthermore straightforward to show that

$$\begin{aligned}
\triangleright_{\mathcal{E},\mu} \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}} &= \sim_{\mathcal{E}} \circ \triangleright_{\mu} \circ \sim_{\mathcal{E}} \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}} \\
&\subseteq \sim_{\mathcal{E}} \circ \triangleright_{\mu} \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \sim_{\mathcal{E}} \\
&\subseteq \sim_{\mathcal{E}} \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \triangleright_{\mu} \circ \sim_{\mathcal{E}} \\
&\subseteq \rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \sim_{\mathcal{E}} \circ \triangleright_{\mu} \circ \sim_{\mathcal{E}} \\
&= \rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \triangleright_{\mathcal{E},\mu}
\end{aligned}$$

Appendix A. Proofs

By making repeated use of these inclusions, an infinite $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ -sequence is obtained, contradiction the assumption that $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ is terminating.

Case 3: The infinite sequence contains infinitely many $\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu}$ -steps. Recall Lemma 11.10 and the inclusions $\sim_{\mathcal{E}} \circ \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \subseteq \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \sim_{\mathcal{E}}$ and $\rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \subseteq \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}}$. Using the first of these inclusions and the easily seen inclusion $\triangleright_{\mu} \circ \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \subseteq \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \triangleright_{\mu}$, it is furthermore straightforward to show that

$$\begin{aligned} \triangleright_{\mathcal{E},\mu} \circ \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} &= \sim_{\mathcal{E}} \circ \triangleright_{\mu} \circ \sim_{\mathcal{E}} \circ \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \\ &\subseteq \sim_{\mathcal{E}} \circ \triangleright_{\mu} \circ \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \sim_{\mathcal{E}} \\ &\subseteq \sim_{\mathcal{E}} \circ \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \triangleright_{\mu} \circ \sim_{\mathcal{E}} \\ &\subseteq \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \sim_{\mathcal{E}} \circ \triangleright_{\mu} \circ \sim_{\mathcal{E}} \\ &= \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \triangleright_{\mathcal{E},\mu} \end{aligned}$$

By making repeated use of these inclusions, an infinite $\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu}$ -sequence starting with $\pi(t_1\sigma)$ is obtained. But this contradicts the minimality of the infinite minimal $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -chain since then $t_1\sigma$ starts an infinite $\xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\setminus\mathcal{R},\mu}$ reduction. \square

Proof of Theorem 12.11. Similar to the proofs from Chapter 7. \square

Before proving Lemma 12.16, several auxiliary results need to be obtained. The first result is similar to Lemma 2.21 but considers \mathcal{S} .

Lemma A.6. *If $s \rightarrow_{\mathcal{E}\setminus\mathcal{S}}^* \circ \sim_{\mathcal{E}} C[f(t^*)]$ for a context C and an $f \notin \mathcal{F}(\mathcal{E}) \cup \mathcal{F}(\mathcal{S})$, then $s = D[f(s^*)]$ for a context D such that $f(s^*) \xrightarrow{\geq\Lambda}_{\mathcal{E}\setminus\mathcal{S}}^* \circ \gtrsim_{\mathcal{E}}^{\Lambda} f(t^*)$.*

Proof. Due to Lemma 2.21, it suffices to consider $s \rightarrow_{\mathcal{S}} C[f(t^*)]$. If $s \rightarrow_{\mathcal{S}} t = C[f(t^*)]$, then there exists a rule $l \rightarrow r \in \mathcal{S}$ such that $s|_p = l\sigma$ and $t = s[r\sigma]_p$ for some position p and some substitution σ . Let q be the position with $t|_q = f(t^*)$, i.e., $C|_q = \square$. Now, a case analysis on the relationship between the positions p and q is performed.

Appendix A. Proofs

Case 1: $p \parallel q$. Then, $s = t[l\sigma]_p = (C[f(t^*)])[l\sigma]_p = (C[l\sigma]_p)[f(t^*)]$.

Case 2: $p = q.q'$ for some position $q' \neq \Lambda$. Then, $s = t[l\sigma]_p = (C[f(t^*)])[l\sigma]_{q.q'} = C[f(t^*)][l\sigma]_{q'}$. Since $q' \neq \Lambda$, the position q' can be written as $q' = i.q''$ for some $1 \leq i \leq \text{arity}(f)$ and some position q'' . Then $s_j = t_j$ if $i \neq j$ and $s_i = t_i[l\sigma]_{q''} \rightarrow_{\mathcal{S}} t_i[r\sigma]_{q''} = t_i$, i.e., $s^* \rightarrow_{\mathcal{S}} t^*$.

Case 3: $q = p.p'$ for some position p' (possibly $p' = \Lambda$). Since $f \notin \mathcal{F}(\mathcal{S})$, the position p' can be written as $p' = p'_1.p'_2$ such that $r|_{p'_1}$ is a variable x and $x\sigma|_{p'_2} = f(t^*)$. There exists a position p''_1 in l such that $l|_{p''_1} = x$. This implies $l\sigma|_{p''_1.p'_2} = x\sigma|_{p'_2} = f(t^*)$. Thus, $s = t[l\sigma]_p = D[f(t^*)]$ where $D = C[l\sigma[\square]_{p''_1.p'_2}]$ and the claim follows. \square

Next, it can be shown that $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ and $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}, \mu}$ are finitely branching if \mathcal{E} is size-preserving.

Lemma A.7. *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ be a CS-CERS such that \mathcal{E} is size-preserving. Then $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ and $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}, \mu}$ are finitely branching.*

Proof. Similar to the proof of Lemma A.3. \square

Now it can be shown that $\mathcal{I}^1(t)$ is indeed well-defined.

Proof of Lemma 12.16. According to Definition 12.15, in order to get an infinite term as a result of $\mathcal{I}^1(t)$, it is necessary to perform an infinite number of applications of $\mathcal{R}ed_{\mathcal{R}}^1$ since $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ and $\triangleright_{\mathcal{E}}$ are well-founded, $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ and $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}, \mu}$ are finitely branching by Lemma A.7, and the \mathcal{E} -equivalence classes are finite. This means that t is terminating and there exists an infinite sequence $t \bowtie_1 s_1 \bowtie_2 s_2 \dots$ where $\bowtie_i \in \{\triangleright, \rightarrow_{\mathcal{E} \setminus \mathcal{S}}, \sim_{\mathcal{E}}, \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}, \mu}\}$ and $\bowtie_i = \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}, \mu}$ for infinitely many i . By considering two consecutive occurrences of $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}, \mu}$ in this sequence and by using the inclusions $\sim_{\mathcal{E}} \circ \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \subseteq \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \circ \sim_{\mathcal{E}}$ obtained from the strong \mathcal{E} -coherence of $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ and the easily seen inclusions $\triangleright \circ \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \subseteq \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \circ \triangleright$ and $\triangleright \circ \sim_{\mathcal{E}} \subseteq \sim_{\mathcal{E}} \circ \triangleright$, a sequence $t \xrightarrow{*}_{\mathcal{E} \setminus \mathcal{S}} \circ \sim_{\mathcal{E}} u_1 \triangleright t_1 \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}, \mu} \circ \xrightarrow{*}_{\mathcal{E} \setminus \mathcal{S}} \circ \sim_{\mathcal{E}}$

Appendix A. Proofs

$u_2 \trianglerighteq t_2 \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\backslash\mathcal{R},\mu} \circ \rightarrow_{\mathcal{E}\backslash\mathcal{S}}^* \circ \sim_{\mathcal{E}} u_3 \dots$ where $\text{root}(t_i) \notin \Delta^1$ and t_i is terminating for all $i \geq 1$ is obtained. It can be assumed without loss of generality that it is not the case that $t_i \triangleright \circ \rightarrow_{\mathcal{E}\backslash\mathcal{S}}^* \circ \sim_{\mathcal{E}} t_{i+1}$ (since otherwise the modified sequence obtained from $\dots \trianglerighteq t_{i-1} \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\backslash\mathcal{R},\mu} \circ \rightarrow_{\mathcal{E}\backslash\mathcal{S}}^* \circ \sim_{\mathcal{E}} u_i \triangleright \circ \rightarrow_{\mathcal{E}\backslash\mathcal{S}}^* \circ \sim_{\mathcal{E}} t_{i+1} \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\backslash\mathcal{R},\mu} \dots$ by using the inclusions $\sim_{\mathcal{E}} \circ \rightarrow_{\mathcal{E}\backslash\mathcal{S}} \subseteq \rightarrow_{\mathcal{E}\backslash\mathcal{S}} \circ \sim_{\mathcal{E}}$, $\triangleright \circ \rightarrow_{\mathcal{E}\backslash\mathcal{S}} \subseteq \rightarrow_{\mathcal{E}\backslash\mathcal{S}} \circ \triangleright$, and $\triangleright \circ \sim_{\mathcal{E}} \subseteq \sim_{\mathcal{E}} \circ \triangleright$ mentioned above can be considered instead).

For all $i \geq 1$, there exist a rule $l_i \rightarrow r_i[\varphi_i] \in \mathcal{R}$, a Th -based substitution σ_i , and an active position p_i in t_i such that $t_i = C_i[l'_i]_{p_i}$, $l'_i \xrightarrow{\geq \Lambda}_{\mathcal{E}\backslash\mathcal{S}} l_i \sigma_i$, $C_i[r_i \sigma_i]_{p_i} \rightarrow_{\mathcal{E}\backslash\mathcal{S}}^* \circ \sim_{\mathcal{E}} u_{i+1}$, and $u_{i+1} \trianglerighteq t_{i+1}$. Now $u_{i+1} \trianglerighteq t_{i+1}$ and $\text{root}(t_{i+1}) \notin \mathcal{F}(\mathcal{E}) \cup \mathcal{F}(\mathcal{S})$ (since $\text{root}(t_{i+1}) \notin \Delta^1$ and $\mathcal{F}(\mathcal{E}) \cup \mathcal{F}(\mathcal{S}) \subseteq \Delta^1$) imply by Lemma A.6 that $C_i[r_i \sigma_i] \trianglerighteq t'_{i+1}$ for some t'_{i+1} with $\text{root}(t'_{i+1}) = \text{root}(t_{i+1})$ and $t'_{i+1} \xrightarrow{\geq \Lambda}_{\mathcal{E}\backslash\mathcal{S}} \circ \sim_{\mathcal{E}} t_{i+1}$. There are three possibilities:

1. t'_{i+1} is a subterm of $C_i[r_i \sigma_i]_{p_i}$ at a position above p_i , i.e., $t'_{i+1} \trianglerighteq r_i \sigma_i$. Then $C_i[r_i \sigma_i]_{p_i} \trianglerighteq_{\mu} t'_{i+1}$ since p_i is an active position.
2. t'_{i+1} is a subterm of $C_i[r_i \sigma_i]_{p_i}$ at a position that is independent of p_i . Then $t_i \triangleright t'_{i+1} \rightarrow_{\mathcal{E}\backslash\mathcal{S}}^* \circ \sim_{\mathcal{E}} t_{i+1}$, contradicting the assumption.
3. t'_{i+1} is a subterm of $C_i[r_i \sigma_i]_{p_i}$ at a position strictly below p_i , i.e., $r_i \sigma_i \triangleright t'_{i+1}$. Notice that there is no variable $x \in \mathcal{V}(r_i)$ with $\sigma_i(x) \triangleright t'_{i+1}$ since this would imply $l_i \sigma_i \triangleright t'_{i+1}$. Then $l'_i \triangleright t''_{i+1}$ for some t''_{i+1} such that $t''_{i+1} \rightarrow_{\mathcal{E}\backslash\mathcal{S}}^* \circ \sim_{\mathcal{E}} t'_{i+1}$ by Lemma A.6, which implies $t_i \triangleright t''_{i+1} \rightarrow_{\mathcal{E}\backslash\mathcal{S}}^* \circ \sim_{\mathcal{E}} t_{i+1}$, contradicting the assumption. Therefore, there exists a term r'_i with $r'_i \notin \mathcal{V}$ and $r_i \trianglerighteq r'_i$ such that $r'_i \sigma_i = t'_{i+1}$. Now, $\text{root}(r'_i) = \text{root}(t'_{i+1}) \notin \Delta^1$ and $\mathcal{F}^{-\mu}(r_i) \subseteq \Delta^1$ imply $r_i \trianglerighteq_{\mu} r'_i$ and thus $C_i[r_i \sigma_i]_{p_i} \trianglerighteq_{\mu} t'_{i+1}$.

The resulting sequence is thus $t \rightarrow_{\mathcal{E}\backslash\mathcal{S}}^* \circ \sim_{\mathcal{E}} u_1 \trianglerighteq_{\mu} t_1 \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\backslash\mathcal{R},\mu} C_1[r_1 \sigma_1]_{p_1} \trianglerighteq_{\mu} t'_2 \rightarrow_{\mathcal{E}\backslash\mathcal{S}}^* \circ \sim_{\mathcal{E}} t_2 \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\backslash\mathcal{R},\mu} C_2[r_2 \sigma_2]_{p_2} \trianglerighteq_{\mu} t'_3 \rightarrow_{\mathcal{E}\backslash\mathcal{S}}^* \circ \sim_{\mathcal{E}} t_3 \dots$

Using the inclusions $\sim_{\mathcal{E}} \circ \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\backslash\mathcal{R},\mu} \subseteq \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\backslash\mathcal{R},\mu} \circ \sim_{\mathcal{E}}$ and $\rightarrow_{\mathcal{E}\backslash\mathcal{S}} \circ \xrightarrow{\mathcal{S}}_{Th\|\mathcal{E}\backslash\mathcal{R},\mu}$

Appendix A. Proofs

$\subseteq \xrightarrow{\mathcal{S}} \overset{+}{\mathcal{T}h\|\mathcal{E}\backslash\mathcal{R},\mu} \circ \rightarrow_{\overline{\mathcal{E}}\backslash\mathcal{S}}$ from Lemma 11.10 and $\triangleright_{\mathcal{E},\mu} \circ \xrightarrow{\mathcal{S}} \mathcal{T}h\|\mathcal{E}\backslash\mathcal{R},\mu \subseteq \xrightarrow{\mathcal{S}} \mathcal{T}h\|\mathcal{E}\backslash\mathcal{R},\mu \circ \triangleright_{\mathcal{E},\mu}$ from the proof of Theorem 12.9, an infinite $\xrightarrow{\mathcal{S}} \mathcal{T}h\|\mathcal{E}\backslash\mathcal{R},\mu$ sequence starting with t is obtained, contradicting the assumption that t is terminating. \square

Proof of Lemma 12.17. Let $s, t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{T}h}, \mathcal{V})$ and let σ be a $\mathcal{T}h$ -based substitution.

1. $\mathcal{I}^1(s\sigma) = s\mathcal{I}^1(\sigma)$ is proved by induction on s . If $s \in \mathcal{V}$ then this is immediate by the definition of $\mathcal{I}^1(\sigma)$. Otherwise, $s = f(s_1, \dots, s_n)$ with $f \in \Delta^1$. But then

$$\begin{aligned} \mathcal{I}^1(s\sigma) &= \mathcal{I}^1(f(s_1\sigma, \dots, s_n\sigma)) \\ &= f(\mathcal{I}^1(s_1\sigma), \dots, \mathcal{I}^1(s_n\sigma)) \\ &= f(s_1\mathcal{I}^1(\sigma), \dots, s_n\mathcal{I}^1(\sigma)) \\ &= s\mathcal{I}^1(\sigma) \end{aligned}$$

by the inductive assumption.

2. $\mathcal{I}^1(s\sigma) \rightarrow_{\mathcal{R}_{\Pi}}^* s\mathcal{I}^1(\sigma)$ is proved by induction on s . If $s \in \mathcal{V}$ then this is immediate by the definition of $\mathcal{I}^1(\sigma)$. Otherwise, $s = f(s_1, \dots, s_n)$. By the assumption, if $i \notin \mu(f)$, then $s_i \in \mathcal{T}(\Delta^1, \mathcal{V})$ and hence $\mathcal{I}^1(s_i\sigma) = s_i\mathcal{I}^1(\sigma)$ by 1. If $i \in \mu(f)$, then $\mathcal{I}^1(s_i\sigma) \rightarrow_{\mathcal{R}_{\Pi}}^* s_i\mathcal{I}^1(\sigma)$ by the inductive assumption. If $f \in \Delta^1$ or s is not terminating, then $\mathcal{I}^1(s\sigma) = \mathcal{I}^1(f(s_1\sigma, \dots, s_n\sigma)) = f(\mathcal{I}^1(s_1\sigma), \dots, \mathcal{I}^1(s_n\sigma)) \rightarrow_{\mathcal{R}_{\Pi}}^* f(s_1\mathcal{I}^1(\sigma), \dots, s_n\mathcal{I}^1(\sigma)) = s\mathcal{I}^1(\sigma)$. If $f \notin \Delta^1$ and s is terminating, then $\mathcal{I}^1(s\sigma) = \text{Comp}_{\text{sort}(s\sigma)}(\mathcal{R}ed_{\mathcal{S}}^1(s\sigma) \cup \mathcal{R}ed_{\mathcal{R}}^1(s\sigma) \cup \mathcal{E}q_{\mathcal{E}}^1(s\sigma))$. The definition of $\mathcal{E}q_{\mathcal{E}}^1(s\sigma)$ implies $f(\mathcal{I}^1(s_1\sigma), \dots, \mathcal{I}^1(s_n\sigma)) \in \mathcal{E}q_{\mathcal{E}}^1(s\sigma)$ and thus $\mathcal{I}^1(s\sigma) \rightarrow_{\mathcal{R}_{\Pi}}^+ f(\mathcal{I}^1(s_1\sigma), \dots, \mathcal{I}^1(s_n\sigma)) \rightarrow_{\mathcal{R}_{\Pi}}^* f(s_1\mathcal{I}^1(\sigma), \dots, s_n\mathcal{I}^1(\sigma))$ by Lemma 8.16.

3. It suffices to show that $s \vdash_{\mathcal{E}} t$ implies $\mathcal{I}^1(s) \vdash_{\overline{\mathcal{E}}} \mathcal{I}^1(t)$ since the statement then follows by induction on the number of $\vdash_{\mathcal{E}}$ -steps in $s \sim_{\mathcal{E}} t$.

Thus, let $s \vdash_{\mathcal{E}} t$ and perform an induction on the position p where the step takes places. If $\text{root}(s) \notin \Delta^1$ and s is terminating, then $\text{root}(t) \notin \Delta^1$ as

Appendix A. Proofs

well by the definition of Δ^1 and t is terminating by Corollary 11.11.1. Since $s \sim_{\mathcal{E}} t$, Definition 3.12 implies that whenever $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}} s'$, then $t \rightarrow_{\mathcal{E} \setminus \mathcal{S}} t'$ for some $t' \sim_{\mathcal{E}} s'$. Thus, $\mathcal{R}ed_{\mathcal{S}}^1(s) \subseteq \mathcal{R}ed_{\mathcal{S}}^1(t)$. Similarly, Lemma 11.10.2 implies $\mathcal{R}ed_{\mathcal{R}}^1(s) \subseteq \mathcal{R}ed_{\mathcal{R}}^1(t)$. Finally, if $s \sim_{\mathcal{E}} g(s_1, \dots, s_n)$, then $t \sim_{\mathcal{E}} g(s_1, \dots, s_n)$, which immediately implies $\mathcal{E}q_{\mathcal{E}}^1(s) = \mathcal{E}q_{\mathcal{E}}^1(t)$.

Using these properties, $\mathcal{R}ed_{\mathcal{S}}^1(s) \cup \mathcal{R}ed_{\mathcal{R}}^1(s) \cup \mathcal{E}q_{\mathcal{E}}^1(s) \subseteq \mathcal{R}ed_{\mathcal{S}}^1(t) \cup \mathcal{R}ed_{\mathcal{R}}^1(t) \cup \mathcal{E}q_{\mathcal{E}}^1(t)$. Since the same reasoning can be applied with s and t interchanged,

$$\begin{aligned} & \text{Comp}_{\text{sort}(s)}(\mathcal{R}ed_{\mathcal{S}}^1(s) \cup \mathcal{R}ed_{\mathcal{R}}^1(s) \cup \mathcal{E}q_{\mathcal{E}}^1(s)) \\ &= \text{Comp}_{\text{sort}(s)}(\mathcal{R}ed_{\mathcal{S}}^1(t) \cup \mathcal{R}ed_{\mathcal{R}}^1(t) \cup \mathcal{E}q_{\mathcal{E}}^1(t)) \end{aligned}$$

and thus $\mathcal{I}^1(s) = \mathcal{I}^1(t)$.

Otherwise, $\text{root}(s) \in \Delta^1$ or s is not terminating. If $p = \Lambda$, then there exist an equation $u \approx v$ (or $v \approx u$) in \mathcal{E} and a substitution σ such that $s = u\sigma$ and $t = v\sigma$. By the definition of Δ^1 , $u, v \in \mathcal{T}(\Delta^1, \mathcal{V})$. Hence, $\mathcal{I}^1(s) = \mathcal{I}^1(u\sigma) = u\mathcal{I}^1(\sigma) \vdash_{\mathcal{E}} v\mathcal{I}^1(\sigma) = \mathcal{I}^1(v\sigma) = \mathcal{I}^1(t)$ by 1. If $p \neq \Lambda$ then $s = f(s_1, \dots, s_i, \dots, s_n)$, $t = f(s_1, \dots, t_i, \dots, s_n)$, and $s_i \vdash_{\mathcal{E}} t_i$. Now $\mathcal{I}(s_i) \vdash_{\mathcal{E}} \mathcal{I}(t_i)$ by the inductive assumption, which implies

$$\begin{aligned} \mathcal{I}(s) &= f(\mathcal{I}^1(s_1), \dots, \mathcal{I}^1(s_i), \dots, \mathcal{I}^1(s_n)) \\ &\vdash_{\mathcal{E}} f(\mathcal{I}^1(s_1), \dots, \mathcal{I}^1(s'_i), \dots, \mathcal{I}^1(s_n)) \\ &= \mathcal{I}^1(t) \end{aligned}$$

4. It suffices to show that $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}} t$ implies $\mathcal{I}^1(s) \rightsquigarrow_1 \mathcal{I}^1(t)$ since the statement then follows by induction on the number of $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ -steps in $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}}^* t$.

Thus, let $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}} t$ and perform an induction on the position p where the reduction takes places. If $\text{root}(s) \notin \Delta^1$ and s is terminating, then $\mathcal{I}^1(t) \in \mathcal{R}ed_{\mathcal{S}}^1(s)$, which implies $\mathcal{I}^1(s) \rightarrow_{\mathcal{R}_{\Pi}}^+ \mathcal{I}^1(t)$ by Lemma 8.16.

If $\text{root}(s) \in \Delta^1$ or s is not terminating, first consider the case $p = \Lambda$. Then, there exist a rule $l \rightarrow r \in \mathcal{S}$ and a substitution σ such that $s \sim_{\mathcal{E}} l\sigma \rightarrow_{\mathcal{S}}$

Appendix A. Proofs

$r\sigma = t$. Using 1 and 3, $\mathcal{I}^1(s) \sim_{\mathcal{E}} \mathcal{I}^1(l\sigma) = l\mathcal{I}^1(\sigma) \rightarrow_{\mathcal{S}} r\mathcal{I}^1(\sigma) = \mathcal{I}^1(r\sigma) = \mathcal{I}^1(t)$, and thus $\mathcal{I}^1(s) \rightarrow_{\mathcal{E}\setminus\mathcal{S}} \mathcal{I}^1(t)$. If $\text{root}(s) \in \Delta^1$ and $p \neq \Lambda$, then $s = f(s_1, \dots, s_i, \dots, s_n)$ and $t = f(s_1, \dots, t_i, \dots, s_n)$, where $s_i \rightarrow_{\mathcal{E}\setminus\mathcal{S}} t_i$. This implies $\mathcal{I}^1(s_i) \rightsquigarrow_1 \mathcal{I}^1(t_i)$ by the inductive assumption and therefore $\mathcal{I}^1(s) = f(\mathcal{I}^1(s_1), \dots, \mathcal{I}^1(s_i), \dots, \mathcal{I}^1(s_n)) \rightsquigarrow_1 f(\mathcal{I}^1(s_1), \dots, \mathcal{I}^1(t_i), \dots, \mathcal{I}^1(s_n)) = \mathcal{I}^1(t)$.

5. It suffices to show that $s \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} t$ implies $\mathcal{I}^1(s) \rightsquigarrow_2 \mathcal{I}^1(t)$ since the statement then follows by induction on the number of $\xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu}$ -steps in $s \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu}^* t$.

Thus, let $s \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} t$ and perform an induction on the position p where the reduction takes places. If $\text{root}(s) \notin \Delta^1$, then $\mathcal{I}^1(t) \in \mathcal{Red}_{\mathcal{R}}^1(s)$, which implies $\mathcal{I}^1(s) \rightarrow_{\mathcal{R}\Pi}^+ \mathcal{I}^1(t)$ by Lemma 8.16.

If $\text{root}(s) \in \Delta^1$, first consider the case $p = \Lambda$. Then, there exist a rule $l \rightarrow r[\varphi] \in \mathcal{R}$ and a \mathcal{Th} -based substitution σ with $s = f(s^*) \xrightarrow{>\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \circ \xrightarrow{\sim}_{\mathcal{E}} \Lambda$ $l\sigma \rightarrow_{\mathcal{R}} r\sigma = t$ such that $\varphi\sigma$ is \mathcal{Th} -valid. Since $\text{root}(l) = \text{root}(s) = f$ and $f \in \Delta^1$, the definition of Δ^1 implies that $l \rightarrow r[\varphi] \in \mathcal{R}(\Delta^1)$, $r \in \mathcal{T}(\Delta^1, \mathcal{V})$, and $l' \in \mathcal{T}(\Delta^1, \mathcal{V})$ whenever $l \triangleright_{\neg\mu} l'$. Using 1, 2, 3, and 4, $\mathcal{I}^1(s) \rightsquigarrow_1^* \circ \sim_{\mathcal{E}} \mathcal{I}^1(l\sigma) \rightarrow_{\mathcal{R}\Pi}^* l\mathcal{I}^1(\sigma) \rightarrow_{\mathcal{R}(\Delta^1),\mu} r\mathcal{I}^1(\sigma) = \mathcal{I}^1(r\sigma) = \mathcal{I}^1(t)$ where $\varphi\mathcal{I}^1(\sigma) = \varphi\sigma$ is \mathcal{Th} -valid, and thus $\mathcal{I}^1(s) \rightsquigarrow_2 \mathcal{I}^1(t)$. If $\text{root}(s) \in \Delta^1$ and $p \neq \Lambda$, then $s = f(s_1, \dots, s_i, \dots, s_n)$ and $t = f(s_1, \dots, t_i, \dots, s_n)$, where $s_i \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu} t_i$ for an $i \in \mu(f)$. Now $\mathcal{I}^1(s_i) \rightsquigarrow_2 \mathcal{I}^1(t_i)$ by the inductive assumption since s_i is terminating and therefore $\mathcal{I}^1(s) = f(\mathcal{I}^1(s_1), \dots, \mathcal{I}^1(s_i), \dots, \mathcal{I}^1(s_n)) \rightsquigarrow_2 f(\mathcal{I}^1(s_1), \dots, \mathcal{I}^1(t_i), \dots, \mathcal{I}^1(s_n)) = \mathcal{I}^1(t)$.

6. Let $s\sigma \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R},\mu}^* \rightarrow_{\mathcal{E}\setminus\mathcal{S}}^! \circ \sim_{\mathcal{E}} t\sigma$. Using 3, 4, and 5, $\mathcal{I}^1(s\sigma) \rightsquigarrow_2^* \circ \rightsquigarrow_1^* \circ \sim_{\mathcal{E}} \mathcal{I}^1(t\sigma)$. Using 1 and 2 this implies $s\mathcal{I}^1(\sigma) \rightsquigarrow_2^* \circ \rightsquigarrow_1^* \circ \sim_{\mathcal{E}} \circ \rightarrow_{\mathcal{R}\Pi}^* t\mathcal{I}^1(\sigma)$. \square

Proof of Theorem 12.18. Similar to the proof of Theorem 8.18 and making use of Lemma 12.17.6. \square

Before proving Lemma 12.24, some auxiliary results need to be obtained. Lemma

Appendix A. Proofs

12.24 will be shown by well-founded induction using the following relations. This is similar to Definition A.1.

Definition A.8 ($\overline{\mathfrak{Q}}^2, \mathfrak{Q}^2$). *Let s and t be terms. Then $s \overline{\mathfrak{Q}}^2 t$ iff $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \cup \triangleright_{\mathcal{E}} t$ and $s \mathfrak{Q}^2 t$ iff $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \cup \xrightarrow{\mathcal{S}}_{\text{Th} \parallel \mathcal{E} \setminus \mathcal{R}, \mu} \cup \triangleright_{\mathcal{E}, \mu} t$.*

In order to use $\overline{\mathfrak{Q}}^2$ and \mathfrak{Q}^2 for inductive proofs, they need to be well-founded.

Lemma A.9.

1. *If s is terminating and $s \mathfrak{Q}^2 t$, then t is terminating.*
2. *$\overline{\mathfrak{Q}}^2$ is well-founded.*
3. *\mathfrak{Q}^2 is well-founded on terminating terms.*

Proof.

1. If $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}} t$, then t is terminating by Corollary 11.11.2. If $s \xrightarrow{\mathcal{S}}_{\text{Th} \parallel \mathcal{E} \setminus \mathcal{R}, \mu} t$ then t is clearly terminating if s is terminating. If $s \triangleright_{\mathcal{E}, \mu} t$ then t is terminating due to the inclusion $\triangleright_{\mathcal{E}, \mu} \circ \xrightarrow{\mathcal{S}}_{\text{Th} \parallel \mathcal{E} \setminus \mathcal{R}, \mu} \subseteq \xrightarrow{\mathcal{S}}_{\text{Th} \parallel \mathcal{E} \setminus \mathcal{R}, \mu} \circ \triangleright_{\mathcal{E}, \mu}$ from the proof of Theorem 12.9.
2. Assume that $\overline{\mathfrak{Q}}^2$ is not well-founded. Then, there exists an infinite $\rightarrow_{\mathcal{E} \setminus \mathcal{S}} \cup \triangleright_{\mathcal{E}}$ -sequence. Since $\triangleright_{\mathcal{E}}$ is well-founded by Lemma 6.23.2, this sequence contains infinitely many $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ -steps. Using the inclusion $\triangleright_{\mathcal{E}} \circ \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \subseteq \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \circ \triangleright_{\mathcal{E}}$ from the proof of Theorem 6.24, an infinite $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ sequence is obtained, contradicting the assumption that $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ is terminating.
3. Assume that \mathfrak{Q}^2 is not well-founded on terminating terms. Then, there exists an infinite $\xrightarrow{\mathcal{S}}_{\text{Th} \parallel \mathcal{E} \setminus \mathcal{R}, \mu} \cup \rightarrow_{\mathcal{E} \setminus \mathcal{S}} \cup \triangleright_{\mathcal{E}, \mu}$ -sequence containing only terminating terms.

If this sequence contains only finitely many $\xrightarrow{\mathcal{S}}_{\text{Th} \parallel \mathcal{E} \setminus \mathcal{R}, \mu}$ -steps, then an infinite $\rightarrow_{\mathcal{E} \setminus \mathcal{S}} \cup \triangleright_{\mathcal{E}, \mu}$ -sequence is obtained. Since $\triangleright_{\mathcal{E}, \mu}$ is well-founded by Lemma

Appendix A. Proofs

12.8.2, this sequence contains infinitely many $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ -steps. Using the inclusion $\triangleright_{\mathcal{E},\mu} \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}} \subseteq \rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \triangleright_{\mathcal{E},\mu}$ from the proof of Theorem 12.9, an infinite $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ -sequence is obtained, contradicting the assumption that $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ is terminating.

Otherwise, the sequence contains infinitely many $\xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu}$ -steps. Using the inclusions $\rightarrow_{\mathcal{E}\setminus\mathcal{S}} \circ \xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu} \subseteq \xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ from Lemma 11.10.3 and $\triangleright_{\mathcal{E},\mu} \circ \xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu} \subseteq \xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu} \circ \triangleright_{\mathcal{E},\mu}$ from the proof of Theorem 12.9, an infinite $\xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R},\mu}$ -sequence starting with a terminating term is obtained, which is clearly impossible. \square

The first property will be used freely in the following.

Proof of Lemma 12.24. The first claim is proved by induction on $\overline{\mathfrak{Q}}^2$, which, by Lemma A.9.2, is well-founded.

If $t \in \mathcal{V}$ then $\overline{\mathcal{I}}^2(t) = t$ and nothing needs to be shown. If $t = f(t_1, \dots, t_n)$ with $f \in \overline{\Delta}^2$ then $\overline{\mathcal{I}}^2(t) = f(\overline{\mathcal{I}}^2(t_1), \dots, \overline{\mathcal{I}}^2(t_n))$ and the inductive assumption implies that $\overline{\mathcal{I}}^2(t_i)$ is a finite term for all $1 \leq i \leq n$. This implies that $\overline{\mathcal{I}}^2(t)$ is a finite term. Finally, let $\text{root}(t) \notin \overline{\Delta}^2$. Then the sets $\overline{\mathcal{R}ed}_{\mathcal{S}}^2(t)$ and $\overline{\mathcal{E}q}_{\mathcal{E}}^2(t)$ are finite since $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ is finitely branching by Lemma A.7 and \mathcal{E} is size-preserving, which implies that the \mathcal{E} -equivalence classes are finite. By the inductive assumption, $\overline{\mathcal{I}}^2(t')$ is a finite term for any $\overline{\mathcal{I}}^2(t') \in \overline{\mathcal{R}ed}_{\mathcal{S}}^2(t)$ and $\overline{\mathcal{I}}^2(t_i)$ is a finite term for any $g(\overline{\mathcal{I}}^2(t_1), \dots, \overline{\mathcal{I}}^2(t_m)) \in \overline{\mathcal{E}q}_{\mathcal{E}}^2(t)$ and all $1 \leq i \leq m$. But then $\overline{\mathcal{I}}^2(t)$ is a finite term as well.

Similarly, the second claim is proved by induction on \mathfrak{Q}^2 , which, by Lemma A.9.3, is well-founded on terminating terms.

If $t \in \mathcal{V}$ then $\mathcal{I}^2(t) = t$ and nothing needs to be shown. If $t = f(t_1, \dots, t_n)$ with $f \in \Delta^2$ then $\mathcal{I}^2(t) = f(\overline{t}_1, \dots, \overline{t}_n)$ and the inductive assumption implies that $\overline{t}_i = \mathcal{I}^2(t_i)$ is a finite term for all $1 \leq i \leq n$ with $i \in \mu(f)$. If $1 \leq i \leq n$ with $i \notin \mu(f)$, then $\overline{t}_i = \overline{\mathcal{I}}^2(t_i)$, which is a finite term by the first claim. Together, this implies that $\mathcal{I}^2(t)$

Appendix A. Proofs

is a finite term. Finally, let $\text{root}(t) \notin \Delta^2$. Then the sets $\mathcal{R}ed_{\mathcal{S}}^2(t)$, $\mathcal{R}ed_{\mathcal{R}}^2(t)$, $\mathcal{E}q_{\mathcal{E}}^2(t)$, and $\overline{\mathcal{E}q}_{\mathcal{E}}^2(t)$ are finite since $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ and $\xrightarrow{\mathcal{S}}_{\mathcal{Th}\parallel\mathcal{E}\setminus\mathcal{R},\mu}$ are finitely branching by Lemma A.7 and \mathcal{E} is size-preserving, which implies that the \mathcal{E} -equivalence classes are finite. By the inductive assumption, $\mathcal{I}^2(t')$ is a finite term for any $\mathcal{I}^2(t') \in \mathcal{R}ed_{\mathcal{S}}^2(t) \cup \mathcal{R}ed_{\mathcal{R}}^2(t)$ and \overline{t}_i is a finite term for any $g(\overline{t}_1, \dots, \overline{t}_m) \in \mathcal{E}q_{\mathcal{E}}^2(t)$ and all $1 \leq i \leq m$ as above. Furthermore, $\overline{\mathcal{I}^2}(s)$ is a finite term for all $\overline{\mathcal{I}^2}(s) \in \overline{\mathcal{E}q}_{\mathcal{E}}^2(t)$ by the first claim. But then $\mathcal{I}^2(t)$ is a finite term as well. \square

Next, some simple notation is introduced.

Definition A.10. For any term t and any terminating substitution σ define the substitution σ_t as $\sigma_t(x) = \mathcal{I}^2(\sigma(x))$ if $x \in \mathcal{V}^\mu(t)$ and $\sigma_t(x) = \overline{\mathcal{I}^2}(\sigma(x))$ otherwise.

Lemma A.11. If $s\sigma$ is terminating and $\mathcal{V}^\mu(s) \cap \mathcal{V}^{-\mu}(s) = \emptyset$, then $s\sigma_s = [s, \sigma]$.

Proof. Trivial. \square

Proof of Lemma 12.25. Let $s, t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{Th}}, \mathcal{V})$ and let σ be a \mathcal{Th} -based substitution such that $s, t, s\sigma$ are terminating.

1. $\overline{\mathcal{I}^2}(s\sigma) = s\overline{\mathcal{I}^2}(\sigma)$ is proved by induction on s . If $s \in \mathcal{V}$ then this is immediate by the definition of $\overline{\mathcal{I}^2}(\sigma)$. Otherwise, $s = f(s_1, \dots, s_n)$ with $f \in \overline{\Delta}^2$. But then $\overline{\mathcal{I}^2}(s\sigma) = \overline{\mathcal{I}^2}(f(s_1\sigma, \dots, s_n\sigma)) = f(\overline{\mathcal{I}^2}(s_1\sigma), \dots, \overline{\mathcal{I}^2}(s_n\sigma)) = f(s_1\overline{\mathcal{I}^2}(\sigma), \dots, s_n\overline{\mathcal{I}^2}(\sigma)) = s\overline{\mathcal{I}^2}(\sigma)$ by the inductive assumption.
2. $\mathcal{I}^2(s\sigma) = [s, \sigma]$ is proved by induction on s . If $s \in \mathcal{V}$ then this is immediate by the definition of $[s, \sigma]$. Otherwise, $s = f(s_1, \dots, s_n)$ with $f \in \Delta^2$ and thus $\mathcal{I}^2(s\sigma) = \mathcal{I}^2(f(s_1\sigma, \dots, s_n\sigma)) = f(\overline{s_1\sigma}, \dots, \overline{s_n\sigma})$. For $i \in \mu(f)$, the inductive assumption gives $\overline{s_i\sigma} = \mathcal{I}^2(s_i\sigma) = [s_i, \sigma]$. For $i \notin \mu(f)$, 1 implies $\overline{s_i\sigma} = \overline{\mathcal{I}^2}(s_i\sigma) = s_i\overline{\mathcal{I}^2}(\sigma)$. Together this implies $\mathcal{I}^2(s\sigma) = f(\overline{s_1\sigma}, \dots, \overline{s_n\sigma}) = [s, \sigma]$.
3. $\overline{\mathcal{I}^2}(s\sigma) \xrightarrow{*}_{\mathcal{R}_{\Pi}} s\overline{\mathcal{I}^2}(\sigma)$ is proved by induction on s . If $s \in \mathcal{V}$ then this is immediate by the definition of $\overline{\mathcal{I}^2}(\sigma)$. Otherwise, $s = f(s_1, \dots, s_n)$. If $f \in \overline{\Delta}^2$

Appendix A. Proofs

then $\overline{\mathcal{I}}^2(s\sigma) = f(\overline{\mathcal{I}}^2(s_1\sigma), \dots, \overline{\mathcal{I}}^2(s_n\sigma)) \rightarrow_{\mathcal{R}_\Pi}^* f(s_1\overline{\mathcal{I}}^2(\sigma), \dots, s_n\overline{\mathcal{I}}^2(\sigma)) = s\overline{\mathcal{I}}^2(\sigma)$ by the inductive assumption. Otherwise, $\overline{\mathcal{I}}^2(s\sigma) = \text{Comp}_{\text{sort}(s\sigma)}(\overline{\text{Red}}_S^2(s\sigma) \cup \overline{\mathcal{E}q}_\mathcal{E}^2(s\sigma))$. Notice that $f(\overline{\mathcal{I}}^2(s_1\sigma), \dots, \overline{\mathcal{I}}^2(s_n\sigma)) \in \overline{\mathcal{E}q}_\mathcal{E}^2(s\sigma)$, which gives the desired $\overline{\mathcal{I}}^2(s\sigma) \rightarrow_{\mathcal{R}_\Pi}^+ f(\overline{\mathcal{I}}^2(s_1\sigma), \dots, \overline{\mathcal{I}}^2(s_n\sigma)) \rightarrow_{\mathcal{R}_\Pi}^* f(s_1\overline{\mathcal{I}}^2(\sigma), \dots, s_n\overline{\mathcal{I}}^2(\sigma))$ using Lemma 8.16 and the inductive assumption.

4. $\mathcal{I}^2(s\sigma) \rightarrow_{\mathcal{R}_\Pi}^* [s, \sigma]$ is proved by induction on s . If $s \in \mathcal{V}$ then this is immediate by the definition of $[s, \sigma]$. Otherwise, $s = f(s_1, \dots, s_n)$. If $f \in \Delta^2$, then $\mathcal{I}^2(s\sigma) = \mathcal{I}^2(f(s_1\sigma, \dots, s_n\sigma)) = f(\overline{s_1\sigma}, \dots, \overline{s_n\sigma})$. If $f \notin \Delta^2$, then $f(\overline{s_1\sigma}, \dots, \overline{s_n\sigma}) \in \mathcal{E}q_\mathcal{E}^2(s\sigma)$ and thus $\mathcal{I}^2(s\sigma) \rightarrow_{\mathcal{R}_\Pi}^+ f(\overline{s_1\sigma}, \dots, \overline{s_n\sigma})$ by Lemma 8.16. For $i \in \mu(f)$, the inductive assumption implies $\overline{s_i\sigma} = \mathcal{I}^2(s_i\sigma) \rightarrow_{\mathcal{R}_\Pi}^* [s_i, \sigma]$. For $i \notin \mu(f)$, \mathcal{I} implies $\overline{s_i\sigma} = \overline{\mathcal{I}}^2(s_i\sigma) \rightarrow_{\mathcal{R}_\Pi}^* s_i\overline{\mathcal{I}}^2(\sigma)$. Together this implies $\mathcal{I}^2(s\sigma) \rightarrow_{\mathcal{R}_\Pi}^* f(\overline{s_1\sigma}, \dots, \overline{s_n\sigma}) \rightarrow_{\mathcal{R}_\Pi}^* [s, \sigma]$.
5. The statement is proved by induction on s . If $s \in \mathcal{V}$ then $\mathcal{I}^2(s) = s = \overline{\mathcal{I}}^2(s)$. Otherwise, $s = f(s_1, \dots, s_n)$. If $f \in \Delta^2 \subseteq \overline{\Delta}^2$, then $\mathcal{I}^2(s) = \mathcal{I}^2(f(s_1, \dots, s_n)) = f(\overline{s_1}, \dots, \overline{s_n})$. If $i \in \mu(f)$, the inductive assumption implies $\overline{s_i} = \mathcal{I}^2(s_i) \rightarrow_{\mathcal{R}_\Pi}^* \overline{\mathcal{I}}^2(s_i)$. If $i \notin \mu(f)$, then $\overline{s_i} = \overline{\mathcal{I}}^2(s_i)$ is immediate. Therefore, $\mathcal{I}^2(s) \rightarrow_{\mathcal{R}_\Pi}^* f(\overline{\mathcal{I}}^2(s_1), \dots, \overline{\mathcal{I}}^2(s_n)) = \overline{\mathcal{I}}^2(s)$. If $f \notin \Delta^2$, then $\overline{\mathcal{I}}^2(s) \in \overline{\mathcal{E}q}_\mathcal{E}^2(s)$ and thus $\mathcal{I}^2(s) \rightarrow_{\mathcal{R}_\Pi}^+ \overline{\mathcal{I}}^2(s)$ by Lemma 8.16.
6. It suffices to show that $s \vdash_{\mathcal{E}} t$ implies $\overline{\mathcal{I}}^2(s) \vdash_{\mathcal{E}(\overline{\Delta}^2)}^{\overline{=}} \overline{\mathcal{I}}^2(t)$ since the statement then follows by induction on the number of $\vdash_{\mathcal{E}}$ -steps in $s \sim_{\mathcal{E}} t$.

Thus, let $s \vdash_{\mathcal{E}} t$ and perform an induction on the position p where the step takes places. If $\text{root}(s) \notin \overline{\Delta}^2$, then $\text{root}(t) \notin \overline{\Delta}^2$ as well by the definition of $\overline{\Delta}^2$. Since $s \sim_{\mathcal{E}} t$, the strong \mathcal{E} -coherence of $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ implies that whenever $s \rightarrow_{\mathcal{E}\setminus\mathcal{S}} s'$, then $t \rightarrow_{\mathcal{E}\setminus\mathcal{S}} t'$ for some $t' \sim_{\mathcal{E}} s'$. Thus, $\overline{\text{Red}}_S^2(s) \subseteq \overline{\text{Red}}_S^2(t)$. Similarly, if $s \sim_{\mathcal{E}} g(s_1, \dots, s_n)$, then $t \sim_{\mathcal{E}} g(s_1, \dots, s_n)$, which immediately implies $\overline{\mathcal{E}q}_\mathcal{E}^2(s) = \overline{\mathcal{E}q}_\mathcal{E}^2(t)$. Using these properties, $\overline{\text{Red}}_S^2(s) \cup \overline{\mathcal{E}q}_\mathcal{E}^2(s) \subseteq \overline{\text{Red}}_S^2(t) \cup \overline{\mathcal{E}q}_\mathcal{E}^2(t)$. Since the same reasoning can be applied with s and t interchanged, $\text{Comp}_{\text{sort}(s)}(\overline{\text{Red}}_S^2(s) \cup$

Appendix A. Proofs

$$\overline{\mathcal{E}q_{\mathcal{E}}^2}(s) = \text{Comp}_{\text{sort}(t)}(\overline{\mathcal{R}ed}_{\mathcal{S}}^2(t) \cup \overline{\mathcal{E}q}_{\mathcal{E}}^2(t)) \text{ and thus } \overline{\mathcal{I}}^2(s) = \overline{\mathcal{I}}^2(t).$$

Otherwise, $\text{root}(s) \in \overline{\Delta}^2$. If $p = \Lambda$, then there exist an equation $u \approx v$ (or $v \approx u$) in \mathcal{E} and a substitution σ such that $s = u\sigma$ and $t = v\sigma$. By the definition of $\overline{\Delta}^2$, $u, v \in \mathcal{T}(\overline{\Delta}^2, \mathcal{V})$. Thus, $\overline{\mathcal{I}}^2(s) = \overline{\mathcal{I}}^2(u\sigma) = u\overline{\mathcal{I}}^2(\sigma) \vdash_{\mathcal{E}(\overline{\Delta}^2)} v\overline{\mathcal{I}}^2(\sigma) = \overline{\mathcal{I}}^2(v\sigma) = \overline{\mathcal{I}}^2(t)$ by 1. If $p \neq \Lambda$, then $s = f(s_1, \dots, s_i, \dots, s_n)$ and $t = f(s_1, \dots, t_i, \dots, s_n)$ such that $s_i \vdash_{\mathcal{E}} t_i$. Now $\overline{\mathcal{I}}^2(s_i) \vdash_{\mathcal{E}(\overline{\Delta}^2)} \overline{\mathcal{I}}^2(t_i)$ by the inductive assumption and therefore $\overline{\mathcal{I}}^2(s) \vdash_{\mathcal{E}(\overline{\Delta}^2)} \overline{\mathcal{I}}^2(t)$ because the definition of the mapping $\overline{\mathcal{I}}^2$ implies that $\overline{\mathcal{I}}^2(s) = f(\overline{\mathcal{I}}^2(s_1), \dots, \overline{\mathcal{I}}^2(s_i), \dots, \overline{\mathcal{I}}^2(s_n))$ and $\overline{\mathcal{I}}^2(t) = f(\overline{\mathcal{I}}^2(s_1), \dots, \overline{\mathcal{I}}^2(t_i), \dots, \overline{\mathcal{I}}^2(s_n))$.

7. It suffices to show that $s \vdash_{\mathcal{E}} t$ implies $\mathcal{I}^2(s) \vdash_{\mathcal{E}(\Delta^2)} \mathcal{I}^2(t)$ since the statement then follows by induction on the number of $\vdash_{\mathcal{E}}$ -steps in $s \sim_{\mathcal{E}} t$ since $\Delta^2 \subseteq \overline{\Delta}^2$.

Thus, let $s \vdash_{\mathcal{E}} t$ and perform an induction on the position p where the step takes places. If $\text{root}(s) \notin \Delta^2$, then $\text{root}(t) \notin \Delta^2$ as well by the definition of Δ^2 . Since $s \sim_{\mathcal{E}} t$, the strong \mathcal{E} -coherence of $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ implies that whenever $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}} s'$, then $t \rightarrow_{\mathcal{E} \setminus \mathcal{S}} t'$ for some $t' \sim_{\mathcal{E}} s'$. Thus, $\mathcal{R}ed_{\mathcal{S}}^2(s) \subseteq \mathcal{R}ed_{\mathcal{S}}^2(t)$. Similarly, Lemma 11.10.2 implies $\mathcal{R}ed_{\mathcal{R}}^2(s) \subseteq \mathcal{R}ed_{\mathcal{R}}^2(t)$. Finally, if $s \sim_{\mathcal{E}} g(s_1, \dots, s_n)$, then $t \sim_{\mathcal{E}} g(s_1, \dots, s_n)$, which immediately implies $\mathcal{E}q_{\mathcal{E}}^2(s) = \mathcal{E}q_{\mathcal{E}}^2(t)$. Also, $\overline{\mathcal{E}q}_{\mathcal{E}}^2(s) = \overline{\mathcal{E}q}_{\mathcal{E}}^2(t)$ since $s \sim_{\mathcal{E}} t$. Using these properties, $\mathcal{R}ed_{\mathcal{S}}^2(s) \cup \mathcal{R}ed_{\mathcal{R}}^2(s) \cup \mathcal{E}q_{\mathcal{E}}^2(s) \cup \overline{\mathcal{E}q}_{\mathcal{E}}^2(s) \subseteq \mathcal{R}ed_{\mathcal{S}}^2(t) \cup \mathcal{R}ed_{\mathcal{R}}^2(t) \cup \mathcal{E}q_{\mathcal{E}}^2(t) \cup \overline{\mathcal{E}q}_{\mathcal{E}}^2(t)$. Since the same reasoning can be applied with s and t interchanged, $\text{Comp}_{\text{sort}(s)}(\mathcal{R}ed_{\mathcal{S}}^2(s) \cup \mathcal{R}ed_{\mathcal{R}}^2(s) \cup \mathcal{E}q_{\mathcal{E}}^2(s) \cup \overline{\mathcal{E}q}_{\mathcal{E}}^2(s)) = \text{Comp}_{\text{sort}(t)}(\mathcal{R}ed_{\mathcal{S}}^2(t) \cup \mathcal{R}ed_{\mathcal{R}}^2(t) \cup \mathcal{E}q_{\mathcal{E}}^2(t) \cup \overline{\mathcal{E}q}_{\mathcal{E}}^2(t))$ and thus $\mathcal{I}^2(s) = \mathcal{I}^2(t)$.

Otherwise, $\text{root}(s) \in \Delta^2$. If $p = \Lambda$, then there exist an equation $u \approx v$ (or $v \approx u$) in \mathcal{E} and a substitution σ such that $s = u\sigma$ and $t = v\sigma$. By the definition of Δ^2 , $u, v \in \mathcal{T}(\Delta^2, \mathcal{V})$. Since \mathcal{E} is strongly conservative, $u\sigma_u = [u, \sigma]$ and $v\sigma_v = [v, \sigma]$ by Lemma A.11. Moreover, for all variables x , Definition 11.7.2a implies $\sigma_u(x) = \sigma_v(x)$, i.e., $\sigma_u = \sigma_v$. Hence,

Appendix A. Proofs

$$\begin{aligned}
\mathcal{I}^2(s) &= \mathcal{I}^2(u\sigma) \\
&= [u, \sigma] && \text{by } 2 \\
&= u\sigma_u \\
&= u\sigma_v \\
&\vdash_{\mathcal{E}(\Delta^2)} v\sigma_v \\
&= [v, \sigma] \\
&= \mathcal{I}^2(v\sigma) && \text{by } 2 \\
&= \mathcal{I}^2(t)
\end{aligned}$$

If $p \neq \Lambda$ then $s = f(s_1, \dots, s_i, \dots, s_n)$, $t = f(s_1, \dots, t_i, \dots, s_n)$, and $s_i \vdash_{\mathcal{E}} t_i$. Notice that $\mathcal{I}^2(s) = f(\overline{s}_1, \dots, \overline{s}_i, \dots, \overline{s}_n)$ and $\mathcal{I}^2(t) = f(\overline{s}_1, \dots, \overline{t}_i, \dots, \overline{s}_n)$. If $i \in \mu(f)$, then $\overline{s}_i = \mathcal{I}^2(s_i) \vdash_{\overline{\mathcal{E}}(\Delta^2)} \mathcal{I}^2(t_i) = \overline{t}_i$ by the inductive assumption. If $i \notin \mu(f)$, then $\overline{s}_i = \overline{\mathcal{I}^2}(s_i) \vdash_{\overline{\mathcal{E}}(\Delta^2)} \overline{\mathcal{I}^2}(t_i) = \overline{t}_i$ by 6. Thus, $\mathcal{I}^2(s) \vdash_{\overline{\mathcal{E}}(\Delta^2)} \mathcal{I}^2(t)$ in either case.

8. It suffices to show that $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}} t$ implies $\overline{\mathcal{I}^2}(s) \rightsquigarrow_1 \overline{\mathcal{I}^2}(t)$ since the statement then follows by induction on the number of $\rightarrow_{\mathcal{E} \setminus \mathcal{S}}$ -steps in $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}}^* t$.

Thus, let $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}} t$ and perform an induction on the position p where the reduction takes places. If $\text{root}(s) \notin \overline{\Delta}^2$, then $\overline{\mathcal{I}^2}(t) \in \overline{\mathcal{R}ed}_{\mathcal{S}}^2(s)$, which implies $\overline{\mathcal{I}^2}(s) \rightarrow_{\overline{\mathcal{R}}_{\Pi}}^+ \overline{\mathcal{I}^2}(t)$ by Lemma 8.16.

If $\text{root}(s) \in \overline{\Delta}^2$, first consider the case $p = \Lambda$. Then, there exist a rule $l \rightarrow r \in \mathcal{S}$ and a substitution σ such that $s \sim_{\mathcal{E}} l\sigma \rightarrow_{\mathcal{S}} r\sigma = t$. Since $\text{root}(s) \in \overline{\Delta}^2$, the definition of $\overline{\Delta}^2$ implies that $\text{root}(l) \in \overline{\Delta}^2$, $l \rightarrow r \in \mathcal{S}(\overline{\Delta}^2)$, and $r \in \mathcal{T}(\overline{\Delta}^2, \mathcal{V})$. Using 1, 3, and 6, $\overline{\mathcal{I}^2}(s) \sim_{\mathcal{E}(\overline{\Delta}^2)} \overline{\mathcal{I}^2}(l\sigma) \rightarrow_{\overline{\mathcal{R}}_{\Pi}}^* l\overline{\mathcal{I}^2}(\sigma) \rightarrow_{\mathcal{S}(\overline{\Delta}^2)} r\overline{\mathcal{I}^2}(\sigma) = \overline{\mathcal{I}^2}(r\sigma) = \overline{\mathcal{I}^2}(t)$, and thus $\overline{\mathcal{I}^2}(s) \rightsquigarrow_1 \overline{\mathcal{I}^2}(t)$. If $\text{root}(s) \in \overline{\Delta}^2$ and $p \neq \Lambda$, then $s = f(s_1, \dots, s_i, \dots, s_n)$ and $t = f(s_1, \dots, t_i, \dots, s_n)$, where $s_i \rightarrow_{\mathcal{E} \setminus \mathcal{S}} t_i$. By the inductive assumption, $\overline{\mathcal{I}^2}(s_i) \rightsquigarrow_1 \overline{\mathcal{I}^2}(t_i)$ and therefore $\overline{\mathcal{I}^2}(s) = f(\overline{\mathcal{I}^2}(s_1), \dots, \overline{\mathcal{I}^2}(s_i), \dots, \overline{\mathcal{I}^2}(s_n)) \rightsquigarrow_1 f(\overline{\mathcal{I}^2}(s_1), \dots, \overline{\mathcal{I}^2}(t_i), \dots, \overline{\mathcal{I}^2}(s_n)) = \overline{\mathcal{I}^2}(t)$.

9. It suffices to show that $s \rightarrow_{\mathcal{E} \setminus \mathcal{S}} t$ implies $\mathcal{I}^2(s) \rightsquigarrow_1 \mathcal{I}^2(t)$ since the statement

Appendix A. Proofs

then follows by induction on the number of $\rightarrow_{\mathcal{E}\setminus\mathcal{S}}$ -steps in $s \rightarrow_{\mathcal{E}\setminus\mathcal{S}}^* t$.

Thus, let $s \rightarrow_{\mathcal{E}\setminus\mathcal{S}} t$ and perform an induction on the position p where the reduction takes places. If $\text{root}(s) \notin \Delta^2$, then $\mathcal{I}^2(t) \in \mathcal{Red}_{\mathcal{S}}^2(s)$, which implies $\mathcal{I}^2(s) \rightarrow_{\mathcal{R}_{\Pi}}^+ \mathcal{I}^2(t)$ by Lemma 8.16.

If $\text{root}(s) \in \Delta^2$, first consider the case $p = \Lambda$. Then, there exist a rule $l \rightarrow r \in \mathcal{S}$ and a substitution σ such that $s \sim_{\mathcal{E}} l\sigma \rightarrow_{\mathcal{S}} r\sigma = t$. By the definitions of $\overline{\Delta}^2$ and Δ^2 , $\text{root}(l) \in \Delta^2$, $l \rightarrow r \in \mathcal{S}(\overline{\Delta}^2)$, and $r \in \mathcal{T}(\Delta^2, \mathcal{V})$. Since $\mathcal{S}(\overline{\Delta}^2)$ is strongly conservative, $l\sigma_l = [l, \sigma]$ and $r\sigma_r = [r, \sigma]$ by Lemma A.11. Moreover, $\sigma_l(x) \rightarrow_{\mathcal{R}_{\Pi}}^* \sigma_r(x)$ for all variables x . To see this, notice that by strong conservativeness of $\mathcal{S}(\overline{\Delta}^2)$, the substitutions σ_l and σ_r differ at most on variables $x \in \mathcal{V}^{\mu}(l) - \mathcal{V}^{\mu}(r)$. For these variables $\sigma_l(x) = \mathcal{I}^2(\sigma(x)) \rightarrow_{\mathcal{R}_{\Pi}}^* \overline{\mathcal{I}}^2(\sigma(x)) = \sigma_r(x)$ by 5. Hence,

$$\begin{aligned}
\mathcal{I}^2(s) &\sim_{\mathcal{E}(\overline{\Delta}^2)} \mathcal{I}^2(l\sigma) && \text{by } 7 \\
&\rightarrow_{\mathcal{R}_{\Pi}}^* [l, \sigma] && \text{by } 4 \\
&= l\sigma_l \\
&\rightarrow_{\mathcal{R}_{\Pi}}^* l\sigma_r \\
&\rightarrow_{\mathcal{S}(\overline{\Delta}^2)} r\sigma_r \\
&= [r, \sigma] \\
&= \mathcal{I}^2(r\sigma) && \text{by } 2 \\
&= \mathcal{I}^2(t)
\end{aligned}$$

If $p \neq \Lambda$, then $s = f(s_1, \dots, s_i, \dots, s_n)$, $t = f(s_1, \dots, t_i, \dots, s_n)$, and $s_i \rightarrow_{\mathcal{E}\setminus\mathcal{S}} t_i$. Also, $\mathcal{I}^2(s) = f(\overline{s}_1, \dots, \overline{s}_i, \dots, \overline{s}_n)$ and $\mathcal{I}^2(t) = f(\overline{s}_1, \dots, \overline{t}_i, \dots, \overline{s}_n)$. If $i \in \mu(f)$, then $\overline{s}_i = \mathcal{I}^2(s_i) \rightsquigarrow_1 \mathcal{I}^2(t_i) = \overline{t}_i$ by the inductive assumption. If $i \notin \mu(f)$, then $\overline{s}_i = \overline{\mathcal{I}}^2(s_i) \rightsquigarrow_1 \overline{\mathcal{I}}^2(t_i) = \overline{t}_i$ by 8. Thus, $\mathcal{I}^2(s) \rightsquigarrow_1 \mathcal{I}^2(t)$ in either case.

10. It suffices to show that $s \xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}, \mu} t$ implies $\mathcal{I}^2(s) \rightsquigarrow_2 \mathcal{I}^2(t)$ since the statement then follows by induction on the number of $\xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}, \mu}$ -steps in $s \xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}, \mu}^* t$.

Thus, let $s \xrightarrow{\mathcal{S}}_{\mathcal{T}h\|\mathcal{E}\setminus\mathcal{R}, \mu} t$ and perform an induction on the position p where the reduction takes places. If $\text{root}(s) \notin \Delta^2$, then $\mathcal{I}^2(t) \in \mathcal{Red}_{\mathcal{R}}^1(s)$, which implies

Appendix A. Proofs

$\mathcal{I}^2(s) \rightarrow_{\mathcal{R}_\Pi}^+ \mathcal{I}^2(t)$ by Lemma 8.16.

If $\text{root}(s) \in \Delta^2$, first consider the case $p = \Lambda$. Then, there exist a rule $l \rightarrow r[\llbracket\varphi\rrbracket] \in \mathcal{R}$ and a \mathcal{Th} -based substitution σ with $s = f(s^*) \xrightarrow{>\Lambda!}_{\mathcal{E}\setminus\mathcal{S}} \circ \overset{>\Lambda}{\sim}_{\mathcal{E}} l\sigma \rightarrow_{\mathcal{R}} r\sigma = t$ such that $\varphi\sigma$ is \mathcal{Th} -valid. Since $\text{root}(l) = \text{root}(s) = f$ and $f \in \Delta^2$, the definitions of $\overline{\Delta^2}$ and Δ^2 implies that $l \rightarrow r[\llbracket\varphi\rrbracket] \in \mathcal{R}(\Delta^2)$, $r \in \mathcal{F}(\overline{\Delta^2}, \mathcal{V})$, and $\mathcal{F}^\mu(r) \subseteq \Delta^2$. Since $\mathcal{R}(\Delta^2)$ is strongly conservative, $l\sigma_l = [l, \sigma]$ and $r\sigma_r = [r, \sigma]$ by Lemma A.11. Moreover, $\sigma_l(x) \rightarrow_{\mathcal{R}_\Pi}^* \sigma_r(x)$ for all variables x . To see this, notice that by strong conservativeness of $\mathcal{R}(\Delta^2)$, the substitutions σ_l and σ_r differ at most on variables $x \in \mathcal{V}^\mu(l) - \mathcal{V}^\mu(r)$. For these variables, $\sigma_l(x) = \mathcal{I}^2(\sigma(x)) \rightarrow_{\mathcal{R}_\Pi}^* \overline{\mathcal{I}^2}(\sigma(x)) = \sigma_r(x)$ by 5. Also, notice that $\varphi\sigma_r$ is \mathcal{Th} -valid since $\sigma_r(x) = \sigma(x)$ for variables of sort **base** since σ is \mathcal{Th} -based. Hence,

$$\begin{aligned}
\mathcal{I}^2(s) &\rightsquigarrow_1^* \circ \sim_{\mathcal{E}(\overline{\Delta^2})} \mathcal{I}^2(l\sigma) && \text{by 7 and 9} \\
&\rightarrow_{\mathcal{R}_\Pi}^* [l, \sigma] && \text{by 4} \\
&= l\sigma_l \\
&\rightarrow_{\mathcal{R}_\Pi}^* l\sigma_r \\
&\rightarrow_{\mathcal{R}(\Delta^2), \mu} r\sigma_r \\
&= [r, \sigma] \\
&= \mathcal{I}^2(r\sigma) && \text{by 2} \\
&= \mathcal{I}^2(t)
\end{aligned}$$

If $p \neq \Lambda$, then $s = f(s_1, \dots, s_i, \dots, s_n)$, $t = f(s_1, \dots, t_i, \dots, s_n)$, and there exists $i \in \mu(f)$ such that $s_i \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}, \mu} t_i$. Also, $\mathcal{I}^2(s) = f(\overline{s}_1, \dots, \overline{s}_i, \dots, \overline{s}_n)$ and $\mathcal{I}^2(t) = f(\overline{s}_1, \dots, \overline{t}_i, \dots, \overline{s}_n)$ where $\overline{s}_i = \mathcal{I}^2(s_i) \rightsquigarrow_2 \mathcal{I}^2(t_i) = \overline{t}_i$ by the inductive assumption.

11. If $s \xrightarrow{\mathcal{S}}_{\mathcal{Th}\|\mathcal{E}\setminus\mathcal{R}, \mu}^* \rightarrow_{\mathcal{E}\setminus\mathcal{S}}^! \circ \sim_{\mathcal{E}} t$, then $\mathcal{I}^2(s) \rightsquigarrow_2^* \circ \rightsquigarrow_1^* \circ \sim_{\mathcal{E}(\overline{\Delta^2})} \mathcal{I}^2(t)$ using 7, 9, and 10. \square

Proof of Theorem 12.26. In the second case soundness is obvious. Otherwise, it needs to be shown that every infinite minimal $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -chain contains only

Appendix A. Proofs

finitely many dependency pairs from \mathcal{P}' . Let $s_1 \rightarrow t_1[\varphi_1], s_2 \rightarrow t_2[\varphi_2], \dots$ be an infinite minimal $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mu)$ -chain with the \mathcal{Th} -based substitution σ . Thus, $t_i \sigma \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}, \mu}^* \circ \xrightarrow{> \Lambda} \mathcal{E} \setminus \mathcal{S} \circ \gtrsim_{\mathcal{E}}^{\Lambda} s_{i+1} \sigma$ and $\varphi_i \sigma$ is \mathcal{Th} -valid for all $i \geq 1$. By the definition of Δ^2 , $\text{root}(t'_i) \in \Delta^2$ for all $t'_i \notin \mathcal{V}$ such that $t_i \triangleright_{\mu} t'_i$. Similarly to the proof of Lemma 12.25.11 and using Lemma 12.25, $t_i \sigma_{t_i} = [t_i, \sigma] = \mathcal{I}^2(t_i \sigma) \rightsquigarrow_2^* \circ \rightsquigarrow_1^* \circ \sim_{\mathcal{E}(\overline{\Delta}^2)}$ $\mathcal{I}^2(s_{i+1} \sigma) \rightarrow_{\mathcal{R}_{\Pi}}^* [s_{i+1}, \sigma] = s_{i+1} \sigma_{s_{i+1}} \rightarrow_{\mathcal{R}_{\Pi}}^* s_{i+1} \sigma_{t_{i+1}}$ since \mathcal{P} is strongly conservative.

As in the previous proofs, $t_i \sigma_{t_i} \gtrsim s_{i+1} \sigma_{t_{i+1}}$. Notice that $s_i \sigma_{t_i} \gtrsim t_i \sigma_{t_i}$ or $s_i \sigma_{t_i} \succ t_i \sigma_{t_i}$ for all $i \geq 1$ since $\varphi_i \sigma_{t_i} = \varphi_i \sigma$ is \mathcal{Th} -valid and $s_i[\varphi_i] \gtrsim t_i[\varphi_i]$ for all $s_i \rightarrow t_i[\varphi_i] \in \mathcal{P} - \mathcal{P}'$ and $s_i[\varphi_i] \succ t_i[\varphi_i]$ for all $s_i \rightarrow t_i[\varphi_i] \in \mathcal{P}'$. Hence, the infinite minimal chain gives rise to

$$s_1 \sigma_{t_1} \bowtie_1 t_1 \sigma_{t_1} \gtrsim s_2 \sigma_{t_2} \bowtie_2 t_2 \sigma_{t_2} \gtrsim \dots$$

where $\bowtie_i \in \{\gtrsim, \succ\}$. If the above infinite minimal chain contains infinitely many dependency pairs from \mathcal{P}' , then $\bowtie_i = \succ$ for infinitely many i . In this case, the compatibility of \succ with \gtrsim produces an infinite \succ chain, contradicting the well-foundedness of \succ . Thus, only finitely many dependency pairs from \mathcal{P}' occur in the above infinite minimal chain. \square

A.12 Proofs from Chapter 13

Proof of Lemma 13.7. Let t be a \mathbb{Z} -free term.

1. t is trivially \mathbb{Z} -normal since it does not contain any occurrence of $+$ or $-$, which implies that t satisfies the conditions of Definition 13.5.
2. Obvious. \square

Proof sketch of Lemma 13.8. In a first step, a term t' with $t \rightarrow_{\mathcal{E}_{\mathcal{Th}_{\mathbb{Z}}} \setminus \mathcal{S}_{\mathcal{Th}_{\mathbb{Z}}}}^! t'$ is computed. Then, t' is brought into right-associated form such that the arguments

Appendix A. Proofs

of nested occurrences of $+$ are sorted w.r.t. $>_{\mathcal{T}}$. Formally, this can be achieved by suitably applying the equations from $\mathcal{E}_{\mathcal{Th}_{\mathbb{Z}}}$. \square

Proof of Lemma 13.11. Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a \mathbb{Z} -CERS and let s be a \mathbb{Z} -normal term such that $s \rightarrow_{\mathcal{R}, \mathbb{Z}} t$. Let p be the position used for $s \rightarrow_{\mathcal{R}, \mathbb{Z}} t$, i.e., $s|_p = l\sigma$ for some $l \rightarrow r[[\varphi]] \in \mathcal{R}$.

1. Since $\text{sort}(l) \neq \text{int}$, also $\text{sort}(s|_p) \neq \text{int}$ and the path from Λ to p in s does not contain any function symbol with resulting sort int since \mathcal{F} does not contain any function symbols with resulting sort int . Thus, since s is \mathbb{Z} -normal, the only subterm of $t = s[r\sigma]_p$ that might prevent t from being \mathbb{Z} -normal is $r\sigma$. Notice that $\sigma(x)$ is \mathbb{Z} -normal for all $x \in \mathcal{V}(r)$. Thus, $r\sigma$ is \mathbb{Z} -normal by Lemma 13.7 since r is \mathbb{Z} -free.
2. Obvious since rewriting does not introduce new function symbols that do not occur in the rewrite rules.
3. Since s is \mathbb{Z} -normal, $s|_p$ is \mathbb{Z} -normal and thus $s|_p \xrightarrow{\geq \Lambda!}_{\mathcal{E} \setminus \mathcal{S}} l\sigma \circ \approx_{\mathcal{E}}^{\geq \Lambda} l\sigma$. The remaining conditions in the definitions of $\rightarrow_{\mathcal{R}, \mathbb{Z}}$ and $\xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}}$ are identical and thus $s \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} t$. \square

Proof of Lemma 13.12. If $s \xrightarrow{\mathcal{S}}_{\mathcal{Th} \parallel \mathcal{E} \setminus \mathcal{R}} t$, then $s|_p \xrightarrow{\geq \Lambda!}_{\mathcal{E} \setminus \mathcal{S}} l\sigma \circ \approx_{\mathcal{E}}^{\geq \Lambda} l\sigma$, $r = s[r\sigma]_p$, and $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid for some position $p \in \mathcal{Pos}(s)$ and some $l \rightarrow r[[\varphi]] \in \mathcal{R}$. Since $s|_p \leftrightarrow_{\mathcal{E}_{\mathcal{Th}_{\mathbb{Z}}} \cup \mathcal{S}_{\mathcal{Th}_{\mathbb{Z}}}}^* l\sigma$, $\text{norm}(s|_p) = \text{norm}(l\sigma)$ by Lemma 13.8. Since l is \mathbb{Z} -free, $\text{norm}(l\sigma) = l\text{norm}(\sigma)$ and thus $\text{norm}(s|_p) = l\text{norm}(\sigma)$. Notice that p is also a position in $\text{norm}(s)$ because $\text{sort}(s|_p) \neq \text{int}$. Furthermore, $\text{norm}(s)|_p = \text{norm}(s|_p)$, which implies $\text{norm}(s)|_p = l\text{norm}(\sigma)$. Since $\varphi\text{norm}(\sigma)$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid, $\text{norm}(s) \rightarrow_{\mathcal{R}, \mathbb{Z}} \text{norm}(s)[r\text{norm}(\sigma)]_p = \text{norm}(s[r\text{norm}(\sigma)]_p) = \text{norm}(s[r\sigma]_p) = \text{norm}(t)$. \square

Proof of Theorem 13.20. Let \mathcal{R} be a normal \mathbb{Z} -CERS and let $\overline{\mathcal{R}} = \{l \rightarrow r \mid l \rightarrow r[[\varphi]] \in \mathcal{R}\}$ be the ordinary TRS obtained from \mathcal{R} by dropping the constraints.

First, it is shown that \mathcal{R} is quasi-reductive iff $\overline{\mathcal{R}}$ is quasi-reductive. The direction

Appendix A. Proofs

from left to right is immediate since $\rightarrow_{\mathcal{R},\mathbb{Z}} \subseteq \rightarrow_{\overline{\mathcal{R}}}$. For the direction from right to left, let $f(t_1, \dots, t_n)$ be a \mathbb{Z} -normal ground term with $f \in \mathcal{D}(\mathcal{R})$ and $t_1, \dots, t_n \in \mathcal{T}(\mathcal{C}(\mathcal{R}) \cup \mathcal{F}_{\mathcal{Th}_{\mathbb{Z}}})$. Since $\overline{\mathcal{R}}$ is quasi-reductive, there exists a rule $l \rightarrow r \in \overline{\mathcal{R}}$ such that $f(t_1, \dots, t_n) = l\sigma$. By Definition 13.18.4, there exists a rule $l \rightarrow r'[\![\varphi]\!] \in \mathcal{R}$ such that $f(t_1, \dots, t_n) = l\sigma$ and $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid. Therefore, $f(t_1, \dots, t_n)$ is reducible by $\rightarrow_{\mathcal{R},\mathbb{Z}}$.

It thus suffices to determine whether $\overline{\mathcal{R}}$ is quasi-reductive. But this can easily be done, for instance using the narrowing-based method for left-linear constructor-based ordinary TRSs in [78, 94] that furthermore computes a set of missing patterns, i.e., left-hand sides for rules that need to be added in order to make $\overline{\mathcal{R}}$ quasi-reductive. \square

Proof sketch of Theorem 13.21. It will be shown below that normal \mathbb{Z} -CERSs satisfy the following property:

Whenever $s \rightarrow_{\mathcal{R},\mathbb{Z}} t_1$ and $s \rightarrow_{\mathcal{R},\mathbb{Z}} t_2$ such that the reductions take place at the same position, then $t_1 = t_2$.

Thus, normal \mathbb{Z} -CERSs satisfy the crucial property needed in order to show confluence of orthogonal ordinary TRSs and the proof used for this result in [17, Corollary 6.3.11] applies for normal \mathbb{Z} -CERSs as well since normal \mathbb{Z} -CERSs are left-linear.

To show the above property, it can without loss of generality be assumed that the reductions $s \rightarrow_{\mathcal{R},\mathbb{Z}} t_1$ and $s \rightarrow_{\mathcal{R},\mathbb{Z}} t_2$ take place at the root position. Thus, there exist rules $l_1 \rightarrow r_1[\![\varphi_1]\!]$, $l_2 \rightarrow r_2[\![\varphi_2]\!]$ and substitutions σ_1, σ_2 such that $s = l_1\sigma_1 = l_2\sigma_2$ and $\varphi_1\sigma_1, \varphi_2\sigma_2$ are $\mathcal{Th}_{\mathbb{Z}}$ -valid. By Definition 13.18.2, $l_1 = l_2$ and therefore $\sigma_1 = \sigma_2$. Now, Definition 13.18.3 implies that the rules $l_1 \rightarrow r_1[\![\varphi_1]\!]$ and $l_2 \rightarrow r_2[\![\varphi_2]\!]$ are identical, i.e., $t_1 = r_1\sigma_1 = r_2\sigma_2 = t_2$. \square

Proof of Lemma 13.26. Consider a ground substitution σ . Since \mathcal{R} is quasi-reductive, there exists a constructor ground substitution $\widehat{\sigma}$ such that $\sigma(x) \rightarrow_{\mathcal{R},\mathbb{Z}}^* \widehat{\sigma}(x)$ for all $x \in \mathcal{V}$. Then $s\sigma \rightarrow_{\mathcal{R},\mathbb{Z}}^* s\widehat{\sigma} \leftrightarrow_{\mathcal{E}_{\mathcal{Th}_{\mathbb{Z}}} \cup \mathcal{S}_{\mathcal{Th}_{\mathbb{Z}}}}^* \text{norm}(s\widehat{\sigma}) = \text{snorm}(\widehat{\sigma}) \leftrightarrow_{\mathcal{R},\mathbb{Z}}^* t\text{norm}(\widehat{\sigma}) =$

Appendix A. Proofs

$\text{norm}(t\hat{\sigma}) \leftrightarrow_{\mathcal{E}_{\mathcal{Th}_{\mathbb{Z}} \cup \mathcal{S}_{\mathcal{Th}_{\mathbb{Z}}}}}^* t\hat{\sigma} \leftarrow_{\mathcal{R}, \mathbb{Z}}^* t\sigma$ by Lemma 13.7, Lemma 13.8, and the assumption. For this, notice that $\text{norm}(\hat{\sigma})$ is a \mathbb{Z} -normal constructor ground substitution. \square

Proof of Lemma 13.31. Since both u and l are \mathbb{Z} -free, $\sigma(x)$ is \mathbb{Z} -free for all variables x . But then $C[r]\sigma$ and $t\sigma$ are \mathbb{Z} -free as well. \square

Proof of Lemma 13.32. Let \mathcal{R} be a quasi-reductive \mathbb{Z} -CERS, let $s \equiv t[\![\varphi]\!]$ be an atomic constraint, and let u be a basic term such that $s = C[u]$.

1. Since u is basic and σ is a \mathbb{Z} -normal constructor ground substitution, the term $u\sigma$ has the form $f(u_1, \dots, u_n)$ where u_1, \dots, u_n are \mathbb{Z} -normal constructor ground terms. Thus, since \mathcal{R} is quasi-reductive, there exists a rule $l \rightarrow r[\![\psi]\!]$ $\in \mathcal{R}$ such that $u\sigma$ is an instance of l , i.e., $u\sigma = l\tau$ for some τ . Furthermore, $\psi\tau$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid. Without loss of generality it can be assumed that s and l are variable-disjoint, and the substitution σ can thus be extended to obtain $u\sigma = l\sigma$. Furthermore, $\psi\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid. Since σ is a unifier of u and l , there exists a substitution θ such that $\sigma = \iota\theta$ where $\iota = \text{mgu}(u, l)$. Thus, $s\sigma = C[u]\sigma = C[u]\iota\theta = C\iota\theta[u\iota\theta] = C\iota\theta[l\iota\theta] \rightarrow_{\mathcal{R}, \mathbb{Z}} C\iota\theta[r\iota\theta] \leftrightarrow_{\text{Expd}_u(s, t, \varphi), \mathbb{Z}} t\iota\theta = t\sigma$. For this, notice that $\varphi\iota\theta = \varphi\sigma$ and $\psi\iota\theta = \psi\sigma$ are $\mathcal{Th}_{\mathbb{Z}}$ -valid.
2. Let $v \leftrightarrow_{\text{Expd}_u(s, t, \varphi), \mathbb{Z}} w$. Then $v = \widehat{C}[C[r]\sigma\hat{\sigma}]$ and $w = \widehat{C}[t\sigma\hat{\sigma}]$ (or $w = \widehat{C}[C[r]\sigma\hat{\sigma}]$ and $v = \widehat{C}[t\sigma\hat{\sigma}]$) for some context \widehat{C} and some substitution $\hat{\sigma}$, where $\sigma = \text{mgu}(u, l)$, $s = C[u]$, $l \rightarrow r[\![\psi]\!]$ $\in \mathcal{R}$, and $\varphi\sigma\hat{\sigma} \wedge \psi\sigma\hat{\sigma}$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid. Then $v = \widehat{C}[C[r]\sigma\hat{\sigma}] \leftarrow_{\mathcal{R}, \mathbb{Z}} \widehat{C}[C[l]\sigma\hat{\sigma}] = \widehat{C}[C\sigma[l\sigma]\hat{\sigma}] = \widehat{C}[C\sigma[u\sigma]\hat{\sigma}] = \widehat{C}[C[u]\sigma\hat{\sigma}] = \widehat{C}[s\sigma\hat{\sigma}] \rightarrow_{\{s \rightarrow t[\![\varphi]\!]\}, \mathbb{Z}} \widehat{C}[t\sigma\hat{\sigma}] = w$. \square

Proof of Lemma 13.37. Perform a case analysis on the inference rule that is applied in $\langle E_n, H_n \rangle \vdash_{\mathcal{I}} \langle E_{n+1}, H_{n+1} \rangle$. If this inference rule is **Simplify**, then $E_n = E \uplus \{s \equiv t[\![\varphi]\!]\}$, $E_{n+1} = E \cup \{s' \equiv t[\![\varphi]\!]\}$, and $H_{n+1} = H_n$, where $s[\![\varphi]\!] \rightarrow_{\mathcal{R} \cup H, \mathbb{Z}} s'[\![\varphi]\!]$. First, consider the inclusion “ \subseteq ”. For this, let $v \leftrightarrow_{\mathcal{R} \cup E_n \cup H_n, \mathbb{Z}} w$ for \mathbb{Z} -normal ground terms v, w . If $v \leftrightarrow_{\mathcal{R} \cup E \cup H_n, \mathbb{Z}} w$, then $v \leftrightarrow_{\mathcal{R} \cup E_{n+1} \cup H_{n+1}, \mathbb{Z}} w$ is immediate. Otherwise,

Appendix A. Proofs

$v = C[s\sigma]$, $w = C[t\sigma]$ (or $w = C[s\sigma]$, $v = C[t\sigma]$), and $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid for a ground substitution σ . Now $s[\![\varphi]\!] \rightarrow_{\mathcal{R} \cup H_n, \mathbb{Z}} s'[\![\varphi]\!]$ implies that $s = D[l\tau]$, $s' = D[r\tau]$, and $\varphi \Rightarrow \psi\tau$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid for some $l \rightarrow r[\![\psi]\!] \in \mathcal{R} \cup H_n = \mathcal{R} \cup H_{n+1}$ and some ground substitution τ . Since $\varphi \Rightarrow \psi\tau$ and $\varphi\sigma$ are $\mathcal{Th}_{\mathbb{Z}}$ -valid, $\psi\tau\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid as well. Therefore, $v = C[s\sigma] = C[D\sigma[l\tau\sigma]] \rightarrow_{\mathcal{R} \cup H_{n+1}, \mathbb{Z}} C[D\sigma[r\tau\sigma]] = C[s'\sigma] \rightarrow_{E_{n+1}, \mathbb{Z}} C[t\sigma] = w$. For the inclusion “ \supseteq ”, it again suffices to consider the case where $v = C[s'\sigma]$, $w = C[t\sigma]$ (or $w = C[s'\sigma]$, $v = C[t\sigma]$), and $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid. Similar to above, $v = C[s'\sigma] = C[D\sigma[r\tau\sigma]] \leftarrow_{\mathcal{R} \cup H_n, \mathbb{Z}} C[D\sigma[l\tau\sigma]] = C[s\sigma] \rightarrow_{E_n, \mathbb{Z}} C[t\sigma] = w$.

For Delete, the inclusion “ \supseteq ” is obvious since an atomic conjecture is removed from E_n . For “ \subseteq ”, it suffices to notice that $v \leftrightarrow_{\{s \doteq s[\![\varphi]\!]\}, \mathbb{Z}} w$ for \mathbb{Z} -normal ground terms v and w implies $v = w$.

For Theory₁, the inclusion “ \supseteq ” is again obvious. For “ \subseteq ”, let $v \leftrightarrow_{\{s \doteq t[\![\varphi]\!]\}, \mathbb{Z}} w$ for \mathbb{Z} -normal ground terms v and w . Thus, there exists a ground substitution σ such that $v = C[s\sigma]$, $w = C[t\sigma]$ (or $w = C[s\sigma]$, $v = C[t\sigma]$), and $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid. Since $\varphi \Rightarrow s \simeq t$ is LIAC-valid and $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid, $s\sigma \simeq t\sigma$ is LIAC-valid. Since σ is \mathbb{Z} -normal, this implies $s\sigma = t\sigma$ and thus $v = w$.

For Expand, the inclusion “ \subseteq ” is obvious. For “ \supseteq ”, it suffices to show that $v \leftrightarrow_{\text{Expd}_u(s,t,\varphi), \mathbb{Z}} w$ implies $v \leftrightarrow_{\mathcal{R} \cup E_n \cup H_n, \mathbb{Z}}^* w$ for all \mathbb{Z} -normal ground terms v, w . But this follows from Lemma 13.32.2 since $s \doteq t[\![\varphi]\!] \in E_n$. \square

Proof of Lemma 13.38. The proof is by induction on the length k of the derivation $\langle E_n, H_n \rangle \vdash_{\mathcal{I}}^* \langle \emptyset, H \rangle$. If $k = 0$, then $E_n = \emptyset$ and the claim is obvious. Otherwise, $\langle E_n, H_n \rangle \vdash_{\mathcal{I}} \langle E_{n+1}, H_{n+1} \rangle \vdash_{\mathcal{I}}^* \langle \emptyset, H \rangle$ where $\leftrightarrow_{E_{n+1}, \mathbb{Z}} \subseteq \rightarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* \circ \leftarrow_{\mathcal{R} \cup H, \mathbb{Z}}^*$ by the inductive assumption. Now, perform a case analysis on the inference rule that is applied in $\langle E_n, H_n \rangle \vdash_{\mathcal{I}} \langle E_{n+1}, H_{n+1} \rangle$.

If this inference rule is Simplify, then $E_n = E \uplus \{s \doteq t[\![\varphi]\!]\}$, $E_{n+1} = E \cup \{s' \doteq t[\![\varphi]\!]\}$, and $H_{n+1} = H_n$, where $s[\![\varphi]\!] \rightarrow_{\mathcal{R} \cup H_n, \mathbb{Z}} s'[\![\varphi]\!]$. Let $v \leftrightarrow_{E_n, \mathbb{Z}} w$ for \mathbb{Z} -normal ground

Appendix A. Proofs

terms v, w . If $v \leftrightarrow_{E, \mathbb{Z}} w$, then the claim is immediate from the inductive assumption. Otherwise, $v \leftrightarrow_{\{s \doteq t[\varphi]\}, \mathbb{Z}} w$, i.e., $v = C[s\sigma]$ and $w = C[t\sigma]$ (or $v = C[t\sigma]$ and $w = C[s\sigma]$) where $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid. Since $v' := C[s'\sigma] \leftrightarrow_{E_{n+1}, \mathbb{Z}} w$, the inductive assumption implies $v' \rightarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* \circ \leftarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* w$. It now suffices to show $v \rightarrow_{\mathcal{R} \cup H, \mathbb{Z}} v'$ since then $v \rightarrow_{\mathcal{R} \cup H, \mathbb{Z}} v' \rightarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* \circ \leftarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* w$. For this, recall that $s[\varphi] \rightarrow_{\mathcal{R} \cup H_n, \mathbb{Z}} s'[\varphi]$ implies $s = D[l\tau]$, $s' = D[r\tau]$, and $\varphi \Rightarrow \psi\tau$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid for some $l \rightarrow r[\psi] \in \mathcal{R} \cup H_n$ and some ground substitution τ . As in the proof of Lemma 13.37, $v = C[s\sigma] = C[D\sigma[l\tau\sigma]] \rightarrow_{\mathcal{R} \cup H_n, \mathbb{Z}} C[D\sigma[r\tau\sigma]] = C[s'\sigma] = v'$ and the claim follows since $H_n \subseteq H$.

If the inference rule **Delete** was applied, then $E_n = E \uplus \{s \doteq s[\varphi]\}$, $E_{n+1} = E$, and $H_{n+1} = H_n$. If $v \leftrightarrow_{E, \mathbb{Z}} w$, then the claim follows from the inductive assumption. Otherwise, $v = w$ and the claim is immediate.

If the inference rule **Theory₁** was applied, then $E_n = E \uplus \{s \doteq t[\varphi]\}$, $E_{n+1} = E$, and $H_{n+1} = H_n$, where $\varphi \Rightarrow s \simeq t$ is **LIAC**-valid. Again, if $v \leftrightarrow_{E, \mathbb{Z}} w$, then the claim follows from the inductive assumption. Otherwise, $v = C[s\sigma]$ and $w = C[t\sigma]$ (or $v = C[t\sigma]$ and $w = C[s\sigma]$) where $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid. Thus, $s\sigma \simeq t\sigma$ is **LIAC**-valid, which implies $s\sigma = t\sigma$ since σ is \mathbb{Z} -normal. Hence $v = w$ and thus $v \rightarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* \circ \leftarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* w$.

Finally, assume that the inference rule **Expand** was applied. Thus, $E_n = E \uplus \{s \doteq t[\varphi]\}$, $E_{n+1} = E \cup \text{Expd}_u(s, t, \varphi)$, and $H_{n+1} = H_n$. As in the other cases, $v \leftrightarrow_{E, \mathbb{Z}} w$ is taken care of by the inductive assumption. Otherwise, $v = C[s\sigma]$ and $w = C[t\sigma]$ (or $v = C[t\sigma]$ and $w = C[s\sigma]$) where $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid. By Lemma 13.32.1, $v \rightarrow_{\mathcal{R}, \mathbb{Z}} \circ \leftrightarrow_{\text{Expd}_u(s, t, \varphi), \mathbb{Z}} w$ and thus $v \rightarrow_{\mathcal{R}, \mathbb{Z}} \circ \leftrightarrow_{E_{n+1}, \mathbb{Z}} w$. Now the inductive assumption implies $v \rightarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* \circ \leftarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* w$. \square

Proof of Lemma 13.39. Let $s \rightarrow t[\varphi] \in H$ and $v \rightarrow_{\{s \rightarrow t[\varphi]\}, \mathbb{Z}} w$ for \mathbb{Z} -normal ground terms v and w , i.e., $v = C[s\sigma]$ and $w = C[t\sigma]$ for a \mathbb{Z} -normal ground substitution σ such that $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid.

First, assume that $\sigma(y)$ is reducible by $\rightarrow_{\mathcal{R}, \mathbb{Z}}$ for at least one variable $y \in \mathcal{V}(s) \cup$

Appendix A. Proofs

$\mathcal{V}(t)$. Since $\rightarrow_{\mathcal{R},\mathbb{Z}}$ is terminating, there exists a substitution $\widehat{\sigma}$ such that $\sigma(x) \rightarrow_{\mathcal{R},\mathbb{Z}}^* \widehat{\sigma}(x)$ and $\widehat{\sigma}(x)$ is irreducible by $\rightarrow_{\mathcal{R},\mathbb{Z}}$ for all variables $x \in \mathcal{V}(s) \cup \mathcal{V}(t)$. Then, if $C[s\widehat{\sigma}] \rightarrow_{\mathcal{R},\mathbb{Z}} \circ \rightarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* \circ \leftarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* C[t\widehat{\sigma}]$ has been shown, $v = C[s\sigma] \rightarrow_{\mathcal{R},\mathbb{Z}} \circ \rightarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* \circ \leftarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* C[t\sigma] = w$ follows as well since $C[s\sigma] \rightarrow_{\mathcal{R},\mathbb{Z}}^* C[s\widehat{\sigma}]$ and $C[t\sigma] \leftarrow_{\mathcal{R},\mathbb{Z}}^* C[t\widehat{\sigma}]$.

Thus, it can be assumed that $\sigma(x)$ is irreducible by $\rightarrow_{\mathcal{R},\mathbb{Z}}$ for all $x \in \mathcal{V}(s) \cup \mathcal{V}(t)$. Since \mathcal{R} is quasi-reductive, σ is a \mathbb{Z} -normal constructor ground substitution. There exists an n such that $\langle E, \emptyset \rangle \vdash_{\mathcal{I}}^* \langle E_n, H_n \rangle \vdash_{\mathcal{I}} \langle E_{n+1}, H_{n+1} \rangle \vdash_{\mathcal{I}}^* \langle \emptyset, H \rangle$ where $H_{n+1} = H_n \cup \{s \rightarrow t[\varphi]\}$, $E_n = E'_n \uplus \{s \equiv t[\varphi]\}$, and $E_{n+1} = E'_n \cup \text{Expd}_u(s, t, \varphi)$ such that $\rightarrow_{\mathcal{R} \cup H \cup \{s \rightarrow t[\varphi]\}, \mathbb{Z}}$ is terminating. Then, by Lemma 13.32.1, $s\sigma \rightarrow_{\mathcal{R},\mathbb{Z}} \circ \leftrightarrow_{\text{Expd}_u(s, t, \varphi), \mathbb{Z}} t\sigma$ and thus $v = C[s\sigma] \rightarrow_{\mathcal{R},\mathbb{Z}} \circ \leftrightarrow_{E_{n+1}, \mathbb{Z}} C[t\sigma] = w$. Therefore, $v \rightarrow_{\mathcal{R},\mathbb{Z}} \circ \rightarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* \circ \leftarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* w$ by Lemma 13.38. \square

Proof of Theorem 13.40. By Lemma 13.37, $\leftrightarrow_{\mathcal{R} \cup E, \mathbb{Z}} = \leftrightarrow_{\mathcal{R} \cup H, \mathbb{Z}}$ on \mathbb{Z} -normal ground terms. Thus, it suffices to show that $\leftrightarrow_{\mathcal{R} \cup H, \mathbb{Z}} = \leftrightarrow_{\mathcal{R}, \mathbb{Z}}$ on \mathbb{Z} -normal ground terms. For this, the following principle of Koike and Toyama [115] as reported in [7, 9] is used (where $\rightarrow_1 = \rightarrow_{\mathcal{R}, \mathbb{Z}}$, $\rightarrow_2 = \rightarrow_{H, \mathbb{Z}}$, and A is the set of \mathbb{Z} -normal ground terms):

Let $\rightarrow_1, \rightarrow_2$ be binary relations on a set A and let $\rightarrow_{1 \cup 2} = \rightarrow_1 \cup \rightarrow_2$.

Assume that

1. $\rightarrow_{1 \cup 2}$ is well-founded.
2. $\rightarrow_2 \subseteq \rightarrow_1 \circ \rightarrow_{1 \cup 2}^* \circ \leftarrow_{1 \cup 2}^*$

Then $\leftrightarrow_1^* = \leftrightarrow_{1 \cup 2}^*$.

The first condition, i.e., that $\rightarrow_{\mathcal{R} \cup H, \mathbb{Z}}$ is terminating, follows from the condition of **Expand**. The second condition, i.e., $\rightarrow_{H, \mathbb{Z}} \subseteq \rightarrow_{\mathcal{R}, \mathbb{Z}} \circ \rightarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* \circ \leftarrow_{\mathcal{R} \cup H, \mathbb{Z}}^*$, follows from Lemma 13.39. \square

Proof of Theorem 13.42. Let $\langle E, \emptyset \rangle \vdash_{\mathcal{I}}^* \langle E_n, H_n \rangle \vdash_{\mathcal{I}} \perp$ where $E_n = E'_n \uplus \{s \equiv t[\varphi]\}$

Appendix A. Proofs

such that $\varphi \Rightarrow s \simeq t$ is not LIAC-valid. Therefore, there exists a \mathbb{Z} -normal constructor ground substitution σ such that $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid and $s\sigma \simeq t\sigma$ is not LIAC-valid, which implies $s\sigma \neq t\sigma$ since σ is \mathbb{Z} -normal. Since $s\sigma \leftrightarrow_{E_n, \mathbb{Z}} t\sigma$, Lemma 13.37 implies $s\sigma \leftrightarrow_{\mathcal{R} \cup E, \mathbb{Z}}^* t\sigma$. If all atomic conjectures in E are inductive theorems, then this implies $s\sigma \leftrightarrow_{\mathcal{R}, \mathbb{Z}}^* t\sigma$, which gives $s\sigma \rightarrow_{\mathcal{R}, \mathbb{Z}}^* \circ \leftarrow_{\mathcal{R}, \mathbb{Z}}^* t\sigma$ because $\rightarrow_{\mathcal{R}, \mathbb{Z}}$ is confluent and thus Church-Rosser.³ But $s\sigma \rightarrow_{\mathcal{R}, \mathbb{Z}}^* \circ \leftarrow_{\mathcal{R}, \mathbb{Z}}^* t\sigma$ is not possible since $s\sigma \neq t\sigma$ and both terms are irreducible by $\rightarrow_{\mathcal{R}, \mathbb{Z}}$ since s and t do not contain defined symbols and σ is a \mathbb{Z} -normal constructor ground substitution. \square

A.13 Proofs from Chapter 14

Proof of Theorem 14.6. Let $g(x^*) \equiv t$ be a simple conjecture and consider a rule $g(l^*) \rightarrow C[g(r_1^*), \dots, g(r_m^*)][\varphi] \in \mathcal{R}(g)$. Provided $g(x^*)$ and $g(l^*)$ are unifiable and $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -satisfiable for $\sigma = \text{mgu}(g(x^*), g(l^*))$, application of **Expand** to $g(x^*) \equiv t$ produces (amongst others) $C\sigma[g(r_1^*)\sigma, \dots, g(r_m^*)\sigma] \equiv t\sigma[\varphi\sigma]$. After application of **Expand**, the set H of hypotheses consists of the rule $g(x^*) \rightarrow t$.

Now, if $x_i = x_j$ for $i < j$, then $\langle i, j \rangle \in \text{ImpEq}(g)$. First, since $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -satisfiable, $l_i \simeq l_j \wedge \varphi$ is $\mathcal{Th}_{\mathbb{Z}}$ -satisfiable. Thus, since $x_i = x_j$ implies $l_i\sigma = l_j\sigma$, the definition of **ImpEq** yields $r_{k,i}\sigma = r_{k,j}\sigma$ for all $1 \leq k \leq m$. Hence, **Simplify** applies m times to the conjecture $C\sigma[g(r_1^*)\sigma, \dots, g(r_m^*)\sigma] \equiv t\sigma[\varphi\sigma]$ using the hypothesis $g(x^*) \rightarrow t \in H$ and $C\sigma[t\tau_1, \dots, t\tau_m] \equiv t\sigma[\varphi\sigma]$ is obtained, where $\tau_i = \{x^* \mapsto r_i^*\sigma\}$. Since both sides are from $\mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$, either **Theory₁** or **Theory₂** can be applied to this conjecture. \square

Proof of Theorem 14.12. Let $\{g_1(x_1^*) \equiv t_1, \dots, g_n(x_n^*) \equiv t_n\}$ be a simple set of conjectures. Consider the atomic conjecture $g_k(x_k^*) \equiv t_k$ from this set and the rule $g_k(l^*) \rightarrow C[g_{k_1}(r_1^*), \dots, g_{k_m}(r_m^*)][\varphi] \in \mathcal{R}(\mathcal{G})$. Provided $g_k(x_k^*)$ and $g_k(l^*)$ are

³A relation \rightarrow is Church-Rosser iff $s \leftrightarrow^* t$ implies $s \rightarrow^* \circ \leftarrow^* t$ for all s, t . It is well-known that this property is equivalent to confluence, see, e.g., [17].

Appendix A. Proofs

unifiable and $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -satisfiable for $\sigma = \text{mgu}(g_k(x_k^*), g_k(l^*))$, application of **Expand** to $g_k(x_k^*) \equiv t_k$ produces (amongst others) $C\sigma[g_{k_1}(r_1^*)\sigma, \dots, g_{k_m}(r_m^*)\sigma] \equiv t_k\sigma[\varphi\sigma]$. After application of **Expand** to each atomic conjecture in the simple set of conjectures, the set H consists of the rules $g_1(x_1^*) \rightarrow t_1, \dots, g_n(x_n^*) \rightarrow t_n$.

Now, if $x_{k_\kappa, i_\kappa} = x_{k_\kappa, j_\kappa}$ for any $1 \leq \kappa \leq m$ and $i_\kappa < j_\kappa$, then there exists an $\langle g_{k_\kappa}, i_\kappa, j_\kappa, \Gamma \rangle \in \mathcal{ImpEq}(\mathcal{G})$ such that $x_{k', i'} = x_{k', j'}$ for all $\langle g_{k'}, i', j' \rangle \in \Gamma$. In particular, all such restrictions for g_k are satisfied. As in the proof of Theorem 14.6, the definition of \mathcal{ImpEq} thus implies $r_{\kappa, i_\kappa} = r_{\kappa, j_\kappa}$ since $x_{k, i'} = x_{k, j'}$ implies $l_{i'}\sigma = l_{j'}\sigma$. Thus, **Simplify** applies m times to $C\sigma[g_{k_1}(r_1^*)\sigma, \dots, g_{k_m}(r_m^*)\sigma] \equiv t_k\sigma[\varphi\sigma]$ using the hypotheses $g_1(x_1^*) \rightarrow t_1, \dots, g_n(x_n^*) \rightarrow t_n \in H$ and $C\sigma[t_{k_1}\tau_1, \dots, t_{k_m}\tau_m] \equiv t\sigma[\varphi\sigma]$ is obtained, where $\tau_i = \{x_{k_i}^* \mapsto r_i^*\sigma\}$. Since both sides of are from $\mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$, either **Theory**₁ or **Theory**₂ can be applied to this conjecture. \square

Proof of Lemma 14.18. The statement is proved by induction on $C_{\mathcal{Q}}$. If $C_{\mathcal{Q}}$ is a \mathcal{Q} -context, then

$$\begin{aligned} g(x_1, \dots, x_{j-1}, C_{\mathcal{Q}}[z_1, \dots, z_n], x_{j+1}, \dots, x_m) &\rightarrow_{\mathcal{R}, \mathbb{Z}}^* \\ D[g(x_1, \dots, x_{j-1}, z_{i_1}, x_{j+1}, \dots, x_m), \dots, g(x_1, \dots, x_{j-1}, z_{i_k}, x_{j+1}, \dots, x_m)] \end{aligned}$$

by Definition 14.15 where $z_i \notin \mathcal{V}(D)$ for all $1 \leq i \leq n$. Thus, it only remains to be shown that D is a repeated g -context. But this easily follows since g is LIAC-based and $C_{\mathcal{Q}}$ is a context over $\widehat{\mathcal{C}}(\mathcal{R})$.

If $C_{\mathcal{Q}}$ is a repeated \mathcal{Q} -context of the form $C[C_1, \dots, C_k]$ for repeated \mathcal{Q} -contexts C, C_1, \dots, C_k , then the inductive assumption implies that there exists a repeated g -context D with $z_i \notin \mathcal{V}(D)$ for all $1 \leq i \leq n$ such that

$$\begin{aligned} g(x_1, \dots, x_{j-1}, C_{\mathcal{Q}}[z_1, \dots, z_n], x_{j+1}, \dots, x_m) &= \\ g(x_1, \dots, x_{j-1}, C[C_1, \dots, C_k][z_1, \dots, z_n], x_{j+1}, \dots, x_m) &\rightarrow_{\mathcal{R}, \mathbb{Z}}^* \\ D[g(x_1, \dots, x_{j-1}, u_1, x_{j+1}, \dots, x_m), \dots, g(x_1, \dots, x_{j-1}, u_d, x_{j+1}, \dots, x_m)] \end{aligned}$$

Appendix A. Proofs

with $u_l = C_{e_l}[z_1, \dots, z_n]$ for all $1 \leq l \leq d$. Furthermore, the inductive assumption implies that there exist repeated g -context D_1, \dots, D_d with $z_i \notin \mathcal{V}(D_l)$ for all $1 \leq i \leq n$ and $1 \leq l \leq d$ such that

$$\begin{aligned} g(x_1, \dots, x_{j-1}, u_l, x_{j+1}, \dots, x_m) &\rightarrow_{\mathcal{R}, \mathbb{Z}}^* \\ D_l[g(x_1, \dots, x_{j-1}, z_{l_1}, x_{j+1}, \dots, x_m), \dots, g(x_1, \dots, x_{j-1}, z_{l_{k_l}}, x_{j+1}, \dots, x_m)] \end{aligned}$$

for all $1 \leq l \leq d$, where $l_1, \dots, l_{k_l} \in \{1, \dots, n\}$. Therefore,

$$\begin{aligned} g(x_1, \dots, x_{j-1}, C_{\mathcal{Q}}[z_1, \dots, z_n], x_{j+1}, \dots, x_m) &\rightarrow_{\mathcal{R}, \mathbb{Z}}^* \\ D[g(x_1, \dots, x_{j-1}, u_1, x_{j+1}, \dots, x_m), \dots, g(x_1, \dots, x_{j-1}, u_d, x_{j+1}, \dots, x_m)] &\rightarrow_{\mathcal{R}, \mathbb{Z}}^* \\ D[D_1, \dots, D_d][g(x_1, \dots, x_{j-1}, z_{1_1}, x_{j+1}, \dots, x_m), \dots, g(x_1, \dots, x_{j-1}, z_{d_{k_d}}, x_{j+1}, \dots, x_m)] \end{aligned}$$

for a repeated g -context $C_g := D[D_1, \dots, D_d]$. Furthermore, notice that $\mathcal{V}(C_g) = \mathcal{V}(D[D_1, \dots, D_d]) = \mathcal{V}(D) \cup \mathcal{V}(D_1) \cup \dots \cup \mathcal{V}(D_d)$ does not contain any of the z_i . \square

Proof of Lemma 14.20. Define a sequence of terms by $s_d = f_d(x_{d,1}, \dots, x_{d,m_d})$ and $s_i = f_i(x_{i,1}, \dots, x_{i,j_i-1}, s_{i+1}, x_{i,j_i+1}, \dots, x_{i,m_i})$ for all $1 \leq i \leq d-1$. The lemma is proved by showing the following statement for all $1 \leq i \leq d$:

$$\begin{aligned} s_i \langle C[g_1(r_1^*), \dots, g_n(r_n^*)] \rangle &\rightarrow_{\mathcal{R}, \mathbb{Z}}^* D[s_i \langle g_{j_1}(r_{j_1}^*) \rangle, \dots, s_i \langle g_{j_l}(r_{j_l}^*) \rangle] \text{ for some in-} \\ (\dagger) \text{ dices } j_1, \dots, j_l \in \{1, \dots, n\} \text{ and a repeated } f_i\text{-context } D \text{ with } \mathcal{V}(D) &\subseteq \\ \mathcal{V}(C) \cup \widehat{\mathcal{V}}. \end{aligned}$$

Here, $\widehat{\mathcal{V}} = \{x_{k,j} \mid 1 \leq k \leq d-1 \text{ and } 1 \leq j \leq m_k\}$.

Since $s \langle C[g_1(r_1^*), \dots, g_n(r_n^*)] \rangle = s_1 \langle C[g_1(r_1^*), \dots, g_n(r_n^*)] \rangle \sigma$ for some substitution σ that instantiates at most the $x_{i,j}$ for $i \neq d$ by terms from $\mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$, the statement of the lemma thus follows.

The statement (\dagger) is proved by induction on $d-i$. In the base case, $i = d$ and $s_d \langle C[g_1(r_1^*), \dots, g_n(r_n^*)] \rangle = C[g_1(r_1^*), \dots, g_n(r_n^*)]$ already has the required form.

In the step case, $i < d$ and the inductive assumption for $i+1$ implies

Appendix A. Proofs

$$\begin{aligned}
s_i \langle C[g_1(r_1^*), \dots, g_n(r_n^*)] \rangle &= \\
f_i(y_i^*, s_{i+1} \langle C[g_1(r_1^*), \dots, g_n(r_n^*)] \rangle, z_i^*) &\rightarrow_{\mathcal{R}, \mathbb{Z}}^* \\
f_i(y_i^*, E[s_{i+1} \langle g_{j_1}(r_{j_1}^*) \rangle, \dots, s_{i+1} \langle g_{j_l}(r_{j_l}^*) \rangle], z_i^*) &
\end{aligned}$$

for a repeated f_{i+1} -context E with $\mathcal{V}(E) \subseteq \mathcal{V}(C) \cup \widehat{\mathcal{V}}$. Here, y_i^* contains $x_{i,1}, \dots, x_{i,j_i-1}$ and z_i^* contains $x_{i,j_i+1}, \dots, x_{i,m_i}$. By Lemma 14.18, there exists a repeated f_i -context D such that

$$\begin{aligned}
f_i(y_i^*, E[s_{i+1} \langle g_{j_1}(r_{j_1}^*) \rangle, \dots, s_{i+1} \langle g_{j_l}(r_{j_l}^*) \rangle], z_i^*) &\rightarrow_{\mathcal{R}, \mathbb{Z}}^* \\
D[f_i(y_i^*, s_{i+1} \langle g_{d_1}(r_{d_1}^*) \rangle, z_i), \dots, f_i(y_i^*, s_{i+1} \langle g_{d_e}(r_{d_e}^*) \rangle, z_i^*)] &= \\
D[s_i \langle g(d_1)r_{d_1}^* \rangle, \dots, s_i \langle g_{d_e}(r_{d_e}^*) \rangle] &
\end{aligned}$$

where furthermore $\mathcal{V}(D) \subseteq \mathcal{V}(E) \cup \widehat{\mathcal{V}} \subseteq \mathcal{V}(C) \cup \widehat{\mathcal{V}}$. \square

Proof of Theorem 14.23. Let $D[f(x^*)] \equiv t$ be a simple nested conjecture and consider a rule $f(l^*) \rightarrow C[f(r_1^*), \dots, f(r_m^*)][[\varphi] \in \mathcal{R}(f)$. Provided $f(x^*)$ and $f(l^*)$ are unifiable and $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -satisfiable for $\sigma = \text{mgu}(f(x^*), f(l^*))$, application of **Expand** to $D[f(x^*)] \equiv t$ produces (amongst others) $D[C\sigma[f(r_1^*)\sigma, \dots, f(r_m^*)\sigma] \equiv t\sigma][[\varphi\sigma]$. After application of **Expand**, the set H of hypotheses consists of the rule $D[f(x^*)] \rightarrow t$. By Lemma 14.20,

$$D[C\sigma[f(r_1^*)\sigma, \dots, f(r_m^*)\sigma] \rightarrow_{\mathcal{R}, \mathbb{Z}}^* E[D[f(r_{d_1}^*)\sigma], \dots, D[f(r_{d_e}^*)\sigma]]$$

for some context E over $\widehat{\mathcal{C}}(\mathcal{R})$.

Now, if $x_i = x_j$ for $i < j$, then $\langle i, j \rangle \in \text{ImpEq}(f)$. Since $x_i = x_j$ implies $l_i\sigma = l_j\sigma$, the definition of ImpEq yields $r_{d_k,i}\sigma = r_{d_k,j}\sigma$ for all $1 \leq k \leq e$ as in the proof of Theorem 14.6. Hence, **Simplify** applies m times to the conjecture using the hypothesis $D[f(x^*)] \rightarrow t \in H$ and $E[t\tau_1, \dots, t\tau_e] \equiv t\sigma[[\varphi\sigma]$ is obtained, where $\tau_k = \{x^* \mapsto r_{d_k}^*\sigma\}$. Since both sides are from $\mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$, either **Theory**₁ or **Theory**₂ is applicable. \square

Proof of Theorem 14.27. Adapt the proof of Theorem 14.23 in the same way the proof of Theorem 14.6 was adapted to obtain the proof of Theorem 14.12. \square

Appendix A. Proofs

Proof of Theorem 14.30. Identical to the proof of Theorem 13.40, but the proofs of Lemma 13.37 and Lemma 13.38 have to be adapted as follows.

In Lemma 13.37, in order to show $\leftrightarrow_{\mathcal{R} \cup E_n \cup H_n, \mathbb{Z}}^* \subseteq \leftrightarrow_{\mathcal{R} \cup E_{n+1} \cup H_{n+1}, \mathbb{Z}}^*$ if the inference rule Theory'_1 was applied, assume that $v \leftrightarrow_{\{s \doteq t[\varphi]\}, \mathbb{Z}} w$ for \mathbb{Z} -normal ground terms v and w . Thus, there exists a ground substitution σ such that $v = C[s\sigma]$, $w = C[t\sigma]$ (or $w = C[s\sigma]$, $v = C[t\sigma]$), and $\varphi\sigma$ is $\text{Th}_{\mathbb{Z}}$ -valid. Since $\varphi' \Rightarrow s' \simeq t'$ is LIAC-valid, $s' \doteq t'[\varphi']$ is an inductive theorem of \mathcal{R} . Thus, the conjecture $s \doteq t[\varphi]$ is an inductive theorem of \mathcal{R} as well. Since $\varphi\sigma$ is $\text{Th}_{\mathbb{Z}}$ -valid, this implies $s\sigma \leftrightarrow_{\mathcal{R}, \mathbb{Z}}^* t\sigma$ and thus $v \leftrightarrow_{\mathcal{R}, \mathbb{Z}}^* w$.

For the proof of Lemma 13.38, it suffices to consider the case where the inference rule Theory'_1 was applied. Then, $E_n = E \uplus \{s \doteq t[\varphi]\}$, $E_{n+1} = E$, and $H_{n+1} = H_n$, where $\varphi' \Rightarrow s' \simeq t'$ is LIAC-valid for a safe generalization $s' \doteq t'[\varphi']$ of $s \doteq t[\varphi]$. As before, if $v \leftrightarrow_{E, \mathbb{Z}} w$, then the claim follows from the inductive assumption. Otherwise, $v = C[s\sigma]$ and $w = C[t\sigma]$ (or $v = C[t\sigma]$ and $w = C[s\sigma]$) where $\varphi\sigma$ is $\text{Th}_{\mathbb{Z}}$ -valid. As above, the conjecture $s \doteq t[\varphi]$ is an inductive theorem of \mathcal{R} since $s' \doteq t'[\varphi']$ is an inductive theorem of \mathcal{R} . Thus, $s\sigma \leftrightarrow_{\mathcal{R}, \mathbb{Z}}^* t\sigma$, which implies $s\sigma \rightarrow_{\mathcal{R}, \mathbb{Z}}^* \leftarrow_{\mathcal{R}, \mathbb{Z}}^* t\sigma$ since $\rightarrow_{\mathcal{R}, \mathbb{Z}}$ is confluent and thus Church-Rosser. Therefore, $v \rightarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* \leftarrow_{\mathcal{R} \cup H, \mathbb{Z}}^* w$ is immediate. \square

Proof of Theorem 14.31. Let $\langle E, \emptyset \rangle \vdash_{\mathcal{I}'}^* \langle E_n, H_n \rangle \vdash_{\mathcal{I}'} \perp$ where $E_n = E'_n \uplus \{s \doteq t[\varphi]\}$ such that $\varphi' \Rightarrow s' \simeq t'$ is not LIAC-valid for a safe generalization $s' \doteq t'[\varphi']$ of $s \doteq t[\varphi]$. Therefore, there exists a \mathbb{Z} -normal constructor ground substitution σ' such that $\varphi'\sigma'$ is $\text{Th}_{\mathbb{Z}}$ -valid and $s'\sigma' \simeq t'\sigma'$ is not LIAC-valid, which implies $s'\sigma' \neq t'\sigma'$ since σ' is \mathbb{Z} -normal. Assume that $s' \doteq t'[\varphi']$ is an inductive theorem of \mathcal{R} . Then, $s'\sigma' \leftrightarrow_{\mathcal{R}, \mathbb{Z}}^* t'\sigma'$ which gives $s'\sigma' \rightarrow_{\mathcal{R}, \mathbb{Z}}^* \leftarrow_{\mathcal{R}, \mathbb{Z}}^* t'\sigma'$ because $\rightarrow_{\mathcal{R}, \mathbb{Z}}$ is confluent and thus Church-Rosser. But $s'\sigma' \rightarrow_{\mathcal{R}, \mathbb{Z}}^* \leftarrow_{\mathcal{R}, \mathbb{Z}}^* t'\sigma'$ is not possible since $s'\sigma' \neq t'\sigma'$ and both terms are irreducible by $\rightarrow_{\mathcal{R}, \mathbb{Z}}$ since s' and t' do not contain defined symbols and σ' is a constructor ground substitution.

Appendix A. Proofs

Thus, $s' \equiv t'[\varphi']$ is not an inductive theorem of \mathcal{R} , which implies that $s \equiv t[\varphi]$ is not an inductive theorem of \mathcal{R} , either. Thus, there exists a \mathbb{Z} -normal constructor ground substitution σ such that $\varphi\sigma$ is $\mathcal{Th}_{\mathbb{Z}}$ -valid but $s\sigma \not\leftrightarrow_{\mathcal{R}, \mathbb{Z}}^* t\sigma$. Since $s\sigma \leftrightarrow_{E_n, \mathbb{Z}} t\sigma$, Lemma 13.37 implies $s\sigma \leftrightarrow_{\mathcal{R} \cup E, \mathbb{Z}}^* t\sigma$. If all atomic conjectures in E are inductive theorems, then this would imply $s\sigma \leftrightarrow_{\mathcal{R}, \mathbb{Z}}^* t\sigma$, contradicting $s\sigma \not\leftrightarrow_{\mathcal{R}, \mathbb{Z}}^* t\sigma$. \square

Proof of Theorem 14.33. “ \Leftarrow ” is trivial since \mathcal{R} is quasi-reductive and terminating. For “ \Rightarrow ”, assume that $C[x_{t_1}, \dots, x_{t_n}] \simeq D[x_{s_1}, \dots, x_{s_m}]$ is not LIAC-valid. Then, it needs to be shown that this contradicts that

$$C[t_1, \dots, t_n] \equiv D[s_1, \dots, s_m] \text{ is an inductive theorem of } \mathcal{R} \quad (\dagger)$$

For this, it is first shown that (\dagger) implies $\{t_1, \dots, t_n\} = \{s_1, \dots, s_m\}$. Otherwise, without loss of generality, let $s_i \notin \{t_1, \dots, t_n\}$ and let σ be a \mathbb{Z} -free constructor ground substitution for the variables in $t_1, \dots, t_n, s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_m$. Since \mathcal{R} is quasi-reductive and the t_j, s_k are variable-disjoint, there exist a $q \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$ with $C[t_1, \dots, t_n]\sigma \rightarrow_{\mathcal{R}, \mathbb{Z}}^* q$ and a context D' over $\widehat{\mathcal{C}}(\mathcal{R})$ such that $D[s_1, \dots, s_m]\sigma \rightarrow_{\mathcal{R}, \mathbb{Z}}^* D'[s_i, \dots, s_i]$. Now, (\dagger) implies that $q \equiv D'[s_i, \dots, s_i]$ is an inductive theorem of \mathcal{R} . This implies $q = D'[q_i, \dots, q_i]$ such that $s_i \equiv q_i$ is an inductive theorem. But this contradicts the assumption that s_i satisfies the no-theory condition.

Next, perform an induction on the contexts C and D . If $C = \square$, then the conjecture in (\dagger) has the form $t = D[t, \dots, t]$ where D contains at least one occurrence of \square . If $D = \square$, then $C[x_t] \simeq D[x_t]$ is trivially LIAC-valid, contradicting the assumption. If $D \neq \square$, then (\dagger) contradicts the fact that no term with a sort different from `int` is equal to one of its proper subterms in the theory LIAC.

Similarly, $D = \square$ results in contradictions. Thus, the case where $C \neq \square$ and $D \neq \square$ remains, i.e., $C = c(C_1, \dots, C_e)$ for some constructor c . Then, the conjecture in (\dagger) has the form $c(C_1, \dots, C_e)[t_1, \dots, t_n] \equiv D[s_1, \dots, s_m]$. Inductive validity of this conjecture implies that $D = c(D_1, \dots, D_e)$ and the conjectures $C_i[t_1, \dots, t_n] \equiv$

Appendix A. Proofs

$D_i[s_1, \dots, s_m]$ are inductive theorems for all $1 \leq i \leq e$. The desired contradiction then follows from the inductive assumption. \square

Proof of Theorem 14.35. Let $s^*, t, q \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$ such that s^* and t do not contain any of the variables x^* in q . Let $q[s^*]$ denote the term obtained from q by replacing x_i by s_i for all i .

First, it is shown by induction on t that $q[s^*] = t$ iff $q \in Q_{s^*}(t)$. If $q[s^*] = x$ for a variable $x \notin \mathcal{V}(q)$, then $q = x_i$ and $s_i = x$ for some i . Hence, $q \in Q_{s^*}(t)$. If $q[s^*] = c(t_1, \dots, t_k)$, then there are two possibilities. If q is a variable x_i , then $s_i = c(t_1, \dots, t_k)$ and thus $q \in Q_{s^*}(t)$. Otherwise, $q = c(q_1, \dots, q_k)$ where $q_i[s^*] = t_i$ for all i . By the inductive assumption, $q_i \in Q_{s^*}(t_i)$ and thus $q \in Q_{s^*}(c(t_1, \dots, t_k))$.

Now, the statement of the lemma can be shown. For this, assume that f does not satisfy the no-theory condition. Thus, $f(x^*) \equiv q$ is an inductive theorem for some $q \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$. In particular, for the non-recursive rule $f(s^*) \rightarrow r \in \mathcal{R}(f)$ that was chosen in the construction of $Q(f)$, $f(s^*) \equiv q[s^*]$ is an inductive theorem. Thus, $q[s^*] = r$ since both $q[s^*]$ and r do not contain defined symbols and \mathcal{R} is confluent and thus Church-Rosser. By the property shown above, $q \in Q_{s^*}(r)$. It now needs to be shown that $\widehat{l} := l \downarrow_{f(x^*) \rightarrow q} = r \downarrow_{f(x^*) \rightarrow q} =: \widehat{r}$ for all $l \rightarrow r \in \mathcal{R}(f)$. Since $l \equiv r$ is an inductive theorem of \mathcal{R} and $f(x^*) \equiv q$ is an inductive theorem by the assumption, $\widehat{l} \equiv \widehat{r}$ is an inductive theorem as well. Since \mathcal{R} is Church-Rosser and $\widehat{l}, \widehat{r} \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$, it follows that $\widehat{l} = \widehat{r}$. \square

Proof of Theorem 14.40. Let \mathcal{R}' be the unrolling of f_i . It is first shown that $f_i(s^*) \downarrow_{\mathcal{R}, \mathbb{Z}} = f_i(s^*) \downarrow_{\mathcal{R}', \mathbb{Z}}$ for all \mathbb{Z} -normal constructor ground terms s^* . This is shown by induction on the number of reduction steps in $f_i(s^*) \rightarrow_{\mathcal{R}, \mathbb{Z}}^* q = f_i(s^*) \downarrow_{\mathcal{R}, \mathbb{Z}}$. Since \mathcal{R} is quasi-reductive, $q \in \mathcal{T}(\mathcal{C}(\mathcal{R}) \cup \mathcal{F}_{Th_{\mathbb{Z}}})$ and at least one reduction step is needed. If exactly one reduction step is needed, then the used rewrite rule is non-recursive and thus also contained in \mathcal{R}' . Otherwise, $f_i(s^*) = f_i(l^*)\sigma$ for some $f_i(l^*) \rightarrow$

Appendix A. Proofs

$C[f_{1-i}(x_1^*), \dots, f_{1-i}(x_n^*)] \in \mathcal{R}(f_i)$. After application of this rewrite rule, the term $C\sigma[f_{1-i}(x_1^*)\sigma, \dots, f_{1-i}(x_n^*)\sigma]$ is obtained and applying $f_{1-i}(l_1^*) \rightarrow r_1, \dots, f_{1-i}(l_n^*) \rightarrow r_n$ using substitutions μ_j (i.e., $f_{1-i}(x_j^*)\sigma = f_{1-i}(l_j^*)\mu_j$) gives $C\sigma[r_1\mu_1, \dots, r_n\mu_n]$. It now suffices to show that $f_i(s^*) \rightarrow_{\mathcal{R}, \mathbb{Z}} C\sigma[r_1\mu_1, \dots, r_n\mu_n]$ since the claim then follows from the inductive assumption. For each $1 \leq j \leq n$, let $\tau_j = \{x_j^* \mapsto l_j^*\}$ and $\tau = \tau_1 \cup \dots \cup \tau_n$. Next, extend τ to behave like σ on all variables not occurring in x_1^*, \dots, x_n^* . Without loss of generality, the rules $f_{1-i}(l_1^*) \rightarrow r_1, \dots, f_{1-i}(l_n^*) \rightarrow r_n$ are variable-disjoint and contain only fresh variables, which implies that $\mu = \mu_1 \cup \dots \cup \mu_n$ is well-defined. Then, $\sigma = \tau\mu$ and $f_i(s^*) = f_i(l^*)\sigma = f_i(l^*)\tau\mu \rightarrow_{\mathcal{R}, \mathbb{Z}} C\tau\mu[r_1\mu, \dots, r_n\mu] = C\sigma[r_1\mu_1, \dots, r_n\mu_n]$.

Now, the statement of the theorem can be proved. For this, assume that f_i does not satisfy the no-theory condition w.r.t. \mathcal{R} . Then, $f_i(x_1, \dots, x_n) \equiv q$ is an inductive theorem of \mathcal{R} for some $q \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$. Thus, $f_i(x_1, \dots, x_n)\sigma \leftrightarrow_{\mathcal{R}, \mathbb{Z}}^* q\sigma$ for all \mathbb{Z} -normal constructor ground substitutions. Since \mathcal{R} is confluent and thus Church-Rosser and $q\sigma$ is irreducible by $\rightarrow_{\mathcal{R}, \mathbb{Z}}$, this implies $f_i(x_1, \dots, x_n)\sigma \downarrow_{\mathcal{R}, \mathbb{Z}} = q\sigma$. By the above, $f_i(x_1, \dots, x_n)\sigma \downarrow_{\mathcal{R}', \mathbb{Z}} = q\sigma$. Since this is true for all \mathbb{Z} -normal constructor ground substitutions, $f_i(x_1, \dots, x_n) \equiv q$ is an inductive theorem of \mathcal{R}' , i.e., f_i does not satisfy the no-theory condition w.r.t. \mathcal{R}' . \square

Proof of Theorem 14.42. Recall from the proof of Theorem 14.35 that $q[s^*] = t$ iff $q \in Q_{s^*}(t)$ for $s^*, t, q \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$ such that s^* and t do not contain any of the variables x^* in q . Assume that f does not satisfy the no-theory condition. Thus, $f(x^*) \equiv q$ is an inductive theorem for some $q \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$. In particular, for the non-recursive rule $f(s^*) \rightarrow r \in \mathcal{R}(f)$ that was chosen in the construction of $Q(f)$, $f(s^*) \equiv q[s^*]$ is an inductive theorem. Thus, $q[s^*] = r$ since both $q[s^*]$ and r do not contain defined symbols and \mathcal{R} is confluent and thus Church-Rosser. By the property from above, $q \in Q_{s^*}(r)$.

It now needs to be shown that $l \downarrow_{f(x^*) \rightarrow q} \simeq \text{CAP}_{\mathcal{D}}(r \downarrow_{f(x^*) \rightarrow q})$ is LIAC-satisfiable for

Appendix A. Proofs

all $l \rightarrow r \in \mathcal{R}(f)$. For this, it suffices to show that whenever $s \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$ and $s \equiv t$ is an inductive theorem, then $s = \text{CAP}_{\mathcal{D}}(t)\sigma$ for some constructor substitution σ that only instantiates the variables introduced by $\text{CAP}_{\mathcal{D}}$. The claim then follows since $l \downarrow_{f(x^*) \rightarrow q} \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$ and $l \downarrow_{f(x^*) \rightarrow q} \equiv r \downarrow_{f(x^*) \rightarrow q}$ is an inductive theorem since both $l \equiv r$ and $f(x^*) \equiv q$ are inductive theorems by the assumption. The property is shown by induction on t . If t is a variable, then $s = t = \text{CAP}_{\mathcal{D}}(t)$ since each sort has at least two distinct constructor ground terms. If $\text{root}(t)$ is a defined symbol, then $\text{CAP}_{\mathcal{D}}(t)$ is a fresh variable z , which implies $s = \text{CAP}_{\mathcal{D}}(t)\sigma$ for $\sigma = \{z \mapsto s\}$. If $t = c(t_1, \dots, t_n)$ for some $c \in \widehat{\mathcal{C}}(\mathcal{R})$, then $s = c(s_1, \dots, s_n)$ such that $s_i \equiv t_i$ is an inductive theorem for all $1 \leq i \leq n$ since \mathcal{R} is Church-Rosser and $s \in \mathcal{T}(\widehat{\mathcal{C}}(\mathcal{R}), \mathcal{V})$. Now $\text{CAP}_{\mathcal{D}}(t) = c(\text{CAP}_{\mathcal{D}}(t_1), \dots, \text{CAP}_{\mathcal{D}}(t_n))$ and the inductive assumption implies $s_i = \text{CAP}_{\mathcal{D}}(t_i)\sigma_i$ for some σ_i that only instantiates the variables introduced by $\text{CAP}_{\mathcal{D}}$. Since the variables introduced by $\text{CAP}_{\mathcal{D}}(t_i)$ and $\text{CAP}_{\mathcal{D}}(t_j)$ are disjoint whenever $i \neq j$, the substitution $\sigma := \sigma_1 \cup \dots \cup \sigma_n$ satisfies $s = \text{CAP}_{\mathcal{D}}(t)\sigma$. \square

Proof of Theorem 14.50. Let $D[f(x^*)] \equiv E[g(y^*)]$ be a complex conjecture and consider a rule $f(l^*) \rightarrow C[f(r_1^*), \dots, f(r_m^*)] \in \mathcal{R}(f)$. Provided $f(x^*)$ and $f(l^*)$ are unifiable with $\sigma = \text{mgu}(f(x^*), f(l^*))$, application of **Expand** to $D[f(x^*)] \equiv E[g(y^*)]$ produces (amongst others) $D[C\sigma[f(r_1^*)\sigma, \dots, f(r_m^*)\sigma]] \equiv E[g(y^*)\sigma]$. After application of **Expand**, the set H of hypotheses consists of the rule $D[f(x^*)] \rightarrow E[g(y^*)]$. By Lemma 14.20,

$$D[C\sigma[f(r_1^*)\sigma, \dots, f(r_m^*)\sigma]] \rightarrow_{\mathcal{R}, \mathbb{Z}}^* D'[D[f(r_{d_1}^*)\sigma], \dots, D[f(r_{d_k}^*)\sigma]]$$

for some context D' over $\widehat{\mathcal{C}}(\mathcal{R})$. Due to the condition on the definition schemes $\text{Def}(f(x^*))$ and $\text{Def}(g(y^*))$, there exists a rule $g(s^*) \rightarrow C'[g_1(t_1^*), \dots, g_{m'}(t_{m'}^*)] \in \mathcal{R}(g)$ such that $g(y^*\sigma) = g(s^*)$. Applying this rule to $E[g(y^*)\sigma]$ yields the term $E[C'\sigma[g_1(t_1^*)\sigma, \dots, g_{m'}(t_{m'}^*)\sigma]]$ and Lemma 14.20 implies

$$E[C'\sigma[g_1(t_1^*)\sigma, \dots, g_{m'}(t_{m'}^*)\sigma]] \rightarrow_{\mathcal{R}, \mathbb{Z}}^* D''[E[g_{e_1}(t_{e_1}^*)\sigma], \dots, E[g_{e_{k'}}(t_{e_{k'}}^*)\sigma]]$$

for some context D'' over $\widehat{\mathcal{C}}(\mathcal{R})$.

Appendix A. Proofs

As before, the *ImpEq*-condition implies that *Simplify* applies m times using the hypothesis $D[f(x^*)] \rightarrow E[g(y^*)] \in H$ and

$$D'[E[g(y^*)\tau_1], \dots, E[g(y^*)\tau_k]] \equiv D''[E[g_{e_1}(t_{e_1}^*)\sigma], \dots, E[g_{e_{k'}}(t_{e_{k'}})\sigma]]$$

is obtained where $\tau_j = \{x^* \mapsto r_{d_j}^*\sigma\}$. Since $\tau_j \in \mathcal{Call}(f(x^*), l \rightarrow r, \sigma)$, the final condition in Definition 14.48 implies that either **Theory'**₁ or **Theory'**₂ is applicable after normalization w.r.t. \mathcal{R} . □

Proof of Theorem 14.57. Adapt the proof of Theorem 14.27 in the same way the proof of Theorem 14.23 was adapted to obtain the proof of Theorem 14.50. □

Appendix B

Evaluation

B.1 Termination

AProVE-CERS is the implementation of the methods developed in this dissertation, while AProVE-Integer is based in [75, 143]. An “N/A” in the column for AProVE-Integer denotes that the methods from [75, 143] are not applicable to that example. The examples themselves and all proofs generated by AProVE-CERS and AProVE-Integer are available at <http://www.cs.unm.edu/~spf/tdps/>.

| File | AProVE-CERS | | AProVE-Integer | |
|----------|------------------|--------|------------------|--------|
| | Total time (sec) | Result | Total time (sec) | Result |
| 01.patrs | 0.19 | YES | 0.54 | YES |
| 02.patrs | 0.04 | YES | 0.25 | YES |
| 03.patrs | 0.04 | YES | 0.31 | YES |
| 04.patrs | 0.01 | YES | 0.17 | YES |
| 08.patrs | 0.04 | YES | 0.17 | YES |
| 09.patrs | 0.04 | YES | 0.30 | YES |
| 13.patrs | 0.07 | YES | 1.69 | YES |
| 15.patrs | 0.12 | YES | 2.20 | YES |
| 16.patrs | 0.11 | YES | 3.05 | YES |
| 17.patrs | 0.17 | YES | 2.40 | YES |
| 18.patrs | 0.19 | YES | 4.61 | YES |
| 19.patrs | 0.19 | YES | 42.02 | YES |
| 20.patrs | 0.08 | YES | 1.11 | YES |

Appendix B. Evaluation

| File | AProVE-CERS | | AProVE-Integer | |
|-----------------|------------------|---------|------------------|---------|
| | Total time (sec) | Result | Total time (sec) | Result |
| 21.patrs | 0.11 | YES | 2.57 | YES |
| 22.patrs | 0.11 | YES | 2.20 | YES |
| 23.patrs | 0.11 | YES | 2.12 | YES |
| 24.patrs | 0.17 | YES | 6.52 | YES |
| 5.3.patrs | 0.11 | YES | N/A | |
| 5.4.patrs | 0.18 | YES | N/A | |
| 5.5.patrs | 0.31 | YES | N/A | |
| A01.patrs | 0.04 | YES | 0.31 | YES |
| A02.patrs | 0.03 | YES | 0.28 | YES |
| A03.patrs | 0.05 | YES | 2.69 | YES |
| A06.patrs | 0.06 | YES | 5.64 | YES |
| A07.patrs | 0.11 | YES | 1.19 | YES |
| A08.patrs | 0.08 | YES | 4.44 | YES |
| A11.patrs | 0.03 | YES | 0.38 | YES |
| A12.patrs | 0.03 | YES | 0.36 | YES |
| A13.patrs | 0.07 | YES | 0.67 | YES |
| A14.patrs | 0.16 | YES | 22.69 | YES |
| a.01.patrs | 0.11 | YES | 1.36 | YES |
| a.03.patrs | 0.94 | YES | 60.02 | TIMEOUT |
| a.04.patrs | 0.04 | YES | 0.39 | YES |
| a.05.patrs | 0.04 | YES | 0.32 | YES |
| a.06.patrs | 0.04 | YES | 0.52 | YES |
| a.07.patrs | 0.03 | YES | 0.40 | YES |
| a.08.patrs | 0.03 | YES | 0.32 | YES |
| a.09.patrs | 0.03 | YES | 0.39 | YES |
| a.10.patrs | 0.10 | YES | 52.72 | YES |
| a.11.patrs | 0.16 | YES | 6.75 | YES |
| c.01.patrs | 0.10 | YES | 1.83 | YES |
| c.02.patrs | 0.11 | YES | 1.46 | YES |
| c.03.patrs | 0.10 | YES | 2.60 | YES |
| c.04.patrs | 0.07 | YES | 2.51 | YES |
| c.05.patrs | 0.09 | YES | 7.38 | YES |
| choice.patrs | 60.00 | TIMEOUT | 60.04 | TIMEOUT |
| complete2.patrs | 60.00 | TIMEOUT | 10.22 | YES |
| complete3.patrs | 0.47 | YES | 1.99 | YES |
| countdown.patrs | 0.16 | YES | 0.26 | YES |
| csharp1.patrs | 0.09 | YES | 0.46 | YES |
| csharp2.patrs | 0.07 | YES | 0.49 | YES |
| csharp3.patrs | 0.06 | YES | 0.39 | YES |

Appendix B. Evaluation

| File | AProVE-CERS | | AProVE-Integer | |
|--------------------------------|------------------|---------|------------------|---------|
| | Total time (sec) | Result | Total time (sec) | Result |
| divMinus.patrs | 0.04 | YES | 0.34 | YES |
| div.patrs | 0.04 | YES | 0.35 | YES |
| eratosthenes.patrs | 0.44 | YES | 10.32 | YES |
| eratosthenes_small.patrs | 0.2 | YES | 26.45 | YES |
| gcd_minmax.patrs | 0.05 | YES | 1.45 | YES |
| gcd.patrs | 0.07 | YES | N/A | |
| horner.patrs | 0.07 | YES | N/A | |
| increase1.patrs | 0.02 | YES | 0.39 | YES |
| increase2.patrs | 0.05 | YES | 2.85 | YES |
| increase3.patrs | 0.08 | YES | 2.70 | YES |
| increase4.patrs | 0.03 | YES | 0.40 | YES |
| indirect.patrs | 0.04 | YES | 0.68 | YES |
| mergesort_multiset.patrs | 0.14 | YES | N/A | |
| mergesort_set.patrs | 0.14 | YES | N/A | |
| minsort_multiset.patrs | 0.08 | YES | N/A | |
| minsort_set.patrs | 0.08 | YES | N/A | |
| multiset_set.patrs | 0.17 | YES | N/A | |
| mult.patrs | 0.06 | YES | 2.63 | YES |
| nat-list-max.patrs | 0.11 | YES | N/A | |
| nat-mset-min.patrs | 0.09 | YES | N/A | |
| operations_multiset.patrs | 0.48 | YES | N/A | |
| operations_set.patrs | 0.41 | YES | N/A | |
| pathological.patrs | 0.1 | YES | N/A | |
| poly2.patrs | 60.00 | TIMEOUT | 60.00 | TIMEOUT |
| poly4.patrs | 0.32 | YES | 60.03 | TIMEOUT |
| practical1.patrs | 0.13 | YES | 2.47 | YES |
| practical2.patrs | 60.00 | TIMEOUT | 60.03 | TIMEOUT |
| practical3.patrs | 60.11 | TIMEOUT | 60.08 | TIMEOUT |
| quicksort_ins_multiset.patrs | 0.20 | YES | N/A | |
| quicksort_ins_set.patrs | 0.20 | YES | N/A | |
| quicksort_ugly_multiset.patrs | 0.25 | YES | N/A | |
| quicksort_ugly_set.patrs | 0.24 | YES | N/A | |
| quicksort_union_multiset.patrs | 0.25 | YES | N/A | |
| quicksort_union_set.patrs | 0.25 | YES | N/A | |
| random_full_no_wrap.patrs | 0.23 | YES | 42.93 | YES |
| random_full.patrs | 0.09 | YES | 43.40 | YES |
| randomFullUpDown.patrs | 0.37 | YES | 60.01 | TIMEOUT |
| random_no_wrap.patrs | 0.11 | YES | 0.36 | YES |
| random.patrs | 0.04 | YES | 0.70 | YES |

Appendix B. Evaluation

| File | AProVE-CERS | | AProVE-Integer | |
|---------------------|------------------|--------|------------------|---------|
| | Total time (sec) | Result | Total time (sec) | Result |
| removal.patrs | 0.05 | YES | N/A | |
| round.patrs | 0.18 | MAYBE | 0.50 | YES |
| sequents.patrs | 0.73 | MAYBE | N/A | |
| size_multiset.patrs | 0.07 | YES | N/A | |
| size_set.patrs | 0.06 | YES | N/A | |
| sqrt.patrs | 0.05 | YES | 0.56 | YES |
| sumLog.patrs | 0.30 | MAYBE | 60.02 | TIMEOUT |
| sum_multiset.patrs | 0.10 | YES | N/A | |
| sumto_no_if.patrs | 0.03 | YES | 0.49 | YES |
| sumto.patrs | 0.04 | YES | 0.50 | YES |
| sumUp.patrs | 0.03 | YES | 0.50 | YES |
| terminate.patrs | 0.22 | YES | 0.87 | YES |
| test1.patrs | 0.08 | YES | 2.20 | YES |
| test2.patrs | 0.09 | YES | N/A | |
| test3.patrs | 0.01 | YES | N/A | |
| test4.patrs | 0.39 | YES | 0.56 | YES |
| test5.patrs | 0.00 | YES | N/A | |
| test6.patrs | 0.00 | YES | N/A | |
| unsatCond1.patrs | 0.00 | YES | 0.26 | YES |

B.2 Context-Sensitive Termination

The following table contains the results of AProVE-CERS. Notice that [75, 143] does not consider context-sensitive rewriting, i.e., AProVE-Integer is not applicable to these examples. The examples themselves and all proofs generated by AProVE-CERS are available at <http://www.cs.unm.edu/~spf/tdps/>.

| File | Total time (sec) | Result |
|----------------|------------------|--------|
| add.x.patrs | 0.27 | YES |
| after.patrs | 0.23 | YES |
| all-op.patrs | 0.32 | YES |
| diff1.patrs | 0.24 | YES |
| diff2.patrs | 0.23 | YES |
| even-odd.patrs | 0.14 | YES |
| fib.patrs | 0.28 | YES |
| first.patrs | 0.33 | YES |
| from-nth.patrs | 0.26 | YES |

Appendix B. Evaluation

| File | Total time (sec) | Result |
|---------------|------------------|--------|
| from.patrs | 0.12 | YES |
| head.patrs | 0.29 | YES |
| indx.patrs | 0.48 | YES |
| length.patrs | 0.23 | YES |
| misc.patrs | 0.15 | YES |
| pi.patrs | 0.18 | YES |
| prefix.patrs | 0.44 | YES |
| primes.patrs | 0.19 | YES |
| quot.patrs | 0.41 | YES |
| second.patrs | 0.12 | YES |
| sel.patrs | 0.25 | YES |
| silly01.patrs | 0.47 | MAYBE |
| silly02.patrs | 0.13 | YES |
| silly03.patrs | 0.12 | YES |
| silly04.patrs | 0.13 | YES |
| silly05.patrs | 0.12 | YES |
| silly06.patrs | 0.15 | YES |
| silly07.patrs | 0.16 | YES |
| silly08.patrs | 0.12 | YES |
| silly09.patrs | 0.12 | YES |
| silly10.patrs | 0.13 | YES |
| silly11.patrs | 0.18 | YES |
| silly12.patrs | 0.14 | YES |
| silly13.patrs | 0.35 | YES |
| silly14.patrs | 0.22 | YES |
| silly15.patrs | 0.12 | YES |
| silly16.patrs | 0.40 | MAYBE |
| silly17.patrs | 0.16 | YES |
| tail.patrs | 0.12 | YES |
| take.patrs | 0.18 | YES |
| terms.patrs | 0.19 | YES |

B.3 Induction

The following table contains the results of *Sail2*. The examples themselves and all (dis-)proofs generated by *Sail2* are available at <http://www.cs.unm.edu/~spf/sail2/>.

Appendix B. Evaluation

| Conjecture | Checking (msec) | (Dis-)Proving (msec) | (Dis-)Proving w/o Time in CVC3 (msec) | Termination (msec) |
|---|--------------------|-------------------------|--|-----------------------|
| $\text{oddlis}(\text{alternate}(xs, xs)) \equiv xs$ | 0.087 | 12.508 | 0.164 | 226.461 |
| $\text{evenlis}(\text{alternate}(xs, xs)) \equiv xs$ | 0.080 | 12.779 | 0.163 | 254.837 |
| $\text{oddlis}(\text{alternate}(xs, ys)) \equiv xs$ | 0.080 | 12.537 | 0.172 | 226.667 |
| $\text{evenlis}(\text{alternate}(xs, ys)) \equiv ys$ | 0.078 | 12.552 | 0.182 | 140.106 |
| $\text{app}(xs, ys) \equiv ys$ | 0.032 | 12.341 | 0.126 | 36.791 |
| $\text{half}(\text{double}(x)) \equiv x$ | 0.080 | 11.760 | 0.139 | 136.888 |
| $\text{even}(\text{double}(x)) \equiv \text{true}$ | 0.064 | 11.510 | 0.127 | 87.391 |
| $\text{not}(\text{even}(x)) \equiv \text{odd}(x)$ $\text{not}(\text{odd}(x)) \equiv \text{even}(x)$ | 0.186 | 13.168 | 0.211 | 188.983 |
| $\text{gtr}(x, x) \equiv \text{false}$ | 0.059 | 11.554 | 0.133 | 96.108 |
| $\text{geq}(x, x) \equiv \text{true}$ | 0.055 | 11.611 | 0.134 | 95.516 |
| $\text{not}(\text{gtr}(x, x)) \equiv \text{true}$ | 0.087 | 11.561 | 0.185 | 93.591 |
| $\text{not}(\text{geq}(x, x)) \equiv \text{false}$ | 0.088 | 11.574 | 0.144 | 94.273 |
| $\text{maxlis}(xs, xs) \equiv xs$ | 0.055 | 32.083 | 0.209 | 106.579 |
| $\text{maxlis}(xs, xs) \equiv xs$ $\text{max}(x, x) \equiv x$ | 0.074 | 12.029 | 0.186 | 94.276 |
| $\text{max}(x, x) \equiv x$ | 0.054 | 11.538 | 0.118 | 95.318 |
| $\text{min}(x, x) \equiv x$ | 0.052 | 11.574 | 0.127 | 85.469 |
| $\text{min}(x, y) \equiv \text{max}(x, y)$ | 0.199 | 14.365 | 0.219 | 165.281 |
| $\text{minus}(x, x) \equiv \mathcal{O}$ | 0.044 | 11.488 | 0.116 | 34.796 |
| $\text{mix}(xs, xs) \equiv xs$ $\text{mix}'(xs, xs) \equiv xs$ | 0.063 | 11.907 | 0.168 | 55.986 |
| $\text{plus}(x, y) \equiv x$ | 0.033 | 13.075 | 0.125 | 32.788 |
| $\text{prefix}(xs, xs) \equiv xs$ | 0.058 | 40.278 | 0.259 | 274.004 |
| $\text{ptwise}(xs, xs) \equiv \text{true}$ | 0.058 | 32.077 | 0.213 | 67.236 |
| $\text{fst}(\text{stitch}(xs, xs)) \equiv xs$ $\text{fst}(\text{stitch}'(xs, xs)) \equiv xs$ | 0.122 | 12.414 | 0.238 | 124.539 |
| $\text{fst}(\text{stitch}(xs, ys)) \equiv xs$ $\text{fst}(\text{stitch}'(xs, ys)) \equiv xs$ | 0.122 | 16.685 | 0.251 | 115.536 |
| $\text{plen}(\text{stitch}(xs, xs)) = \text{len}(xs)$ $\text{plen}(\text{stitch}'(xs, xs)) = \text{len}(xs)$ | 0.295 | 15.916 | 0.322 | 271.960 |
| $\text{fst}(\text{zip}(xs, xs)) \equiv xs$ | 0.083 | 12.081 | 0.176 | 84.801 |
| $\text{fst}(\text{zip}(xs, ys)) \equiv xs$ | 0.079 | 12.782 | 0.182 | 157.786 |
| $\text{plen}(\text{zip}(xs, xs)) \equiv \text{len}(xs)$ | 0.197 | 13.712 | 0.244 | 123.819 |

Bibliography

- [1] Beatriz Alarcón, Fabian Emmes, Carsten Fuhs, Jürgen Giesl, Raúl Gutiérrez, Salvador Lucas, Peter Schneider-Kamp, and René Thiemann. Improving context-sensitive dependency pairs. In Cervesato et al. [41], pages 636–651.
- [2] Beatriz Alarcón, Raúl Gutiérrez, and Salvador Lucas. Context-sensitive dependency pairs. In S. Arun-Kumar and Naveen Garg, editors, *Proceedings of the 26th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 297–308. Springer-Verlag, 2006.
- [3] Beatriz Alarcón, Raúl Gutiérrez, and Salvador Lucas. Improving the context-sensitive dependency graph. *Electronic Notes in Theoretical Computer Science*, 188:91–103, 2007.
- [4] Ernst Althaus, Evgeny Kruglov, and Christoph Weidenbach. Superposition modulo linear arithmetic: SUP(LA). In Ghilardi and Sebastiani [77], pages 84–99.
- [5] Valentin M. Antimirov and Anatoli Degtyarev. Consistency and semantics of equational definitions over predefined algebras. In Michaël Rusinowitch and Jean-Luc Remy, editors, *Proceedings of the 3rd International Workshop on Conditional Term Rewriting Systems (CTRS '92)*, volume 656 of *Lecture Notes in Computer Science*, pages 67–81. Springer-Verlag, 1993.
- [6] Valentin M. Antimirov and Anatoli Degtyarev. Completeness of equational definitions over predefined algebras. In Maurice Nivat, Charles Rattray, Teodor Rus, and Giuseppe Scollo, editors, *Proceedings of the 3rd International Conference on Algebraic Methodology and Software Technology (AMAST '93)*, pages 377–384. Springer-Verlag, 1994.
- [7] Takahito Aoto. Dealing with non-orientable equations in rewriting induction. In Pfenning [140], pages 242–256.

Bibliography

- [8] Takahito Aoto. Designing a rewriting induction prover with an increased capability of non-orientable theorems. In Bruno Buchberger, Tetsuo Ida, and Temur Kutsia, editors, *Proceedings of the Austrian-Japanese Workshop on Symbolic Computation in Software Science (SCSS '08)*, pages 1–15, 2008.
- [9] Takahito Aoto. Soundness of rewriting induction based on an abstract principle. *IPSJ Transactions on Programming*, 49:14–27, 2008.
- [10] Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors. *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR '08)*, volume 5195 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2008.
- [11] Alessandro Armando, Michaël Rusinowitch, and Sorin Stratulat. Incorporating decision procedures in implicit induction. *Journal of Symbolic Computation*, 34(4):241–258, 2002.
- [12] Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1–2):133–178, 2000.
- [13] Jürgen Avenhaus and Klaus Becker. A framework for operational equational specifications with pre-defined structures. *Journal of Symbolic Computation*, 27(3):271–310, 1999.
- [14] Mauricio Ayala-Rincón. Expressiveness of conditional equational systems with built-in predicates. Doktorarbeit, Fachbereich Informatik, Universität Kaiserslautern, Germany, 1993.
- [15] Mauricio Ayala-Rincón. Embedding built-in predicates as premises of rules of conditional rewriting systems. In *Proceedings of the 2nd Workshop on Logic, Language, Information and Computation (WoLLIC '95)*, 1995.
- [16] Franz Baader, editor. *Proceedings of the 19th International Conference on Automated Deduction (CADE '03)*, volume 2741 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2003.
- [17] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
- [18] Franz Baader and Klaus U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *Journal of Symbolic Computation*, 21(2):211–243, 1996.
- [19] Franz Baader and Jörg H. Siekmann. Unification theory. In Gabbay et al. [76], volume 2, pages 41–125.

Bibliography

- [20] Franz Baader and Wayne Snyder. Unification theory. In Robinson and Voronkov [148], volume 1, chapter 8, pages 445–533.
- [21] Leo Bachmair and Nachum Dershowitz. Completion for rewriting modulo a congruence. *Theoretical Computer Science*, 67(2–3):173–201, 1989.
- [22] Thomas Ball and Robert B. Jones, editors. *Proceedings of the 18th International Conference on Computer Aided Verification (CAV '06)*, volume 4144 of *Lecture Notes in Computer Science*. Springer-Verlag, 2006.
- [23] Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, chapter 26, pages 825–885. IOS Press, 2009.
- [24] Clark Barrett and Cesare Tinelli. CVC3. In Werner Damm and Holger Hermanns, editors, *Proceedings of the 19th International Conference on Computer Aided Verification (CAV '07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 298–302. Springer-Verlag, 2007.
- [25] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. ME(LIA): Model evolution with linear integer arithmetic constraints. In Cervesato et al. [41], pages 258–273.
- [26] Hubert Bertling and Harald Ganzinger. Completion-time optimization of rewrite-time goal solving. In Dershowitz [55], pages 45–58.
- [27] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development*. Springer-Verlag, 2004.
- [28] Frédéric Blanqui, William Delobel, Solange Coupet-Grimal, Sébastien Hinderer, and Adam Koprowski. CoLoR: A Coq library on rewriting and termination. In Alfons Geser and Harald Sønderdergaard, editors, *Proceedings of the 8th International Workshop on Termination (WST '06)*, pages 69–73, 2006.
- [29] Frédéric Blanqui, Thérèse Hardin, and Pierre Weis. On the implementation of construction functions for non-free concrete data types. In Rocco De Nicola, editor, *Proceedings of the 16th European Symposium on Programming (ESOP '07)*, volume 4421 of *Lecture Notes in Computer Science*, pages 95–109. Springer-Verlag, 2007.
- [30] Cristina Borralleras, Salvador Lucas, and Albert Rubio. Recursive path orderings can be context-sensitive. In Andrei Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction (CADE '02)*, volume

Bibliography

- 2392 of *Lecture Notes in Artificial Intelligence*, pages 314–331. Springer-Verlag, 2002.
- [31] Cristina Borralleras and Albert Rubio. Monotonic AC-compatible semantic path orderings. In Robert Nieuwenhuis, editor, *Proceedings of the 14th International Conference on Rewriting Techniques and Applications (RTA '03)*, volume 2706 of *Lecture Notes in Computer Science*, pages 279–295. Springer-Verlag, 2003.
- [32] Alexandre Boudet. Combining unification algorithms. *Journal of Symbolic Computation*, 16(6):597–626, 1993.
- [33] Adel Bouhoula. Automated theorem proving by test set induction. *Journal of Symbolic Computation*, 23(1):47–77, 1997.
- [34] Adel Bouhoula and Florent Jacquemard. Automated induction with constrained tree automata. In Armando et al. [10], pages 539–554.
- [35] Robert S. Boyer and J Strother Moore. *A Computational Logic*. Academic Press, 1979.
- [36] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. Linear ranking with reachability. In Kousha Etessami and Sriram K. Rajamani, editors, *Proceedings of the 17th International Conference on Computer Aided Verification (CAV '05)*, volume 3576 of *Lecture Notes in Computer Science*, pages 491–504. Springer-Verlag, 2005.
- [37] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. The polyranking principle. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP '05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 1349–1361. Springer-Verlag, 2005.
- [38] Mark Braverman. Termination of integer linear programs. In Ball and Jones [22], pages 372–385.
- [39] Alan Bundy. The automation of proof by mathematical induction. In Robinson and Voronkov [148], volume 1, chapter 13, pages 845–911.
- [40] Alan Bundy, David Basin, Dieter Hutter, and Andrew Ireland. *Rippling: Meta-Level Guidance for Mathematical Reasoning*. Cambridge University Press, 2005.

Bibliography

- [41] Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors. *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR '08)*, volume 5330 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2008.
- [42] Aziem Chawdhary, Byron Cook, Sumit Gulwani, Mooly Sagiv, and Hongseok Yang. Ranking abstractions. In Sophia Drossopoulou, editor, *Proceedings of the 17th European Symposium on Programming (ESOP '08)*, volume 4960 of *Lecture Notes in Computer Science*, pages 148–162. Springer-Verlag, 2008.
- [43] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *All About Maude: A High-Performance Logical Framework*, volume 4350 of *Lecture Notes in Computer Science*. Springer-Verlag, 2007.
- [44] Michael Colón and Henny B. Sipma. Synthesis of linear ranking functions. In Tiziana Margaria and Wang Yi, editors, *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '01)*, volume 2031 of *Lecture Notes in Computer Science*, pages 67–81. Springer-Verlag, 2001.
- [45] Michael Colón and Henny B. Sipma. Practical methods for proving program termination. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Proceedings of the 14th International Conference on Computer Aided Verification (CAV '02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 442–454. Springer-Verlag, 2002.
- [46] Hubert Comon. Inductionless induction. In Robinson and Voronkov [148], volume 1, chapter 14, pages 913–962.
- [47] Evelyne Contejean, Pierre Courtieu, Julien Forest, Olivier Pons, and Xavier Urbain. Certification of automated termination proofs. In Boris Konev and Frank Wolter, editors, *Proceedings of the 5th International Symposium on Frontiers of Combining Systems (FroCoS '07)*, volume 4720 of *Lecture Notes in Artificial Intelligence*, pages 148–162. Springer-Verlag, 2007.
- [48] Evelyne Contejean, Claude Marché, Ana Paula Tomás, and Xavier Urbain. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning*, 34(4):325–363, 2005.
- [49] Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Abstraction refinement for termination. In Chris Hankin and Igor Siveroni, editors, *Proceedings of the 12th International Symposium on Static Analysis (SAS '05)*, volume

Bibliography

- 3672 of *Lecture Notes in Computer Science*, pages 87–101. Springer-Verlag, 2005.
- [50] Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Termination proofs for systems code. In *Proceedings of the ACM SIGPLAN 2006 Conference on Programming Language Design and Implementation (PLDI '06)*, pages 415–426, 2006.
- [51] Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Terminator: Beyond safety. In Ball and Jones [22], pages 415–418.
- [52] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM Symposium on Principles of Programming Languages (POPL '77)*, pages 238–252. ACM Press, 1977.
- [53] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM Symposium on Principles of Programming Languages (POPL '78)*, pages 84–96. ACM Press, 1978.
- [54] Nachum Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1–2):69–116, 1987.
- [55] Nachum Dershowitz, editor. *Proceedings of the 3rd International Conference on Rewriting Techniques and Applications (RTA '89)*, volume 355 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989.
- [56] Nachum Dershowitz and David A. Plaisted. Rewriting. In Robinson and Voronkov [148], volume 1, chapter 9, pages 535–610.
- [57] Agostino Dovier, Carla Piazza, and Gianfranco Rossi. A uniform approach to constraint-solving for lists, multisets, compact lists, and sets. *ACM Transactions on Computational Logic*, 9(3), 2008.
- [58] Agostino Dovier, Alberto Policriti, and Gianfranco Rossi. A uniform axiomatic view of lists, multisets, and sets, and the relevant unification algorithms. *Fundamenta Informaticae*, 36(2–3):201–234, 1998.
- [59] Francisco Durán, Salvador Lucas, Claude Marché, José Meseguer, and Xavier Urbain. Proving operational termination of membership equational programs. *Higher-Order and Symbolic Computation*, 21(1–2):59–88, 2008.
- [60] Francisco Durán, Salvador Lucas, and José Meseguer. MTT: The Maude Termination Tool. In Armando et al. [10], pages 313–319.

Bibliography

- [61] Francisco Durán, Salvador Lucas, and José Meseguer. Termination modulo combinations of equational theories. In Ghilardi and Sebastiani [77], pages 246–262.
- [62] Bruno Dutertre and Leonardo de Moura. A fast linear-arithmetic solver for DPLL(T). In Ball and Jones [22], pages 81–94.
- [63] Jörg Endrullis, Johannes Waldmann, and Hans Zantema. Matrix interpretations for proving termination of term rewriting. *Journal of Automated Reasoning*, 40(2–3):195–220, 2008.
- [64] Stephan Falke. *Automated Termination Analysis for Equational Rewriting*. Diplomarbeit. Fachgruppe Informatik, Rheinisch-Westfälische Technische Hochschule Aachen, Germany, 2004.
- [65] Stephan Falke and Deepak Kapur. Inductive decidability using implicit induction. In Miki Hermann and Andrei Voronkov, editors, *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 45–59. Springer-Verlag, 2006.
- [66] Stephan Falke and Deepak Kapur. Dependency pairs for rewriting with non-free constructors. In Pfenning [141], pages 426–442.
- [67] Stephan Falke and Deepak Kapur. Dependency pairs for rewriting with built-in numbers and semantic data structures. In Voronkov [169], pages 94–109.
- [68] Stephan Falke and Deepak Kapur. Operational termination of conditional rewriting with built-in numbers and semantic data structures. *Electronic Notes in Theoretical Computer Science*, 237:75–90, 2009.
- [69] Stephan Falke and Deepak Kapur. A term rewriting approach to the automated termination analysis of imperative programs. In Renate A. Schmidt, editor, *Proceedings of the 22nd International Conference on Automated Deduction (CADE '09)*, volume 5663 of *Lecture Notes in Artificial Intelligence*, pages 277–293. Springer-Verlag, 2009.
- [70] Stephan Falke and Deepak Kapur. Termination of context-sensitive rewriting with built-in numbers and collection data structures. In Santiago Escobar, editor, *Proceedings of the 18th International Workshop on Functional and (Constraint) Logic Programming (WFLP '09)*, pages 111–125, 2009.
- [71] Maria C. F. Ferreira and A. L. Ribeiro. Context-sensitive AC-rewriting. In Paliath Narendran and Michaël Rusinowitch, editors, *Proceedings of the 10th*

Bibliography

- International Conference on Rewriting Techniques and Applications (RTA '99)*, volume 1631 of *Lecture Notes in Computer Science*, pages 286–300. Springer-Verlag, 1999.
- [72] Laurent Fribourg. A strong restriction of the inductive completion procedure. *Journal of Symbolic Computation*, 8(3):253–276, 1989.
- [73] Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl. SAT solving for termination analysis with polynomial interpretations. In João Marques-Silva and Karem A. Sakallah, editors, *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT '05)*, volume 4501 of *Lecture Notes in Computer Science*, pages 340–354. Springer-Verlag, 2007.
- [74] Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl. Maximal termination. In Voronkov [169], pages 110–125.
- [75] Carsten Fuhs, Jürgen Giesl, Martin Plücker, Peter Schneider-Kamp, and Stephan Falke. Proving termination of integer term rewriting. In Ralf Treinen, editor, *Proceedings of the 20th International Conference on Rewriting Techniques and Applications (RTA '09)*, volume 5595 of *Lecture Notes in Computer Science*, pages 32–47. Springer-Verlag, 2009.
- [76] Dov M. Gabbay, Christopher J. Hogger, and J. Alan Robinson, editors. *Handbook of Logic in Artificial Intelligence and Logic Programming*. Oxford University Press, 1994.
- [77] Silvio Ghilardi and Roberto Sebastiani, editors. *Proceedings of the 7th International Symposium on Frontiers of Combining Systems (FroCoS '09)*, volume 5749 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2009.
- [78] Jürgen Giesl. *Mechanized Verification of Imperative and Functional Programs*. Habilitationsschrift. Fachbereich Informatik, Technische Universität Darmstadt, Germany, 1999.
- [79] Jürgen Giesl and Thomas Arts. Verification of Erlang processes by dependency pairs. *Applicable Algebra in Engineering, Communication and Computing*, 12(1–2):39–72, 2001.
- [80] Jürgen Giesl and Deepak Kapur. Decidable classes of inductive theorems. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proceedings of the 1st International Joint Conference on Automated Reasoning (IJCAR '01)*,

Bibliography

- volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 469–484. Springer-Verlag, 2001.
- [81] Jürgen Giesl and Deepak Kapur. Dependency pairs for equational rewriting. In Aart Middeldorp, editor, *Proceedings of the 12th International Conference on Rewriting Techniques and Applications (RTA '01)*, volume 2051 of *Lecture Notes in Computer Science*, pages 93–108. Springer-Verlag, 2001.
- [82] Jürgen Giesl and Deepak Kapur. Deciding inductive validity of equations. In Baader [16], pages 17–31.
- [83] Jürgen Giesl and Aart Middeldorp. Transformation techniques for context-sensitive rewrite systems. *Journal of Functional Programming*, 14(4):379–427, 2004.
- [84] Jürgen Giesl, Peter Schneider-Kamp, and René Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In Ulrich Furbach and Natarajan Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR '06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 281–286. Springer-Verlag, 2006.
- [85] Jürgen Giesl, Stephan Swiderski, Peter Schneider-Kamp, and René Thiemann. Automated termination analysis for Haskell: From term rewriting to programming languages. In Pfenning [140], pages 297–312.
- [86] Jürgen Giesl, René Thiemann, and Peter Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In Franz Baader and Andrei Voronkov, editors, *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. Springer-Verlag, 2005.
- [87] Jürgen Giesl, René Thiemann, and Peter Schneider-Kamp. Proving and disproving termination of higher-order functions. In Bernhard Gramlich, editor, *Proceedings of the 5th International Workshop on Frontiers of Combining Systems (FroCoS '05)*, volume 3717 of *Lecture Notes in Artificial Intelligence*, pages 216–231. Springer-Verlag, 2005.
- [88] Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.
- [89] Jürgen Giesl, René Thiemann, Stephan Swiderski, and Peter Schneider-Kamp. Proving termination by bounded increase. In Pfenning [141], pages 443–459.

Bibliography

- [90] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.
- [91] Bernhard Gramlich. Abstract relations between restricted termination and confluence properties of rewrite systems. *Fundamenta Informaticae*, 24(1–2):2–23, 1995.
- [92] Raúl Gutiérrez, Salvador Lucas, and Xavier Urbain. Usable rules for context-sensitive rewrite systems. In Voronkov [169], pages 126–141.
- [93] John V. Guttag and James J. Horning. The algebraic specification of abstract data types. *Acta Informatica*, 10:27–52, 1978.
- [94] Christian Haselbach. *Transformation Techniques to Verify Imperative and Functional Programs*. Diplomarbeit. Fachgruppe Informatik, Rheinisch-Westfälische Technische Hochschule Aachen, Germany, 2004.
- [95] Nao Hirokawa and Aart Middeldorp. Tyrolean Termination Tool: Techniques and features. *Information and Computation*, 205(4):474–511, 2007.
- [96] Hoon Hong and Dalibor Jakuš. Testing positiveness of polynomials. *Journal of Automated Reasoning*, 21(1):23–38, 1998.
- [97] Gérard P. Huet and Jean-Marie Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and Systems Sciences*, 25(2):239–266, 1982.
- [98] Dieter Hutter and Claus Sengler. INKA: The next generation. In Michael A. McRobbie and John K. Slaney, editors, *Proceedings of the 13th International Conference on Automated Deduction (CADE '96)*, volume 1104 of *Lecture Notes in Computer Science*, pages 288–292. Springer-Verlag, 1996.
- [99] Jean-Pierre Jouannaud. Modular Church-Rosser modulo. In Pfenning [140], pages 96–107.
- [100] Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, chapter 8, pages 257–321. The MIT Press, 1991.
- [101] Jean-Pierre Jouannaud and Hélène Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal on Computing*, 15(4):1155–1194, 1986.

Bibliography

- [102] Jean-Pierre Jouannaud and Emmanuel Kounalis. Automatic proofs by induction in theories without constructors. *Information and Computation*, 82(1):1–33, 1989.
- [103] Samuel Kamin and Jean-Jaques Lévy. Attempts for generalizing the recursive path orderings. Unpublished note, Department of Computer Science, University of Illinois, Urbana-Champaign, USA, 1980.
- [104] Deepak Kapur, Jürgen Giesl, and Mahadevan Subramaniam. Induction and decision procedures. *Revista de la Real Academia de Ciencias, Serie A: Matemáticas*, 98(1):153–180, 2004.
- [105] Deepak Kapur and David R. Musser. Proof by consistency. *Artificial Intelligence*, 31(2):125–157, 1987.
- [106] Deepak Kapur, Paliath Narendran, and Hantao Zhang. On sufficient-completeness and related properties of term rewriting systems. *Acta Informatica*, 24(4):395–415, 1987.
- [107] Deepak Kapur, Paliath Narendran, and Hantao Zhang. Automating induction-less induction using test sets. *Journal of Symbolic Computation*, 11(1–2):81–111, 1991.
- [108] Deepak Kapur and G. Sivakumar. Proving associative-commutative termination using RPO-compatible orderings. In Ricardo Caferra and Gernot Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*, volume 1761 of *Lecture Notes in Artificial Intelligence*, pages 40–62. Springer-Verlag, 2000.
- [109] Deepak Kapur and Mahadevan Subramaniam. Automating induction over mutually recursive functions. In Martin Wirsing and Maurice Nivat, editors, *Proceedings of the 5th International Conference on Algebraic Methodology and Software Technology (AMAST '96)*, volume 1101 of *Lecture Notes in Computer Science*, pages 117–131. Springer-Verlag, 1996.
- [110] Deepak Kapur and Mahadevan Subramaniam. New uses of linear arithmetic in automated theorem proving by induction. *Journal of Automated Reasoning*, 16(1–2):39–78, 1996.
- [111] Deepak Kapur and Mahadevan Subramaniam. Extending decision procedures with induction schemes. In David A. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE '00)*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 324–345. Springer-Verlag, 2000.

Bibliography

- [112] Deepak Kapur and Hantao Zhang. An overview of Rewrite Rule Laboratory (RRL). *Computers & Mathematics with Applications*, 29(2):91–114, 1995.
- [113] Matt Kaufmann, Panagiotis Manolios, and J Strother Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, 2000.
- [114] Donald Knuth and Peter Bendix. Simple word problems in universal algebras. In John Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.
- [115] Hirotaka Koike and Yoshihito Toyama. Inductionless induction and rewriting induction. *JSSST Computer Software*, 17(6):1–12, 2000. In Japanese.
- [116] Konstantin Korovin and Andrei Voronkov. Integrating linear arithmetic into superposition calculus. In Jacques Duparc and Thomas A. Henzinger, editors, *Proceedings of the 21st International Workshop on Computer Science Logic (CSL '07)*, volume 4646 of *Lecture Notes in Computer Science*, pages 223–237. Springer-Verlag, 2007.
- [117] Keiichirou Kusakari, Masaki Nakamura, and Yoshihito Toyama. Argument filtering transformation. In Gopalan Nadathur, editor, *Proceedings of the 1st International Conference on Principles and Practice of Declarative Programming (PPDP '99)*, volume 1702 of *Lecture Notes in Computer Science*, pages 47–61. Springer-Verlag, 1999.
- [118] Keiichirou Kusakari and Yoshihito Toyama. On proving AC-termination by AC-dependency pairs. *IEICE Transactions on Information and Systems*, E84-D(5):604–612, 2001.
- [119] Dallas S. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Mathematics Department, Louisiana Tech University, Ruston, 1979.
- [120] Dallas S. Lankford and Mike A. Ballantyne. Decision procedures for simple equational theories with permutative axioms: Complete sets of permutative reductions. Technical Report ATP-39, Mathematics Department, University of Texas, Austin, 1977.
- [121] Salvador Lucas. Context-sensitive computations in functional and functional logic programs. *Journal of Functional and Logic Programming*, 1998(1), 1998.
- [122] Salvador Lucas. Termination of rewriting with strategy annotations. In Robert Nieuwenhuis and Andrei Voronkov, editors, *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*

Bibliography

- (*LPAR '01*), volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 669–684. Springer-Verlag, 2001.
- [123] Salvador Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):294–343, 2002.
- [124] Salvador Lucas. Lazy rewriting and context-sensitive rewriting. *Electronic Notes in Theoretical Computer Science*, 64:234–254, 2002.
- [125] Salvador Lucas. Polynomials for proving termination of context-sensitive rewriting. In Igor Walukiewicz, editor, *Proceedings of the 7th International Conference on Foundations of Software Science and Computation Structures (FOSSACS '04)*, volume 2987 of *Lecture Notes in Computer Science*, pages 318–332. Springer-Verlag, 2004.
- [126] Salvador Lucas. Proving termination of context-sensitive rewriting by transformation. *Information and Computation*, 204(12):1782–1846, 2006.
- [127] Salvador Lucas, Claude Marché, and José Meseguer. Operational termination of conditional term rewriting systems. *Information Processing Letters*, 95(4):446–453, 2005.
- [128] Claude Marché. Normalized rewriting: An alternative to rewriting modulo a set of equations. *Journal of Symbolic Computation*, 21(3):253–288, 1996.
- [129] Claude Marché and Xavier Urbain. Termination of associative-commutative rewriting by dependency pairs. In Tobias Nipkow, editor, *Proceedings of the 9th International Conference on Rewriting Techniques and Application (RTA '98)*, volume 1379 of *Lecture Notes in Computer Science*, pages 241–255. Springer-Verlag, 1998.
- [130] Claude Marché and Xavier Urbain. Modular and incremental proofs of AC-termination. *Journal of Symbolic Computation*, 38(1):873–897, 2004.
- [131] Massimo Marchiori. Unravelings and ultra-properties. In Michael Hanus and Mario Rodríguez-Artalejo, editors, *Proceedings of the 5th International Conference on Algebraic and Logic Programming (ALP '96)*, volume 1139 of *Lecture Notes in Computer Science*, pages 107–121. Springer-Verlag, 1996.
- [132] John McCarthy. Recursive functions of symbolic expressions and their computation by machine. *Communications of the ACM*, 3(4):184–195, 1960.
- [133] Antoine Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.

Bibliography

- [134] David R. Musser. On proving inductive properties of abstract data types. In *Proceedings of the 7th ACM Symposium on Principles of Programming Languages (POPL '80)*, pages 154–162. ACM Press, 1980.
- [135] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.
- [136] Enno Ohlebusch. Transforming conditional rewrite systems with extra variables into unconditional systems. In Harald Ganzinger, David A. McAllester, and Andrei Voronkov, editors, *Proceedings of the 6th International Conference on Logic Programming and Automated Reasoning (LPAR '99)*, volume 1705 of *Lecture Notes in Artificial Intelligence*, pages 111–130. Springer-Verlag, 1999.
- [137] Enno Ohlebusch. *Advanced Topics in Term Rewriting*. Springer-Verlag, 2002.
- [138] Carsten Otto. *Automated Termination Analysis for Imperative Programs with Dynamic Data Structures*. Diplomarbeit. Fachgruppe Informatik, Rheinisch-Westfälische Technische Hochschule Aachen, Germany, 2008.
- [139] Gerald E. Peterson and Mark E. Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM*, 28(2):233–264, 1981.
- [140] Frank Pfenning, editor. *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, volume 4098 of *Lecture Notes in Computer Science*. Springer-Verlag, 2006.
- [141] Frank Pfenning, editor. *Proceedings of the 21st International Conference on Automated Deduction (CADE '07)*, volume 4603 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2007.
- [142] David A. Plaisted. Equational reasoning and term rewriting systems. In Gabbay et al. [76], volume 1, pages 274–367.
- [143] Martin Plücker. *Termination of Integer Term Rewrite Systems*. Diplomarbeit. Fachgruppe Informatik, Rheinisch-Westfälische Technische Hochschule Aachen, Germany, 2009.
- [144] Andreas Podelski and Andrey Rybalchenko. A complete method for the synthesis of linear ranking functions. In Bernhard Steffen and Giorgio Levi, editors, *Proceedings of the 5th Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI '04)*, volume 2937 of *Lecture Notes in Computer Science*, pages 239–251. Springer-Verlag, 2004.

Bibliography

- [145] Andreas Podelski and Andrey Rybalchenko. Transition invariants. In *Proceedings of the 19th IEEE Symposium on Logic in Computer Science (LICS '04)*, pages 32–41. IEEE Computer Society, 2004.
- [146] Mojzesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du Premier Congrès de Mathématiciens des Pays Slaves*, pages 92–101, 1929.
- [147] Uday S. Reddy. Term rewriting induction. In Mark E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction (CADE '90)*, volume 449 of *Lecture Notes in Computer Science*, pages 162–177. Springer-Verlag, 1990.
- [148] J. Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning*. Elsevier Science Publishers, 2001.
- [149] Barry K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *Journal of the ACM*, 20(1):160–187, 1973.
- [150] Albert Rubio. A fully syntactic AC-RPO. *Information and Computation*, 178(2):515–533, 2002.
- [151] Philipp Rümmer. A constraint sequent calculus for first-order logic with linear integer arithmetic. In Cervesato et al. [41], pages 274–289.
- [152] Manfred Schmidt-Schauß. Unification in a combination of arbitrary disjoint equational theories. *Journal of Symbolic Computation*, 8(1–2):51–99, 1989.
- [153] Peter Schneider-Kamp, Jürgen Giesl, Alexander Serebrenik, and René Thiemann. Automated termination analysis for logic programs by term rewriting. *ACM Transactions on Computational Logic*, 11(1), 2010. To appear.
- [154] Roberto Sebastiani. Lazy satisfiability modulo theories. *Journal on Satisfiability, Boolean Modeling and Computation*, 3(3–4):141–224, 2007.
- [155] Matthias Sondermann. *Automatische Terminierungsanalyse für imperative Programme*. Diplomarbeit. Fachgruppe Informatik, Rheinisch-Westfälische Technische Hochschule Aachen, Germany, 2007. In German.
- [156] Christian Stein. *Das Dependency Pair Framework zur automatischen Terminierungsanalyse von Termersetzung modulo Gleichungen*. Diplomarbeit. Fachgruppe Informatik, Rheinisch-Westfälische Technische Hochschule Aachen, Germany, 2006. In German.

Bibliography

- [157] Sorin Stratulat. Combining rewriting with Noetherian induction to reason on non-orientable equalities. In Voronkov [169], pages 351–365.
- [158] Mahadevan Subramaniam. *Failure Analyses of Inductive Theorem Provers*. Dissertation. Department of Computer Science, State University of New York at Albany, USA, 1996.
- [159] Mahadevan Subramaniam, Deepak Kapur, and Stephan Falke. Predicting Failures of Inductive Proof Attempts. In Wolfgang Ahrendt, Peter Baumgartner, and Hans de Nivelles, editors, *Proceedings of the 3rd Workshop on Disproving: Non-Theorems, Non-Validity, Non-Provability (DISPROVING '06)*, pages 70–81, 2006.
- [160] Mahadevan Subramaniam, Deepak Kapur, and Stephan Falke. Predicting failures of and repairing inductive proof attempts. In S. Ramesh and P. Sampath, editors, *Next Generation Design and Verification Methodologies for Distributed Embedded Control Systems*, pages 177–192. Springer-Verlag, 2007.
- [161] See <http://www.lri.fr/~marche/termination-competition/2007/>.
- [162] Termination problem data base 5.0.2, 2009. Available from <http://dev.aspsimon.org/projects/termcomp/downloads/>.
- [163] René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In Stephan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics (TPHOLs '09)*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468. Springer-Verlag, 2009.
- [164] Ashish Tiwari. Termination of linear programs. In Rajeev Alur and Doron Peled, editors, *Proceedings of the 16th Conference on Computer Aided Verification (CAV '04)*, volume 3114 of *Lecture Notes in Computer Science*, pages 70–82. Springer-Verlag, 2004.
- [165] Yoshihito Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1):128–143, 1987.
- [166] Alan Turing. On computable numbers, with an application to the Entscheidungsproblem. In *Proceedings of the London Mathematical Society*, volume 42, pages 230–265, 1937.
- [167] Xavier Urbain. Modular & incremental automated termination proofs. *Journal of Automated Reasoning*, 32(4):315–355, 2002.

Bibliography

- [168] Sergei G. Vorobyov. Conditional rewrite rule systems with built-in arithmetic and induction. In Dershowitz [55], pages 492–512.
- [169] Andrei Voronkov, editor. *Proceedings of the 19th International Conference on Rewriting Techniques and Applications (RTA '08)*, volume 5117 of *Lecture Notes in Computer Science*. Springer-Verlag, 2008.
- [170] Christoph Walther. Mathematical induction. In Gabbay et al. [76], volume 2, pages 127–228.
- [171] Christoph Walther and Stephan Schweitzer. About VeriFun. In Baader [16], pages 322–327.
- [172] Wikipedia. List of software bugs. See http://en.wikipedia.org/wiki/List_of_software_bugs.
- [173] Hans Zantema. Termination. In TeReSe, editor, *Term Rewriting Systems*, chapter 6, pages 181–259. Cambridge University Press, 2003.
- [174] Hantao Zhang, Deepak Kapur, and Mukkai S. Krishnamoorthy. A mechanizable induction principle for equational specifications. In Ewing L. Lusk and Ross A. Overbeek, editors, *Proceedings of the 9th International Conference on Automated Deduction (CADE '88)*, volume 310 of *Lecture Notes in Computer Science*, pages 162–181. Springer-Verlag, 1988.