

Rewriting Induction for Higher-Order Constrained Rewriting

~

Kasper Hagens

Rewriting Induction for Higher-Order Constrained Rewriting

Kasper Hagens

Radboud University



The research reported in this thesis has been carried out under the auspices of iCIS (Institute for Computing and Information Sciences, Radboud University) and the research school IPA (Institute for Programming research and Algorithmics). This work has been financially supported by NWO VI.Vidi.193.075, project “CHORPE” <https://www.nwo.nl/en/projects/vividi193075>.

IPA Dissertation series, number 2026-16

Kasper Hagens

Rewriting Induction for Higher-Order Constrained Rewriting

Radboud Dissertation Series

ISSN: 2950-2772 (Online); 2950-2780 (Print)

Published by RADBOUD UNIVERSITY PRESS
Postbus 9100, 6500 HA Nijmegen, The Netherlands
www.radbouduniversitypress.nl

Design inside: Kasper Hagens
Cover image: Kasper Hagens and ProefschriftOntwerp.nl
Production coordination & prepress: ProefschriftOntwerp.nl
Printing: ProefschriftMaken/Pumbo

ISBN: 9789465153155

DOI: 10.54195/9789465153155

Free download at: <https://doi.org/10.54195/9789465153155>

© 2026 Kasper Hagens

**RADBOUD
UNIVERSITY
PRESS**

This is an Open Access book published under the terms of Creative Commons Attribution-Non-commercial-NoDerivatives International license (CC BY-NC-ND 4.0). This license allows re-users to copy and distribute the material in any medium or format in unadapted form only, for noncommercial purposes only, and only so long as attribution is given to the creator, see <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

Rewriting Induction for Higher-Order Constrained Rewriting

Proefschrift

ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de rector magnificus prof. dr. J.M. Sanders,
volgens besluit van het college college voor promoties
in het openbaar te verdedigen op

dinsdag 8 december 2026
om 10.30 uur precies

door

Kasper Hagens

geboren op 27 november 1990
te Venray

Promotor:

prof. dr. J.H. Geuvers

Copromotor:

dr. C.L.M. Kop

Manuscriptcommissie:

prof. dr. R.J. Krebbers (voorzitter)

prof. dr. N. Nishida (Nagoya University, Japan)

prof. dr. T. Aoto (Niigata University, Japan)

dr. S. Ciobâcă (Alexandru Ioan Cuza University, Roemenië)

dr. D. Milovancevic (Imperial College London, Verenigd Koninkrijk)

Acknowledgment

When I completed my Master's degree at the end of 2021, I was very grateful for the opportunity to pursue my mathematical interests through a PhD. Thank you, Cynthia, not only for giving me this chance, but also for your excellent supervision throughout the years. During my PhD, I learned that research does not always proceed as smoothly as expected. At times, I found myself in a state of confusion, and you helped me break problems down into smaller, manageable pieces. Thank you, Cynthia, for your constant patience and for the many times I could come to your office for help. My PhD would not have been possible without your moral support and technical guidance. I also want to thank you, Herman, for the yearly evaluations and useful feedback on the writing of this thesis.

As a large part of a PhD is spent on solitary activities, I am grateful to have had many great colleagues in the Software Science department around me. In particular, I want to thank Liye and Deivid for being steadfast office mates: thank you for fostering an informal atmosphere, full of inside jokes. Together, we have shared many moments, both in and outside the office, including some memorable trips abroad. Thank you, Deivid, for hosting the annual Christmas dinner and for introducing us to Brazilian picanha. Thank you, Liye, for introducing us to the only decent Chinese restaurant in the Netherlands.

Thank you, Bharat, for bringing a lot of energy into our office and for the enjoyable conversations and coffee breaks on the second floor. Dennis, I really enjoyed your presence at the ICT open events, and I am still grateful that you got me a bike so that I could join your tour to Kranenburg. Ingrid and Hilde, thank you for helping us with the complicated administrative issues, especially with conference travel. Also, I want to thank Thomas and Simcha. Although our time together was relatively short, towards the end of my PhD, you were both indispensable in keeping the office, and therefore me, alive. Especially the many coffee walks to the sixth floor. Furthermore, I want to thank the members of the manuscript committee for their time and valuable feedback. Last but not least, I am very grateful to my parents for creating the supportive conditions that enabled me to complete this PhD. Their practical and constant support, including countless meals and a welcoming place to stay in Limburg on weekends, was invaluable throughout this journey.

Table of contents

1	Introduction	1
1.1	Objectives	6
1.2	Overview and contributions	12
2	Preliminaries	19
3	Higher-Order Program Equivalence	25
3.1	Rewriting Induction	27
3.1.1	RI for first-order rewriting	27
3.1.2	Higher-Order Inductive Theorems	29
3.1.3	Higher-order quasi-reductivity	31
3.2	Higher-Order Rewriting Induction	32
3.2.1	Deduction rules	32
3.2.2	Comparison to the first-order literature	40
3.2.3	Proof: Soundness of HORI	40
3.3	Global Rewriting Induction	46
3.3.1	Strong quasi-reductivity	47
3.3.2	Natural extensions and global inductive theorems	48
3.3.3	Oracle extensions	50
3.3.4	Global Rewriting Induction	51
3.3.5	Proving oracle termination	52
3.3.6	Proof: Soundness of Global RI	53
3.4	Conclusion and future work	61
4	Lemma Generation	63
4.1	Initialization Generalization	66
4.2	Matrix Invariants	69
4.2.1	Simple divergence	71
4.2.2	Divergence space	73
4.2.3	Matrix Invariants	73
4.2.4	Computing initial vector and divergence substitution	76

4.2.5	Constrained divergence shape	79
4.3	Inequality invariants	80
4.3.1	Divergence constraint	81
4.3.2	InGen-like inequalities	81
4.3.3	Derived invariants	84
4.4	Non-affine matrix Invariants	86
4.5	Mathematical context	88
4.6	Implementation	92
4.7	Conclusion and future work	93
5	Bounded Rewriting Induction	95
5.1	Preliminaries	97
5.2	Bounded Rewriting Induction	98
5.2.1	Bounded Rewriting Induction	99
5.2.2	Explanation by example	103
5.2.3	Completeness	112
5.3	Proofs	117
5.3.1	Bounded equation contexts	118
5.3.2	Multiset orderings	120
5.3.3	Bounded ground convertibility	120
5.3.4	Soundness of Bounded Rewriting Induction	125
5.3.5	Completeness	128
5.4	Finding a Bounding Pair	130
5.4.1	Using a reduction ordering	130
5.4.2	Separating top and inner steps	131
5.5	Obtaining new deduction rules more easily	135
5.6	Ground Confluence	138
5.6.1	Critical pairs	138
5.6.2	Ground confluence by Bounded RI	145
5.7	Implementation	148
5.8	Comparison to related work	149
5.9	Conclusion and future work	153
6	Program Equivalence by Templates	157
6.1	Templates	160
6.1.1	Recursors	162
6.2	Applying Recursor Equivalences	162
6.2.1	One-sided matching	163
6.2.2	Two-sided matching	164
6.2.3	Conditional recursor equivalence	165

6.2.4	Another example: equivalence with composite operators	167
6.3	Integrating templates in Rewriting Induction	167
6.3.1	Template-recursor equivalences	168
6.3.2	Recursor equivalences	168
6.3.3	Conditional recursor equivalences	169
6.3.4	Templates as lemma generation method	170
6.4	Conclusion and future work	171
6.4.1	Related work	172
6.4.2	Future work	173
7	Conclusion and future Work	175
	References	181
	Samenvatting	186
	Summary	187
	Research Data Management	189
	Curriculum Vitae	195

Chapter 1

Introduction

In the field of software engineering, program correctness is an invaluable property. Programs are written to capture a certain computational behavior, and it is often implicitly assumed that the actual program behavior matches the intended behavior. However, in modern society, where interaction with software has become a daily routine, even non-experts readily recognize that this assumption does not always hold. For example, an email app showing an incorrect number of unread emails is certainly not intended behavior. While the implications in such an example can be considered relatively mild, the consequences of unexpected software behavior can be more severe. At the extreme end, in safety-critical systems such as medical devices or traffic control systems, failures can potentially endanger human life. This motivates the need for reliable methods to reason about program correctness.

Software verification is the area in computer science that aims to bridge this gap. Well-established mathematical methods, such as model checking, have been developed to automatically detect bugs by testing a program against a given specification. Although such methods can effectively reveal the presence of bugs, they are not necessarily suitable for proving the absence of bugs. Depending on the specific setting, program correctness can be considered from different perspectives, each associated with its own methodology.

In this thesis, we consider software verification by means of rigorous mathematical correctness proofs. Such reasoning is particularly relevant in software engineering, for example when we wish to replace a piece of code by an equivalent (e.g. more efficient) implementation. For example, consider the following two programs that both seek to

compute $x \mapsto \sum_{i=1}^x i$ for $x \geq 1$ (and both compute 0 for $x < 1$).

```

int sumIter(int x){
  int i=1; int z=0;
  while (i<=x){
    z=z+i;
    i=i+1;}
  return z; }

```

```

int sumRec(int x){
  if (x>=1)
    return (x + sumRec(x-1));
  else
    return 0; }

```

Although both programs have the same input-output behavior, the iterative implementation is more memory-efficient because it avoids the overhead of recursive calls. Consequently, in a practical setting it may be desirable to replace `sumRec` by `sumIter`. However, correctness of such a transformation requires verifying that these programs really produce the same output for every possible input (of which there are theoretically infinitely many). In this thesis, we will refer to this specific equivalence notion as “program equivalence” (other notions of program equivalence, such as contextual equivalence, can also be found in the literature). In general, program equivalence is undecidable. Nevertheless, we aim to automate such justifications as much as possible.

Term rewriting Mentioning programs, one likely thinks about languages such as C, Java, Python, etc. Such languages are used to write executable computer code, but when we want to formally reason about it, we must establish a way to extract the meaning of code. A common approach is to provide a language with (e.g. operational) semantics, explaining how code behaves during execution. This allows us to define two programs as equivalent if they have the same interpretation with respect to a given semantics.

In this thesis, we will not restrict to a particular programming language, but instead use Term Rewriting Systems (TRSs). This is a model of computation using terms to represent program states, and using rewrite rules to perform state transitions. The rewrite relation can then be seen as a language-independent operational semantics.

For example, let us illustrate how to use term rewriting to model a program even on the natural numbers, that outputs `true` on all even numbers, and `false` otherwise. Each $n \in \mathbb{N}$ is represented in unary notation, using the term $s^n(0)$. We introduce a function symbol `even :: nat → bool`, which is defined via the following three rewrite rules:

- (1). $\text{even}(0) \rightarrow \text{true}$ (2). $\text{even}(s(0)) \rightarrow \text{false}$ (3). $\text{even}(s(s(x))) \rightarrow \text{even}(x)$

We may compose individual rewrite steps to represent a full computation path, from input to output. For example, a computation path witnessing that 5 is not even is obtained by composing three rewrite steps

$$\text{even} \left(s(s(\boxed{s(s(0))})) \right) \xrightarrow{(3)} \text{even} \left(s(s(\boxed{s(0)})) \right) \xrightarrow{(3)} \text{even} (s(0)) \xrightarrow{(2)} \text{false}$$

Here, the superscripts on the arrows indicate which rule is applied. The boxes around the terms in the first two rewrite steps indicate the term to be substituted for x in applying the rewrite rule (3).

Programming beyond inductive data structures Term rewriting, as shown in the example above, reflects a structure similar to functional programming, using constructs like pattern matching and recursive function definitions. A big limitation, however, is that classical term rewriting can only deal with inductive data structures. This in contrast to real programming where non-inductive data structures, like integers, play a prominent role. Most programming languages have a native datatype `Int` where all standard arithmetical operators (e.g. `+` and `*`) and predicates (e.g. `<`) are predefined. On the other hand, in classical term rewriting integers and arithmetical operators can only be encoded via rewrite rules. While the inductive reasoning in classical rewriting might be useful to formalize basic arithmetical properties, it is not an ideal model for analyzing real programming code because, on the formal level, there is no distinction between the rules that define the program behavior and the rules that define the underlying data structures. This makes it less convenient to use as a formal verification framework.

Towards real programming *Logically Constrained Term Rewriting Systems (LCTRSs)*, introduced in 2013 [35], provide a formalism in which rewrite rules are of the form $s \rightarrow t [\varphi]$. Here, the boolean constraint φ acts as a guard to regulate control flow over non-inductive primitive data structures. Constraints are *theory terms*: built from variables and *theory symbols* (e.g. `+` and `*`). *Ground* instances (meaning variable-free instances) of theory terms have a semantical interpretation. For example, addition on integers is interpreted as addition on $\mathbb{Z}$. So for a theory term $t := (x + y \geq 2)$ and *ground substitution* $\gamma = [x := 2, y := 3]$ (a substitution that maps every variable to a variable-free term) the instance $t\gamma$ has the semantical interpretation $\llbracket t\gamma \rrbracket = \llbracket 2 + 3 \geq 2 \rrbracket = (2 + 3 \geq 2) = \top$.

This approach makes LCTRSs very useful for program verification: the term rewriting can directly focus on the programming structure, while reasoning over non-inductive data types (i.e. theory terms) is delegated to optimized external solvers (e.g. SMT solvers for integer arithmetic).

To illustrate, the `even` program can be represented using integers, which are more common in real programming than natural numbers, as follows (restricting to non-negative integer inputs)

$$\begin{aligned} \text{even}(x) &\rightarrow \text{true} & [x \leq 0] & & \text{even}(x) &\rightarrow \text{even}(x - 2) & [x \geq 2] \\ \text{even}(x) &\rightarrow \text{false} & [x = 1] \end{aligned}$$

In LCTRSs, the application of a rule $s \rightarrow t [\varphi]$ in a rewrite step $C[s\gamma] \rightarrow_{\mathcal{R}} C[t\gamma]$ is restricted to those substitutions γ that satisfy $\llbracket \varphi\gamma \rrbracket = \top$. For example, we may apply the rule $\text{even}(x) \rightarrow \text{even}(x - 2) [x \geq 2]$ with $\gamma = [x := 5]$ to perform $\text{even}(5) \rightarrow_{\mathcal{R}} \text{even}(5 - 2)$ because $\llbracket 5 \geq 2 \rrbracket = \top$, but we can e.g. not use $\gamma = [x := 1]$. A specific feature of constrained rewriting is the presence of *calculation rules*, providing the connection between semantical interpretations and the reduction relation on theory terms. For example the semantical calculation $5 - 2 = 3$ in $\mathbb{Z}$ is converted to a corresponding syntactical rewrite step $5 - 2 \rightarrow 3$ using the calculation rule $x - y \rightarrow z [z = x - y]$. This, for instance, allows us to perform the rewrite step $\text{even}(5 - 2) \rightarrow \text{even}(3)$.

LCTRSs are not limited to functional languages. They may also represent control flow arising in imperative programming [40]. As an example, we manually represent the accumulating loop in the `sumIter` program using the following LCTRS

$$\begin{aligned} \text{sumI}(x) &\rightarrow \text{while}(x, 1, 0) \\ \text{while}(x, i, z) &\rightarrow \text{while}(x, i + 1, z + i) \quad [i \leq x] \\ \text{while}(x, i, z) &\rightarrow z \quad [i > x] \end{aligned}$$

Higher-order term constrained rewriting While LCTRSs can represent a wide range of programming constructs, they suffer from a significant limitation: they do not support higher-order behavior – a feature widely used in real-life programming and fundamental to functional languages. For example, the following Haskell function

```
mapPos :: (Int → Int) → [Int] → [Int]
mapPos f [] = []
mapPos f (x:l)
  | x > 0    = (f x):(mapPos f l)
  | otherwise = x:(mapPos f l)
```

takes an argument f of type $\text{int} \rightarrow \text{int}$. To facilitate such definitions, we resort to the recently introduced *Logically Constrained Simply-typed Term Rewriting Systems (LCSTRSs)* [25]. This constitutes an extension of LCTRSs to support applicative higher-order rewriting (without λ), while restricting to first-order constraints.

The `mapPos` program is represented by the following LCSTRS

$$\begin{aligned} \text{mapPos } f \text{ nil} &\rightarrow \text{nil} \\ \text{mapPos } f (x : l) &\rightarrow (f x) : (\text{mapPos } f l) \quad [x > 0] \\ \text{mapPos } f (x : l) &\rightarrow x : (\text{mapPos } f l) \quad [x \leq 0] \end{aligned}$$

Program equivalence by ground convertibility Fixing the computational model to LCSTRSs, we address the question of how to precisely define program equivalence. Since a rewrite rule essentially encodes an equality together with a computational

direction, it is natural to interpret a set of rewrite rules $\mathcal{R}$ as if they were equalities (i.e., disregarding their directional nature) that define the elementary axioms.

This leads to an approach, well known in first-order rewriting, in which program equivalence is defined via the *convertibility* relation $\leftrightarrow_{\mathcal{R}}^*$ [22], defined as the smallest equivalence relation which contains the rewrite system and which is closed under substitutions and application of contexts. Programs correspond to terms in $T(\Sigma, \mathcal{V})$, whose reduction behavior is determined by a rewrite system $\mathcal{R}$, and two terms are declared equivalent if and only if all their *ground* instances are convertible. This interpretation leads to the notion of inductive theorem: an equation $s \approx t$ is an *inductive theorem* (a.k.a. a *ground convertible*) if $s\gamma \leftrightarrow_{\mathcal{R}}^* t\gamma$ for every ground substitution γ .

For example, let us recall the imperative and recursive implementations `sumIter` and `sumRec`, represented by the LCSTRSs

$$\begin{array}{ll} \text{sumI}(x) \rightarrow \text{while}(x, 1, 0) & \text{sumR}(x) \rightarrow 0 \quad [x < 1] \\ \text{while}(x, i, z) \rightarrow \text{while}(x, i + 1, z + i) \quad [i \leq x] & \text{sumR}(x) \rightarrow x + \text{sumR}(x - 1) \quad [x \geq 1] \\ \text{while}(x, i, z) \rightarrow z & [i > x] \end{array}$$

To prove equivalence of $\text{sumI}(x)$ and $\text{sumR}(x)$, we should establish that $\text{sumI}(x) \approx \text{sumR}(x)$ is an inductive theorem, i.e. that $\text{sumI}(x)\gamma \leftrightarrow_{\mathcal{R}}^* \text{sumR}(x)\gamma$ for every ground substitution γ . An example of a ground substitution is $\gamma = [x := 1]$, and for this particular choice we indeed have $\text{sumI}(x)\gamma = \text{sumI}(1) \leftrightarrow_{\mathcal{R}}^* \text{sumR}(1) = \text{sumR}(x)\gamma$, as verified by the following reduction sequence

$$\begin{aligned} \text{sumI}(1) &\rightarrow_{\mathcal{R}} \text{while}(1, 1, 0) \rightarrow_{\mathcal{R}} \text{while}(1, 1 + 1, 0 + 1) \rightarrow_{\mathcal{R}} \text{while}(1, 2, 1) \\ &\rightarrow_{\mathcal{R}} 1 \quad \mathcal{R} \leftarrow 1 + 0 \quad \mathcal{R} \leftarrow 1 + \text{sumR}(0) \quad \mathcal{R} \leftarrow 1 + \text{sumR}(1 - 1) \quad \mathcal{R} \leftarrow \text{sumR}(1) \end{aligned}$$

To prove convertibility for *all* ground substitutions γ , we may use well-founded induction. In this example, we may perform induction on the integer input-variable x . Although $(\mathbb{Z}, <)$ is not well-founded, we can treat all cases with $x < 1$ as part of the base case; thereby effectively using the well-founded ordering $(\mathbb{N}, <)$. In general, however, the induction subject may differ between examples, and finding a suitable induction scheme can be a challenge in itself.

This is precisely where term rewriting systems have an advantage, at least if they are *terminating*. If $\mathcal{R}$ is terminating, then by definition the rewrite relation $\rightarrow_{\mathcal{R}}$ is a well-founded ordering on the set of all terms $T(\Sigma, \mathcal{V})$ and directly represents the program's computational behavior, providing a natural basis for well-founded induction proofs. A benefit of a rewriting-based approach is that it permits the development of proof systems where existing termination methods can be utilized to guide the introduction and application of induction hypotheses, thereby reducing the burden of finding suitable induction schemes or decreasing measures.

Partial equivalence In constrained rewriting, constraints can be attached not only to rules but also to equations. This is useful for expressing partial equivalence, referring to equivalence that only holds on a restricted input space. For instance, we may consider an alternative definition of sumR , where the rule $\text{sumR}(x) \rightarrow 0 \ [x < 1]$ is replaced by $\text{sumR}(x) \rightarrow x \ [x < 1]$. Then the equivalence $\text{suml}(x) \approx \text{sumR}(x)$ no longer holds for all integer inputs, but it still holds for all $x \geq 0$. This partial equivalence can be formally expressed by the constrained equation $\text{suml}(x) \approx \text{sumR}(x) \ [x \geq 0]$. More specifically, if we say that $s \approx t \ [\varphi]$ is an inductive theorem, this means that $s\gamma \leftrightarrow_{\mathcal{R}}^* t\gamma$ for every ground substitution γ that satisfies $\llbracket \varphi\gamma \rrbracket = \top$.

1.1 Objectives

This thesis aims to achieve five main objectives:

- (1). Define a notion of program equivalence for LCSTRSs.
- (2). Develop a formal system for proving program equivalence for LCSTRSs.
- (3). Extend this system to disprove equations that do not represent program equivalence.
- (4). Develop a method for establishing ground confluence in LCSTRSs.
- (5). Develop heuristics that facilitate semi-automatic, human-guided program equivalence proofs in practice.

As we will explain, a system for disproving equations that do not represent program equivalence (objective 3) can only be established under the assumption of ground confluence. This provides one of the motivations for objective 4.

Objective 1: define program equivalence for LCSTRSs

Although we can follow the first-order strategy to LCSTRSs, using ground convertibility to represent program equivalence, we question whether this was in fact the appropriate notion for program verification to begin with, even in the setting of first-order rewriting.

Extensibility. One particular disadvantage of ground convertibility is that it does not satisfy extensibility [9]. Intuitively, this property expresses that local provable properties automatically scale globally. Considering software verification, extensibility allows us to reason about a fragment of a program by considering only the code directly related to it. For example, for functional languages like Haskell, it ensures that importing functions from external modules does not break established equivalences.

Besides being convenient to have, the lack of extensibility can be problematic, as it permits equivalences that depend on properties of program code considered in isolation. This is demonstrated by an illustrative example from unconstrained rewriting:

Assume a signature $\Sigma = \{0 :: \text{int}, 1 :: \text{int}, \text{eval}_0 :: (\text{int} \rightarrow \text{int}) \rightarrow \text{int}, \text{const}_1 :: \text{int} \rightarrow \text{int}\}$ and rewrite rules $\mathcal{R} = \{\text{eval}_0(f) \rightarrow f(0), \text{const}_1(x) \rightarrow 1\}$. Then $\text{eval}_0(f) \approx 1$ is an equivalence in $(\Sigma, \mathcal{R})$. However, the reason is purely accidental: the only possible ground instance of $f :: \text{int} \rightarrow \text{int}$ is const_1 (LCSTRSs do not contain λ terms), and indeed $\text{eval}_0(\text{const}_1) \leftrightarrow_{\mathcal{R}}^* 1$. This equivalence does not scale, as we can find an extension $(\Sigma', \mathcal{R}')$ of $(\Sigma, \mathcal{R})$ and a ground instance $f := F$ for which $\text{eval}_0(F) \leftrightarrow_{\mathcal{R}'}^* 1$ does not hold. For instance, take $\Sigma' = \Sigma \cup \{\text{const}_0 :: \text{int} \rightarrow \text{int}\}$, $\mathcal{R}' = \mathcal{R} \cup \{\text{const}_0(x) \rightarrow 0\}$ and $F = \text{const}_0$.

Ideally, we like to exclude equivalences like $\text{eval}_0(f) \approx 1$. The issue here is that inductive theorems define a very local property, only considering ground convertibility in the given rewrite system $(\Sigma, \mathcal{R})$, while we actually want to quantify over all its program-like extensions (what is precisely meant by this will be explained in Chapter 3). The corresponding notion of inductive theorem we will define as *global inductive theorem*.

Extensionality. Another property not included by ground convertibility, specifically related to the higher-order behavior of LCSTRSs, is extensionality. In mathematical logic, extensionality is the principle $\forall f \forall g (\forall x (f(x) = g(x)) \implies f = g)$, i.e. declaring every two functions equivalent if they agree on all their inputs. A similar idea can also be applied to program equivalence. For instance, consider $\text{map} :: (\text{int} \rightarrow \text{int}) \rightarrow \text{list} \rightarrow \text{list}$, defined by the following rewrite rules (in applicative notation)

$$\text{map } f \text{ nil} \rightarrow \text{nil} \qquad \text{map } f (x : l) \rightarrow (f x) : (\text{map } f l)$$

In this system, we have an inductive theorem $\text{map } ([+] 0) l \approx \text{map } ([*] 1) l$, because $(\text{map } ([+] 0) l)\gamma \leftrightarrow_{\mathcal{R}}^* (\text{map } ([*] 1) l)\gamma$ for every ground substitution γ . Note that this does *not* hold for $\text{map } ([+] 0) \approx \text{map } ([*] 1)$, which is therefore not an inductive theorem. From an extensional point of view, however, it makes sense to also treat this last equation as an equivalence, because the terms on both sides behave similarly when viewing them as functions, i.e. $t \mapsto \text{map } ([+] 0) t$ and $t \mapsto ([*] 1) t$ represent the same function with respect to normalization behavior on ground terms t . This gap is exactly what extensional inductive theorems are meant to fill.

Extensional inductive theorems are more complicated to handle because, in their unrestricted form, they break monotonicity [10]: the property that if $s \approx t$ is an extensional inductive theorem, then also $C[s] \approx C[t]$ for every context $C[\square]$. This in turn complicates the development of a corresponding proof system, which is the main goal of objective 2. Therefore, in this thesis we adopt the first-order, non-extensional notion of inductive theorem to study program equivalence. Nevertheless, extensional inductive theorems remain desirable in the setting of higher-order program equivalence

and have been studied in existing literature [9, 10]. They naturally connect to our setting, and it is therefore our duty to study them.

Objective 2: develop a proof system for LCSTRSs

We extend the existing proof system Rewriting Induction (RI) for first-order constrained rewriting to LCSTRSs in three ways:

- *Basic version:* We define *Higher-Order RI (HORI)*, a version of RI for LCSTRSs, where we aim to stay close to the existing RI system for LCTRSs, both in the deduction rules and the corresponding soundness proof.
- *Extensible version:* We define *Global Rewriting Induction (Global RI)*, an adaptation of HORI that proves *global inductive theorems* – a generalization of inductive theorems that captures extensibility in program equivalence.
- *Version with less termination requirements:* We define *Bounded RI*, a version of RI that aims to reduce termination requirements in the induction-based equivalence proofs. Compared to HORI and Global RI, Bounded RI uses a more fine-grained methodology for constructing well-founded orderings. This requires a revision of the HORI system in order to facilitate additional bookkeeping on ordering requirements. The soundness proof of Bounded RI follows a different approach compared to HORI and Global RI. A beneficial side effect of this new proof strategy is increased modularity, which makes it easier to add new deduction rules.

Objectives 3 & 4: develop a method for disproving program equivalence & proving ground confluence

A substantial part of this thesis is centered around ground confluence. This is a relevant property because in a terminating and ground-confluent system every ground term reduces to a unique normal form, so reduction essentially computes a unique result for every ground term. Ground confluence also plays a crucial role in objective 3, as it is a necessary condition for extending RI to a proof system capable of disproving equations that are not inductive theorems. This motivates our interest in developing a method for establishing ground confluence in LCSTRSs.

Although confluent programs in programming languages may automatically translate to confluent systems, not every LCSTRS originates from a program: as a general rewriting formalism, LCSTRSs may also describe non-(ground-) confluent systems. Therefore, a general method for establishing ground confluence remains relevant.

The classical approach to proving confluence of a term rewriting system is via the analysis of critical pairs. This approach has been extended to LCTRSs [44], which we will

use as a basis for defining critical pairs in LCSTRSs. However, since general confluence is stronger than ground confluence, we ideally want to use an approach to analyzing critical pairs that is specifically aimed at proving ground confluence. For first-order TRSs, such an approach has been developed, making use of RI: ground confluence of a terminating rewrite system can be established by using RI to prove that all its critical pairs are bounded ground convertible [8]. The Bounded RI system that we will introduce allows us to extend this result to LCSTRSs.

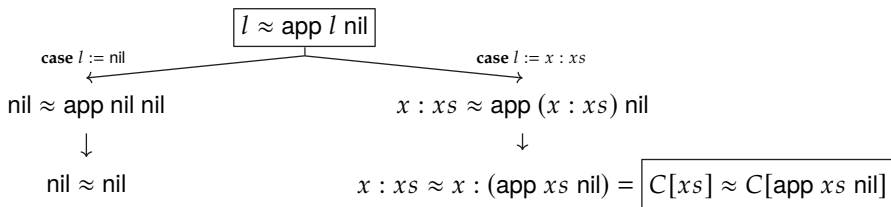
Objective 5: develop heuristics to facilitate semi-automatic, human-guided equivalence proofs

Automatically proving program equivalence in RI amounts to automatically performing induction proofs. Although we cannot expect to develop a fully automatic inductive theorem prover, we may still focus on the development of heuristics that can be used in practice as proof tactics to automatically perform non-trivial intermediate steps in human-guided proofs. In this thesis, we focus on the development of isolated heuristics, leaving their integration in a unified RI tool as future work.

A big challenge in induction-based proofs is that generating induction hypotheses often requires the generalization of equations. For example, consider $\text{rev } L \text{ nil} \approx \text{app } (\text{rev } L \text{ nil}) \text{ nil}$ in the system

$$\begin{array}{ll} \text{app nil } l \rightarrow l & \text{rev nil } l \rightarrow l \\ \text{app } (x : xs) l \rightarrow x : (\text{app } xs l) & \text{rev } (x : xs) l \rightarrow \text{rev } xs (x : l) \end{array}$$

To obtain a provable induction hypothesis, we generalize the subterm $\text{rev } L \text{ nil}$ on both sides of the equation to the variable l . This yields the lemma $l \approx \text{app } l \text{ nil}$ which we can use as an induction hypothesis. Lemmas are themselves inductive theorems, and therefore require a proof. Here, the corresponding induction procedure is represented by the following proof tree



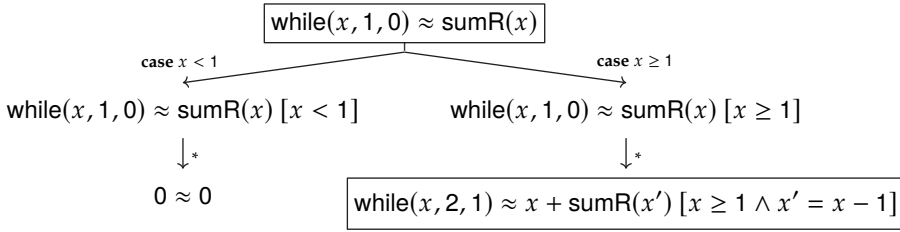
The proof obligation at the lower-right node is removed by applying the induction hypothesis at the top node under the substitution $l := xs$ inside the context $C[\Box] = x : \Box$.

Returning to the question of how to obtain lemmas automatically: in the above example the generalization step only involves the replacement of maximal subterms

that occur on both sides of the equation by a variable. For many practical examples, however, this approach is insufficient and therefore more advanced heuristics in the area of equational reasoning, designed for programs over purely inductive data structures (such as list and nat), have been developed [18, 31, 32, 45, 46].

Lemma generation in constrained rewriting For constrained rewriting, the lemma generation heuristics from equational reasoning are often insufficient. This is because, on top of the syntactical reasoning to generalize term shapes, we also require a semantic analysis, which involves the generation of invariants on theory terms.

To illustrate, consider $\text{sumL}(x) \approx \text{sumR}(x)$ which reduces to $\text{while}(x, 1, 0) \approx \text{sumR}(x)$. To prove this equation, we proceed by induction on x . In contrast to the reverse-list-example, the case analysis is not a structural analysis, but a semantic analysis based on the constraints of $\text{while}(x, i, z) \rightarrow z [i > x]$ and $\text{while}(x, i, z) \rightarrow \text{while}(x, i+1, z+i) [i \leq x]$. That is, we perform the case analysis $i > x$ and $i \leq x$. The corresponding induction procedure is displayed by the following proof tree:



The induction hypothesis at the top node is applied to the subterm $\text{sumR}(x')$ at the lower-right node, using substitution $x := x'$. This does not directly finish the proof but results in a new proof obligation $\text{while}(x, 2, 1) \approx x + \text{while}(x', 1, 0) [x \geq 1 \wedge x' = x - 1]$, which has no dependency on sumR anymore. However, the equation is not provable with straightforward induction. On the other hand, the following generalization is easily provable by induction:

$$\text{while}(x, i, z) \approx x + \text{while}(x', i', z') \quad [x \geq i' \wedge x' = x - 1 \wedge i' = i - 1 \wedge z = z' + i']$$

The difficulty in finding this generalization is not syntactic but rather semantic, requiring the *invariants* $i' = i - 1$, $x \geq i'$, $x' = x - 1$, $z = z' + i'$ to capture the relations between the integer variables in the two programs. More generally, a major challenge in inductive theorem proving for LCSTRSs is to find suitable lemma statements *automatically*. Adapting existing lemma-generation techniques to constrained rewriting does not seem to be straightforward [39], suggesting the need for different approaches. Initial attempts have been made to address this challenge [22, 39]. A central issue is the automatic generation of invariants. The *Initialization Generalization* (InGen) method [22] marks a

step in the right direction, being able to produce non-trivial invariants. Nevertheless, InGen remains limited in practice, as in many examples it only produces a fraction of the required invariants and may also produce non-invariants.

Proof heuristics for lemma generation This thesis aims to develop heuristics for automatic theorem proving in RI for constrained rewriting. We focus on two approaches:

- (1). *Invariant generation*: We introduce a method for generating equality-invariants on simple integer loops. Combined with SMT solving, we can also find inequality-invariants, to obtain a lemma generation heuristic for simple integer loops.
- (2). *Templates*: Lemma generation by invariant generation is not always practically feasible because equivalence proofs may require non-polynomial invariants, of which verification is typically not (efficiently) supported by SMT: a key ingredient for the implementation of rewrite tools for LCSTRSs.

Inspired by an existing method for program transformations [19], we consider an alternative higher-level proof heuristic, aimed at the generalization of programming constructs via *templates*. The idea is to capture commonly used programming constructs in templates (represented as term rewriting systems) that describe classes of programs. Established results between templates can then be instantiated to obtain lemmas for concrete programs. For example, $\text{while}(x, i, z) \rightarrow \text{while}(x, i + 1, z + i) [i \leq x]$ and $\text{while}(x, i, z) \rightarrow z [i > x]$ can be seen as an instance of a more general (higher-order) tail recursion template, represented by the following LCSTRS which defines a function symbol T

$$T f i y a \rightarrow T f (i + 1) y (f i a) [i \leq y] \qquad T f i y a \rightarrow a [i > y]$$

Similarly, $\text{sumR}(x)$ can be seen as an instance of a more general recursion template which can be represented in an LCSTRS, defining a function symbol R (for which we here omit the precise definition). By straightforward induction, we then prove $T f x y a \approx R f x y a$, which in turn is used as a lemma to prove $\text{while}(x, 1, 0) \approx \text{sumR}(x)$.

Scope and perspective

From the perspective of program verification, a very legitimate question to ask is *why* one would consider rewriting induction in the first place, as there already exists a wide range of methods for proving program equivalence, including general-purpose interactive theorem provers such as Lean and Rocq.

In Section 3.3.2 we included a short comparison of ground convertibility with contextual equivalence, but this only involves an informal, high-level discussion. An

extensive comparison with existing equivalence notions and corresponding proof methods is beyond the scope of this thesis: our central focus remains term rewriting, and the central investigation concerns the exploration of how this approach can be further developed beyond the classical setting of equational reasoning, towards a practical tool for proving (a specific notion of) program equivalence.

As mentioned, a particular benefit of the rewriting approach is that it offers a distinctive route towards automation of induction proofs, exploiting existing termination techniques from term rewriting. This becomes especially interesting when the full power of higher-order constrained rewriting can be utilized. We conjecture that techniques that build upon the rewriting approach (such as automatic lemma generation) may also be applicable to other forms of inductive theorem proving. Conversely, techniques developed for inductive theorem proving may provide useful insights for the further development of rewriting induction. These conjectures should, however, be taken with a grain of salt, given our limited knowledge of the state of the art in these research fields.

1.2 Overview and contributions

In Chapter 2, we provide the necessary preliminaries on *Logically Constrained Simply-typed Term Rewrite Systems* (LCSTRSs). Here, we slightly deviate from the original notation [25] in order to retain the RI notation of first-order RI [22] as much as possible.

The remaining chapters present the actual contributions of this thesis, which can roughly be divided into two categories:

- *Theoretical* (Chapter 3 and 5): Devoted to the design of the different RI systems.
- *Practical* (Chapter 4 and 6): Introducing heuristics for proving program equivalence.

Below, we describe in detail the content and contributions of each separate chapter.

Chapter 3: Higher-Order Program Equivalence. This chapter introduces two proof systems for RI in LCSTRSs. It is based on the following paper, published in the proceedings of *Logic-Based Program Synthesis and Transformation (LOPSTR)*:

- K. Hagens and C. Kop. “Rewriting Induction for Higher-Order Constrained Term Rewriting Systems”. In: *Proc. LOPSTR 24*. Vol. 14919. 2024, pp. 202–219. doi: 10.1007/978-3-031-71294-4_12.

Before introducing the two proof systems, we require some technical preparatory work:

- First, we aim to find a suitable definition of inductive theorem to interpret program equivalence. We consult existing literature [9, 10] on higher-order inductive theorems in relation to *extensionality*.

- In first-order constrained rewriting, soundness of RI is guaranteed by requiring *quasi-reductive* term rewriting systems [22]. We will generalize this notion to LCSTRSs by replacing constructor terms with *semi-constructor terms*.

We then introduce the two proof systems:

- *Higher-order Rewriting Induction*: this proof system is designed to closely mirror first-order RI for LCTRSs [22], both in its deduction rules and the soundness proof. This is facilitated by the introduction of semi-constructor terms.
 - Compared to the RI system for LCTRSs, we introduce two new deduction rules, called (Generalize) and (Alter). This allows us to reformulate the deduction rule (Simplify) without “constrained reductions”, thereby yielding a more transparent RI system.
- *Global Rewriting Induction*: an extensional adaptation of HORI that can prove global inductive theorems. The most important contributions include:
 - The introduction of *strong quasi-reductivity* and *natural extensions*, which are used to define global inductive theorems. Together, these two notions aim to rule out rewriting systems that correspond to ill-behaved program behavior.
 - The introduction of *oracle extensions*. Using global inductive theorems confronts us with the challenge of how to obtain a practically useful proof system: a naive approach will cause the termination requirements in the well-founded induction being quantified over *all* natural extensions of a given LCSTRS. This makes verification practically impossible. This problem is solved by showing that for proving global inductive theorems, it suffices to quantify over a more manageable class of extensions: oracle extensions.

Chapter 4: Lemma Generation. In this chapter we introduce *matrix invariants*, a method for invariant generation on simple integer loops. The method converts a diverging proof into a symbolic matrix, whose kernel (to be computed by a computer algebra system) yields a set of polynomial equality-invariants. Matrix invariants can be combined with the InGen method [22], to generate additional inequality-invariants to obtain a lemma generation heuristic. This chapter provides a detailed development and refinement of the ideas introduced in the following workshop paper, presented at *Workshop on Rewriting Techniques for Program Transformations and Evaluation (WPTE)*:

- K. Hagens and C. Kop. “Matrix invariants for program equivalence in LCTRSs”. In: *Proc. WPTE 23*. 2023, pp. 1–13. URL: <https://hdl.handle.net/2066/312790>

Its contributions include

Matrix invariants

- Introducing the notion *simple divergence pattern*, which intuitively represents the evolution of integer program states during the iterations of a “simple” loop.
- Introducing *divergence space*, which extracts a simple divergence pattern into a set of vectors over integer polynomials.
- Defining the notion of an (affine) *invariant*. This leads to a corresponding (affine) *divergence matrix*: a symbolic matrix, whose kernel is generated by a set of polynomial invariants.
- Defining heuristics for how to use matrix invariants to generate lemma candidates.

Candidates constructed from solely matrix invariants often do not constitute a lemma, because they only capture equality-invariants, while inequality-invariants are required as well. This is the topic of the second part of this chapter:

Inequality invariants

- We identify a *divergence constraint* φ_{div} , which constitutes an essential ingredient for generating inequality-invariants.
- We show how to use φ_{div} to generate inequality invariants, similar to those generated by the InGen method.
- We show how to obtain additional *derived invariants*, which in practice may describe the necessary remaining inequalities for obtaining a lemma.

Chapter 5: Bounded Rewriting Induction. This chapter introduces *Bounded Rewriting Induction* (*Bounded RI*): a variant of HORI, employing a more fine-grained method for well-founded induction. This allows us to reduce termination requirements in RI proofs. Also, it allows us to use RI as a tool for proving ground confluence. The chapter is based on the following paper, which has been submitted to the journal *Logical Methods in Computer Science* (LMCS):

- K. Hagens and C. Kop. *Bounded Rewriting Induction for LCSTRSs*. 2026. doi: 10.48550/arXiv.2601.02803

Its contributions are categorized into two parts and discussed below.

Reducing termination requirements In the classical RI approach, also employed by HORI, every induction hypothesis $s \approx t [\varphi]$ must be oriented: either $s > t [\varphi]$ (i.e. $\mathcal{R} \cup \{s \rightarrow t [\varphi]\}$ should be terminating) or $t > s [\varphi]$ (i.e. $\mathcal{R} \cup \{t \rightarrow s [\varphi]\}$ should be terminating). It is well-known [43] that this methodology often produces stronger termination requirements than necessary and attempts have been made to address this issue [4–6]. In Bounded RI we will continue on this: by adopting a more clever induction approach, which is not based on orienting induction hypotheses, we can reduce the termination requirements. This is achieved through the following contributions:

- We introduce the notion of *bounding pair* $(>, \geq)$ replacing the well-founded ordering $> = \rightarrow_{\mathcal{R}} \cup \triangleright$. For some ordering requirements, we can use the less strict $\geq$.
- We replace equations by *equation contexts*, containing the additional information of two *bounding terms* which define an upper bound for applying an induction hypothesis. Bounding terms serve as a bookkeeping device, tracking the terms to be used for generating the ordering requirements.
- We show that Bounded RI can be used to prove that equations are *bounded* ground convertible, which is strictly stronger than ground convertibility, and explicitly relates to a bounding pair $(>, \geq)$.
- We structure the soundness proof in a very modular way, and extract two invariants (properties) that all but one of the deduction rules satisfy. We show that every deduction rule that satisfies those two invariants is automatically sound. This makes it possible to easily add new derivation rules to the existing system without having to redo the proof.
- We show that RI can also be used for disproving an equation that is not an inductive theorem. We will extend Bounded RI with an additional deduction rule called (Disprove) to deduce non-equivalences. The (Disprove) rule can, however, only be used if $\mathcal{R}$ is ground confluent.

Ground confluence Another benefit of the altered induction approach in Bounded RI is that it can be used for proving ground confluence of terminating rewrite systems. This relies on the fact that a terminating system is ground confluent if all its critical pairs are bounded ground convertible. A similar result has already been established for first-order TRSs [8], although using a slightly different notion of bounded ground convertibility than the one used in this thesis. Essentially, we show that Bounded RI allows us to extend this first-order result to LCSTRSs. The contributions include:

- We define *critical pairs* and *critical peaks* for LCSTRSs, building on the definition of first-order critical pairs for LCTRSs [44]. To preserve its specific position-dependent

formulation, we define a corresponding notion of positions, taking care of partial function application (a problem not present in LCTRSs).

- We prove the *critical peak lemma*, expressing that the only non-trivial sources of local non-confluence originate from critical peaks.
- We prove the *ground critical peak theorem*, which states that a terminating LCSTRS is ground confluent whenever all its critical pairs are bounded ground convertible. This immediately gives us the desired result, as Bounded RI is designed to prove bounded ground convertibility.

Chapter 6: Program Equivalence by Templates. In this chapter we introduce a template-based method for proving bounded inductive theorems. This is motivated by the fact that the heuristics from Chapter 4 are not always effective for deriving lemmas. A major problem is that equivalences may require non-polynomial invariants, such as $y = \prod_{i=0}^n (x - i)$, which are not well supported within the SMT-based implementations of RI. Inspired by an existing template-based method for program transformations [19], we introduce an alternative generalization technique.

Templates. A template is a general rewriting scheme that aims to capture all instances of a particular programming construction, such as a tail recursive procedure. This can be employed as a lemma generation strategy to obtain a more high-level approach, complementary to the low-level generalization method in Chapter 4. A template is represented by a corresponding recursor (an LCSTRS), and whenever a program is an instance of a recursor, we may apply a *recursor equivalence* as a transformation step (essentially serving as an induction hypothesis) to simplify a particular proof goal.

This chapter is based on the following paper, published in the proceedings of *Logical and Semantic Frameworks, with Applications (LSFA)*:

- K. Hagens and C. Kop. “Templates in Rewriting Induction”. In: *Proc. LSFA 26*. Vol. 449. EPTCS. 2026, pp. 123–140. doi: 10.4204/EPTCS.449.8

The contributions include:

- Explaining that the template method provides a high-level lemma generation method which can be integrated in Bounded RI, as a complementary method to invariant generation.
- Introducing two templates for describing tail recursion, and two templates for describing non-tail recursion. We demonstrate how to match each of these templates with a concrete example.
- Establishing several template equivalences and demonstrating how to apply this in practice for proving program equivalence. We distinguish two proof tactics:

- *One-sided matching*, transforming a proof goal into an easier one.
- *Two-sided matching*, concluding equivalence directly.

Software This thesis is accompanied by software and scripts to support the theoretical contributions presented in Chapter 4, 5, and 6. The roles in the development of this software and these scripts are as follows:

- The rewriting tool Cora, used in Chapter 5 and 6, was implemented by Cynthia Kop, who also maintains and further develops the tool on GitHub (see <https://github.com/hezzel/cora> for the latest version). I did not contribute to the development of Cora.
- The GNU Octave and Python scripts for generating symbolic-integer invariants, used in Chapter 4, were developed entirely by me.

Chapter 2

Preliminaries

LCSTRSs are a higher-order rewriting formalism with built-in support for integers and booleans (or in fact any arbitrary theory such as bitvectors, floating point numbers or integer arrays) as well as logical constraints to model control flow. We will recap LCSTRSs [25], a higher-order extension of LCTRSs [22]. This considers *applicative* higher-order term rewriting (without λ -abstractions) and *first-order* constraints.

Types Assume given a set of sorts (base types) $\mathcal{S}$; the set $\mathcal{T}$ of types is defined by the grammar $\mathcal{T} ::= \mathcal{S} \mid \mathcal{T} \rightarrow \mathcal{T}$. Here, $\rightarrow$ is right-associative, so all types may be written as $type_1 \rightarrow \dots \rightarrow type_m \rightarrow sort$ with $m \geq 0$.

We also assume given a subset $\mathcal{S}_{theory} \subseteq \mathcal{S}$ of *theory sorts* (e.g., int and bool), and define the *theory types* by the grammar $\mathcal{T}_{theory} ::= \mathcal{S}_{theory} \mid \mathcal{S}_{theory} \rightarrow \mathcal{T}_{theory}$. Each theory sort $\iota \in \mathcal{S}_{theory}$ is associated with a non-empty interpretation set $\mathcal{I}_\iota$ (e.g., $\mathcal{I}_{int} = \mathbb{Z}$, the set of all integers). We define $\mathcal{I}_{\iota \rightarrow \sigma}$ as set of functions from $\mathcal{I}_\iota$ to $\mathcal{I}_\sigma$.

Terms We assume a signature Σ of *function symbols* and a disjoint set $\mathcal{V}$ of variables, and a function *typeof* from $\Sigma \cup \mathcal{V}$ to $\mathcal{T}$; we require that there are infinitely many variables of all types. The set of terms $T(\Sigma, \mathcal{V})$ over Σ and $\mathcal{V}$ are the expressions in $\mathbb{T}$ – defined by the grammar $\mathbb{T} ::= \Sigma \mid \mathcal{V} \mid \mathbb{T} \mathbb{T}$ – that are *well-typed*: if $s :: \sigma \rightarrow \tau$ and $t :: \sigma$ then $s \ t :: \tau$, and $a :: typeof(a)$ for $a \in \Sigma \cup \mathcal{V}$. Application is left-associative, which allows all terms to be written in a form $a \ t_1 \dots t_n$ with $a \in \Sigma \cup \mathcal{V}$ and $n \geq 0$. Writing $t = a \ t_1 \dots t_n$, we define $head(t) = a$. For a term t , let $Var(t)$ be the set of variables occurring in t . A term t is *ground* if $Var(t) = \emptyset$, hence the set of ground terms equals $T(\Sigma, \emptyset)$. It is *linear* if no variable occurs more than once in t .

We assume that Σ is the disjoint union $\Sigma_{theory} \uplus \Sigma_{terms}$, where $typeof(f) \in \mathcal{T}_{theory}$ for all $f \in \Sigma_{theory}$. Each $f \in \Sigma_{theory}$ has an interpretation $\llbracket f \rrbracket \in \mathcal{I}_{typeof(f)}$ (for example, a theory symbol $* :: int \rightarrow int \rightarrow int$ may be interpreted as multiplication on $\mathbb{Z}$). We use $[f]$ for prefix or partially applied notation (e.g., $[+] \ x \ y$ and $x + y$ are the same). Values are theory symbols of base type, i.e. $\mathcal{Val} = \{v \in \Sigma_{theory} \mid typeof(v) \in \mathcal{S}_{theory}\}$. We assume

there is exactly one value for each element of $\mathcal{I}_t$ ($t \in \mathcal{S}_{theory}$). Elements of $T(\Sigma_{theory}, \mathcal{V})$ are called *theory terms*. For *ground* theory terms, we define $\llbracket s \ t \rrbracket = \llbracket s \rrbracket(\llbracket t \rrbracket)$, thus mapping each ground term of type σ to an element of $\mathcal{I}_\sigma$.

We fix a theory sort *bool* with $\mathcal{I}_{bool} = \{\top, \perp\}$. A *constraint* is a theory term s of type *bool*, such that $typeof(x) \in \mathcal{S}_{theory}$ for all $x \in \text{Var}(s)$.

Example 2.0.1. In this thesis, we will often use examples with $\mathcal{S}_{theory} = \{\text{int}, \text{bool}\}$ and $\Sigma_{theory} = \{+, -, *, <, \leq, >, \geq, =, \wedge, \vee, \neg, \text{true}, \text{false}\} \cup \{n \mid n \in \mathbb{Z}\}$, with $+, -, * :: \text{int} \rightarrow \text{int} \rightarrow \text{int}$, $<, \leq, >, \geq, = :: \text{int} \rightarrow \text{int} \rightarrow \text{bool}$, $\wedge, \vee :: \text{bool} \rightarrow \text{bool} \rightarrow \text{bool}$, $\neg :: \text{bool} \rightarrow \text{bool}$, $\text{true}, \text{false} :: \text{bool}$ and $n :: \text{int}$. We always let $\mathcal{I}_{\text{int}} = \mathbb{Z}$, $\mathcal{I}_{\text{bool}} = \{\top, \perp\}$ and interpret all symbols as expected. The values are *true*, *false* and all n . Theory terms are for instance $x + 3$, *true* and $7 * 0$. The latter two are ground. We have $\llbracket 7 * 0 \rrbracket = 0$. The theory term $x > 0$ is a constraint, but the term $(f \ x) > 0$ with $f \in \mathcal{V}$ of type $\text{int} \rightarrow \text{int}$ is not (since $typeof(f) \notin \mathcal{S}_{theory}$), nor is $[>] \ 0 :: \text{int} \rightarrow \text{bool}$ (since constraints must have type *bool*).

Substitutions A substitution is a type-preserving mapping $\gamma : \mathcal{V} \rightarrow T(\Sigma, \mathcal{V})$. We define the domain of a substitution as $\text{dom}(\gamma) = \{x \in \mathcal{V} \mid \gamma(x) \neq x\}$, and its image as $\text{im}(\gamma) = \{\gamma(x) \mid x \in \text{dom}(\gamma)\}$. A substitution on a finite domain $\{x_1, \dots, x_n\}$ is often denoted $[x_1 := s_1, \dots, x_n := s_n]$. A substitution γ is extended to a function $s \mapsto s\gamma$ on terms by placewise substituting variables in the term by their image: (i) $t\gamma = t$ if $t \in \Sigma$, (ii) $t\gamma = \gamma(t)$ if $t \in \mathcal{V}$, and (iii) $(t_0 \ t_1)\gamma = (t_0\gamma) \ (t_1\gamma)$.

For $M \subseteq T(\Sigma, \mathcal{V})$, we define $\gamma(M) = \{t\gamma \mid t \in M\}$. A *ground substitution* is a substitution γ such that $\text{im}(\gamma) \subseteq T(\Sigma, \emptyset)$. A *unifier* of terms s, t is a substitution γ such that $s\gamma = t\gamma$; a *most general unifier* or *mgu* is a unifier γ such that all other unifiers are instances of γ . For unifiable terms, an mgu always exists. We say that a substitution γ *respects* a constraint φ if $\gamma(\text{Var}(\varphi)) \subseteq \mathcal{Val}$ and $\llbracket \varphi\gamma \rrbracket = \top$. We say that a constraint φ is *satisfiable* if there exists a substitution γ that respects φ . It is *valid* if $\llbracket \varphi\gamma \rrbracket = \top$ for all ground substitutions γ such that $\gamma(\text{Var}(\varphi)) \subseteq \mathcal{Val}$.

Contexts and subterms Let $\square_1, \dots, \square_n$ be fresh, typed constants ($n \geq 1$). A context $C[\square_1, \dots, \square_n]$ (or just: C) is a term in $T(\Sigma \cup \{\square_1, \dots, \square_n\}, \mathcal{V})$ in which each $\square_i$ occurs exactly once. (Note that $\square_i$ may occur at the head of an application.) The term obtained from C by replacing each $\square_i$ by a term t_i of the same type is denoted by $C[t_1, \dots, t_n]$.

We say that t is a *subterm* of s , notation $s \geq t$, if there is a context $C[\square]$ such that $s = C[t]$. We say that t is a *strict subterm* of s , notation $s \triangleright t$, if $s \geq t$ and $s \neq t$.

Positions For a term $t = a \ t_1 \cdots t_n$ with $a \in \Sigma \cup \mathcal{V}$ and $n \geq 0$, the set of positions $\text{Pos}(t)$ is defined as $\text{Pos}(t) = (\bigcup_{i=0}^n \{\star i\}) \cup (\bigcup_{i=1}^n \{i \cdot p \mid p \in \text{Pos}(t_i)\})$. Define the subterm $t|_p$ of t at position $p \in \text{Pos}(t)$ as follows:

$$(i). (a \ t_1 \cdots t_n)|_{\star i} = (a \ t_1 \cdots t_{n-i})$$

$$(ii). (a \ t_1 \cdots t_n)|_{i \cdot p} = t_i|_p$$

If t is a term, $p \in \text{Pos}(t)$, and s is a term of the same type as $t|_p$, then let $t[s]_p$ be the term obtained from t by replacing $t|_p$ by s . We may denote the position $\star 0$ as ϵ .

For example, for $t = (+ \ (+ \ 1 \ 2) \ 3)$ we have the following subterms of t and corresponding positions

Position p	Subterm $t _p$	Position p	Subterm $t _p$	Position p	Subterm $t _p$
$\star 0$	$(+ \ (+ \ 1 \ 2) \ 3)$	$1 \cdot \star 0$	$(+ \ 1 \ 2)$	$1 \cdot 1 \cdot \star 0$	1
$\star 1$	$(+ \ (+ \ 1 \ 2))$	$1 \cdot \star 1$	$(+ \ 1)$	$1 \cdot 2 \cdot \star 0$	2
$\star 2$	$+$	$1 \cdot \star 2$	$+$	$2 \cdot \star 0$	3

We define the subset of *full positions* $\text{Pos}_F(t) \subseteq \text{Pos}(t)$ as those positions that end on $\star 0$

$$\text{Pos}_F(t) = \{p \in \text{Pos}(t) \mid p = j_1 \cdots j_n \cdot \star 0\}$$

Rewrite rules A rule is an expression $\ell \rightarrow r[\varphi]$. Here ℓ and r are terms of the same type, ℓ has a form $f \ \ell_1 \cdots \ell_k$ with $f \in \Sigma$ and $k \geq 0$, φ is a constraint and $\text{Var}(r) \subseteq \text{Var}(\ell) \cup \text{Var}(\varphi)$ ¹. If $\varphi = \text{true}$, we may denote the rule as just $\ell \rightarrow r$.

Fixing a signature Σ , we define the set of *calculation rules* as follows

$$\mathcal{R}_{\text{calc}} = \left\{ f \ x_1 \cdots x_m \rightarrow y \ [y = f \ x_1 \cdots x_m] \ \middle| \ \begin{array}{l} f \in \Sigma_{\text{theory}} \setminus \mathcal{V}al, \text{ all } x_i \text{ and } y \in \mathcal{V}, \\ \text{typeof}(f) = \iota_1 \rightarrow \dots \rightarrow \iota_m \rightarrow \kappa \end{array} \right\}$$

We furthermore assume a set of rules $\mathcal{R}$ such that ℓ is not a theory term, for all $\ell \rightarrow r[\varphi] \in \mathcal{R}$ (rules with ℓ a theory are contained in $\mathcal{R}_{\text{calc}}$).

Given Σ and $\mathcal{R}$ as above, we define three classes of function symbols:

- The *defined symbols* $\mathcal{D} = \{f \in \Sigma \mid \text{there is a rule } f \ \ell_1 \cdots \ell_k \rightarrow r[\varphi] \in \mathcal{R}\}$
- The *constructors* $\mathcal{C} = \mathcal{V}al \cup (\Sigma_{\text{terms}} \setminus \mathcal{D})$
- The *calculation symbols* $\Sigma_{\text{calc}} = \Sigma_{\text{theory}} \setminus \mathcal{V}al$

For $f \in \Sigma$, we define $\mathcal{R}_f = \{\ell \rightarrow r[\varphi] \in \mathcal{R} \mid \text{head}(\ell) = f\}$. In particular, $\mathcal{R}_f = \emptyset$ if $f \notin \mathcal{D}$.

Remark 2.0.2. Note that it is theoretically possible for a rule to have a calculation symbol at the head of its left-hand side, so a function symbol *can* be both a calculation symbol and a defined symbol. However, this would rarely occur in practice. It is allowed because methods like Rewriting Induction might create “rules” (in fact oriented equations) of unusual forms to be used in a termination proof.

¹Imposing this requirement avoids the need to define $\text{LVar}(\ell \rightarrow r[\varphi]) = \text{Var}(\varphi) \cup (\text{Var}(r) \setminus \text{Var}(\ell))$ (being used in e.g. in [22, 27]) because $\text{Var}(r) \subseteq \text{Var}(\ell) \cup \text{Var}(\varphi)$ implies $\text{LVar}(\ell \rightarrow r[\varphi]) = \text{Var}(\varphi)$.

Reduction relation For a fixed signature Σ and set of rules $\mathcal{R}$ as discussed above, the reduction relation $\rightarrow_{\mathcal{R}}$ is defined by

$$C[l\gamma] \rightarrow_{\mathcal{R}} C[r\gamma] \text{ if } \ell \rightarrow r [\varphi] \in \mathcal{R} \cup \mathcal{R}_{calc} \text{ and } \gamma \text{ respects } \varphi$$

Note that by definition of a context, reduction at the head of an application is allowed. For example, if $\text{append nil} \rightarrow \text{id} \in \mathcal{R}$, then we could reduce $\text{append nil } s \rightarrow_{\mathcal{R}} \text{id } s$. In case we want to emphasize that reduction $s \rightarrow_{\mathcal{R}} t$ is performed with a rule in $\mathcal{R}_{calc}$, we write $s \rightarrow_{\mathcal{R}_{calc}} t$ instead. We say that s has normal form t if $s \rightarrow_{\mathcal{R}}^* t$ and t cannot be reduced.

LCSTRS A *Logically Constrained Simply-typed Term Rewriting System (LCSTRS)* is a pair $(T(\Sigma, \mathcal{V}), \rightarrow_{\mathcal{R}})$ generated by $(\mathcal{S}, \mathcal{S}_{theory}, \Sigma_{terms}, \Sigma_{theory}, \mathcal{V}, \text{typeof}, \mathcal{I}, \llbracket \cdot \rrbracket, \mathcal{R})$. We often refer to an LCSTRS by $\mathcal{L} = (\Sigma, \mathcal{R})$, or just $\mathcal{R}$, leaving the rest implicit.

Example 2.0.3. Consider the following Haskell definition of `sumfun` which computes the function $(f, x) \mapsto \sum_{i=0}^x f(i)$ for all $x \geq 0$.

```

sumfun :: (Int -> Int) -> Int -> Int
sumfun f x
  | x <= 0      = f x
  | otherwise   = (f x) + (sumfun f (x - 1))

```

This program can be modeled as an LCSTRS: take $\mathcal{S} = \mathcal{S}_{theory} = \{\text{int}, \text{bool}\}$, $\Sigma_{terms} = \{\text{sumfun} :: (\text{int} \rightarrow \text{int}) \rightarrow \text{int} \rightarrow \text{int}\}$ and Σ_{theory} as in Example 2.0.1. We let

$$\mathcal{R} = \left\{ \begin{array}{ll} \text{sumfun } f \ x \rightarrow f \ x & [x \leq 0] \\ \text{sumfun } f \ x \rightarrow [+] (f \ x) (\text{sumfun } f \ (x - 1)) & [x > 0] \end{array} \right\}$$

Then $\mathcal{D} = \{\text{sumfun}\}$, and $\mathcal{C} = \mathcal{V}al = \{\text{true}, \text{false}\} \cup \{n \mid n \in \mathbb{Z}\}$. An example of a rewrite sequence, computing the normal form 2 is given by

$$\begin{aligned}
\text{sumfun } ([*] \ 2) \ 1 &\rightarrow_{\mathcal{R}} [+] ([*] \ 2) \ 1 \ (\text{sumfun } ([*] \ 2) \ (1 - 1)) \\
&\xrightarrow{\mathcal{R}_{calc}} [+] \ 2 \ (\text{sumfun } ([*] \ 2) \ (1 - 1)) \xrightarrow{\mathcal{R}_{calc}} [+] \ 2 \ (\text{sumfun } ([*] \ 2) \ 0) \\
&\rightarrow_{\mathcal{R}} [+] \ 2 \ (([*] \ 2) \ 0) \xrightarrow{\mathcal{R}_{calc}} [+] \ 2 \ 0 \xrightarrow{\mathcal{R}_{calc}} 2
\end{aligned}$$

We consider several properties an LCSTRS $\mathcal{L} = (\Sigma, \mathcal{R})$ can exhibit:

- ▷ $\mathcal{L}$ is *terminating* if there is no infinite reduction sequence $s_0 \rightarrow_{\mathcal{R}} s_1 \rightarrow_{\mathcal{R}} s_2 \rightarrow_{\mathcal{R}} \dots$ for any $s_0 \in T(\Sigma, \mathcal{V})$.
- ▷ $\mathcal{L}$ is *weakly normalising* if every term has at least one normal form.
Note that termination implies weak normalisation, but not the other way around.
- ▷ $\mathcal{L}$ is *confluent* if for all $s, t, q \in T(\Sigma, \mathcal{V})$: if $s \rightarrow_{\mathcal{R}}^* t$ and $s \rightarrow_{\mathcal{R}}^* q$ then there is some $w \in T(\Sigma, \mathcal{V})$ such that $t \rightarrow_{\mathcal{R}}^* w$ and $q \rightarrow_{\mathcal{R}}^* w$.

- ▷ $\mathcal{L}$ is *ground confluent* if it is confluent on ground terms: for all $s, t, q \in T(\Sigma, \emptyset)$: if $s \rightarrow_{\mathcal{R}}^* t$ and $s \rightarrow_{\mathcal{R}}^* q$ then there is some $w \in T(\Sigma, \emptyset)$ such that $t \rightarrow_{\mathcal{R}}^* w$ and $q \rightarrow_{\mathcal{R}}^* w$.
- ▷ $\mathcal{L}$ has *unique normal forms* if for any term s there is at most one term t in normal form so that $s \rightarrow_{\mathcal{R}}^* t$.

Note that confluence implies having unique normal forms, but not the other way around since some terms might not have any normal form.

Viewing LCSTRSs as programs, uniqueness of *ground* normal forms – which is implied by ground confluence – essentially expresses that output is produced deterministically. If, moreover, the LCSTRS is weakly normalising, every term computes a unique result.

Chapter 3

Higher-Order Program Equivalence

In term rewriting, program equivalence is represented by inductive theorems. For first-order rewriting there is a well-established notion of inductive theorem, as well as a corresponding proof system called Rewriting Induction (RI). In higher-order rewriting, however, even finding a suitable definition of inductive theorem is less straightforward, as higher-order behavior introduces additional aspects that should be taken into account. This already received attention in the literature [9, 10], resulting in several notions for higher-order inductive theorems. However, none of these is accompanied by a corresponding proof system: we are not aware of any proof system that can prove a particular definition of higher-order inductive theorems. Here, we aim to fill this gap, developing two LCSTRS-based equivalence notions and corresponding proof systems:

- (1). *Higher-Order Rewriting Induction (HORI)*: introducing a basic version of RI for proving *higher-order inductive theorems* in LCTRSs. Here, our aim is to stay close to the methodology from first-order RI [22].
- (2). *Global Rewriting Induction (Global RI)*: while HORI can only prove inductive theorems (i.e., ground convertibility in a *fixed* rewrite system), Global RI aims to provide an *extensible* proof system for establishing *global inductive theorems*, which concern inductive theorems that remain valid in every program-like extension of the given rewrite system. Since every global inductive theorem is also an inductive theorem, Global RI is strictly stronger than HORI. However, proving global inductive theorems is more demanding, which is reflected in the stronger conditions required by Global RI.

The content in this chapter is structured as follows

- In Section 3.1 we provide the necessary preliminaries. We introduce equations, and briefly recap first-order RI for LCTRSs [22], highlighting the role of *quasi-reductivity*. We discuss the possibility of extensional inductive theorems, before introducing

higher-order inductive theorems. To facilitate a corresponding RI system we generalize quasi-reductivity to the higher-order setting.

- In Section 3.2, we define HORI and state the soundness theorem. We explain the HORI deduction rules using an illustrative running example. We conclude with a proof of the soundness theorem.
- In Section 3.3 we will define Global RI. This requires some preparatory work:
 - We narrow attention to *strong quasi-reductive* LCSTRSs, which is a strictly stronger property than quasi-reductivity and is meant to exclude a class of rewrite systems that correspond to ill-behaved program behavior.
 - Under the assumption of strong quasi-reductivity, extensibility is expressed via the quantification over all *natural extensions*. Correspondingly, we define *global inductive theorems*: ground convertibility in every natural extension.
 - With these ingredients, we are almost ready to define Global RI as a straightforward adaptation of HORI. The remaining challenge is the (Expansion) deduction rule: a naive formulation would require proving termination of induction rules in all natural extensions, which is infeasible. We show that it is actually sufficient to only require termination for all *oracle extensions*.

We then finally define Global RI and conclude by proving its soundness.

This chapter is centered around a running example which considers the following two implementations of the function `sumfun :: (Int -> Int) -> Int -> Int`

$$\begin{array}{ccc}
 \left\| \begin{array}{l} \text{sumfun } f \ x \\ | \ x \leq 0 \quad = f \ x \\ | \text{otherwise} = (f \ x) + (\text{sumfun } f \ (x - 1)) \end{array} \right\| & & \left\| \begin{array}{l} \text{sumfun } f \ x = \\ \text{foldl } (+) \ 0 \ (\text{map } f \ [x, x-1..0]) \end{array} \right\|
 \end{array}$$

(a)
(b)

Fig. 3.1 Two equivalent implementations of `sumfun`, provided $x \geq 0$.

Both compute the function $(f, x) \mapsto \sum_{i=0}^x f(i)$, provided $x \geq 0$. The program in Figure 3.1a is represented by the term `sumfun f x` in the LCSTRS below

$$\mathcal{R} = \left\{ \begin{array}{ll} \text{(R1). } \text{sumfun } f \ x \rightarrow f \ x & [x \leq 0] \\ \text{(R2). } \text{sumfun } f \ x \rightarrow [+] (f \ x) (\text{sumfun } f \ (x - 1)) & [x > 0] \end{array} \right\}$$

and Figure 3.1b is represented by the term $\text{fold } [+] \ 0 \ (\text{map } f \ (\text{init } x))$ in the LCSTRS below

$$\mathcal{R} = \left\{ \begin{array}{ll} \text{(R3). } \text{fold } g \ v \ \text{nil} \rightarrow v & \text{(R6). } \text{map } f \ \text{nil} \rightarrow \text{nil} \\ \text{(R4). } \text{fold } g \ v \ (h : t) \rightarrow \text{fold } g \ (g \ v \ h) \ t & \text{(R7). } \text{map } f \ (h : t) \rightarrow (f \ h) : \text{map } f \ t \\ \text{(R5). } \text{init } n \rightarrow \text{nil} \quad [n < 0] & \text{(R8). } \text{init } n \rightarrow n : \text{init } (n - 1) \quad [n \geq 0] \end{array} \right\}$$

3.1 Rewriting Induction

To express that the terms $\text{sumfun } f \ n$ and $\text{fold } [+] \ 0 \ (\text{map } f \ (\text{init } n)) \ [n \geq 0]$ exhibit the same input-output behavior, we use equations as a formal notation.

Equations An *equation* is an expression of the shape $s \approx t \ [\varphi]$ with $\text{typeof}(s) = \text{typeof}(t)$ and φ a constraint. We say that a substitution γ respects $s \approx t \ [\varphi]$ if γ respects φ and $\text{Var}(s) \cup \text{Var}(t) \subseteq \text{dom}(\gamma)$.

Example 3.1.1. The comparison in Figure 3.1 is expressed by the following equation

$$\text{sumfun } f \ n \approx \text{fold } [+] \ 0 \ (\text{map } f \ (\text{init } n)) \quad [n \geq 0]$$

So even though, for example, the substitution $[n := 0]$ respects φ it does not respect the equation above, since it leaves the variable f undefined. If we extend the substitution to, e.g., $[n := 0, f := [*] \ 2]$ then it does respect this equation.

Now that we have equations, we define program equivalence via *inductive theorems*.

Inductive theorems An equation $s \approx t \ [\varphi]$ is an *inductive theorem* (a.k.a. *$\mathcal{R}$ -ground convertible*) if $s\gamma \leftrightarrow_{\mathcal{R}}^* t\gamma$ for every ground substitution γ that respects this equation. Here, $\leftrightarrow_{\mathcal{R}}$ is the union $\rightarrow_{\mathcal{R}} \cup \leftarrow_{\mathcal{R}}$ and is $\leftrightarrow_{\mathcal{R}}^*$ the transitive, reflexive closure of $\leftrightarrow_{\mathcal{R}}$.

3.1.1 RI for first-order rewriting

RI was introduced by Reddy [43] in the year 1990 as a deduction system for proving inductive theorems, using unconstrained first-order term rewriting systems. Since then, many first-order variations on the system have appeared [4–6, 21, 22, 43]. All are based on well-founded induction, using some well-founded relation $>$, but there are subtle variations in the precise definition of the deduction system.

Rewriting Induction RI is a deduction system on proof states, which are pairs of the shape $(\mathcal{E}, \mathcal{H})$. Intuitively, $\mathcal{E}$ is a set of equations containing all proof goals, and $\mathcal{H}$ is the set of induction hypotheses that have been assumed. At the start $\mathcal{E}$ consists of all

equations that we want to prove inductive theorems, and $\mathcal{H} = \emptyset$. Using a deduction rule, we may transform a proof state $(\mathcal{E}, \mathcal{H})$ into another proof state $(\mathcal{E}', \mathcal{H}')$. This is denoted as $(\mathcal{E}, \mathcal{H}) \vdash (\mathcal{E}', \mathcal{H}')$. We write $\vdash^*$ for the reflexive, transitive closure of $\vdash$.

Correctness of RI is typically expressed by the following statement:

Let $\mathcal{R}$ be a terminating, quasi-reductive LCTRS and $\mathcal{E}$ a set of equations. If $(\mathcal{E}, \emptyset) \vdash^* (\emptyset, \mathcal{H})$ for some $\mathcal{H}$, then every equation in $\mathcal{E}$ is an inductive theorem.

Intuitively, this expresses a soundness statement, and should be interpreted as “If we can remove every proof obligation (making $\mathcal{E}$ empty) by applying the RI deduction rules, then every equation in $\mathcal{E}$ is an inductive theorem.”. The quasi-reductive requirement that appears in this soundness statement will be explained in the next paragraph.

Remark 3.1.2. To avoid confusion we will clarify what we mean by soundness. Those familiar with equational reasoning may expect a statement expressing the implication

$$(\star) \quad \text{RI-provability} \implies \text{equality in every possible model of } \mathcal{R}$$

This is *not* how we will interpret soundness. In this thesis, we do not consider all possible models, but rather one particular model: $\mathcal{R}$ -ground convertibility. Thus, *soundness* should be interpreted as the following implication

$$\text{RI-provability} \implies \mathcal{R}\text{-ground convertibility}$$

Those interested in a more semantical discussion of RI – and in particular the implication $(\star)$ – may consider recent literature [7], which studies this question for LCTRSs.

For a set of equations $\mathcal{E}$, we introduce a relation $\leftrightarrow_{\mathcal{E}}$ that allows us to conclude equivalence between terms, using equations in $\mathcal{E}$. This is defined as follows

$$C[s\gamma] \leftrightarrow_{\mathcal{E}} C[t\gamma] \text{ if } s \approx t [\varphi] \in \mathcal{E} \text{ or } t \approx s [\varphi] \in \mathcal{E} \text{ and } \gamma \text{ respects } \varphi$$

We note that $\mathcal{E}$ consists of inductive theorems if and only if $\leftrightarrow_{\mathcal{E}} \subseteq \leftrightarrow_{\mathcal{R}}^*$ holds on ground terms (i.e. when restricting to ground substitutions γ and ground contexts C).

Quasi-reductivity Soundness of RI can only be proved under the assumption of *quasi-reductivity*. Intuitively, this property expresses that pattern matching on ground terms is exhaustive, i.e. that there are no missing reduction cases. For example, the rewrite system $\mathcal{R} = \{(\mathbf{R1}), (\mathbf{R2})\}$ is quasi-reductive because $x \leq 0$ and $x > 0$ together cover all ground instances of $\text{sumfun } f \ x$. But if we, for example, replace $(\mathbf{R1})$ by $\text{sumfun } f \ x \rightarrow f \ x \ [x < 0]$ then it is not, as we are missing all ground reductions with $x := 0$ (for example $\text{sumfun } ([+] \ 1) \ 0$ does not reduce anymore).

For first-order rewriting, quasi-reductivity is achieved by demanding that there are no other ground normal forms than the ground constructor terms $T(C, \emptyset)$. However, in the situation of higher-order LCSTRSs, this definition does not work as we can have ground normal forms with partially applied defined symbols, such as `sumfun ([+] 1)`. We will therefore (in Section 3.1.3) introduce a higher-order generalization of constructor terms, and the corresponding notion of quasi-reductivity.

First, we take a short intermezzo on a typical higher-order issue: extensionality.

3.1.2 Higher-Order Inductive Theorems

While the first-order definition can still be used in the higher-order setting, it is not immediately obvious if this is entirely appropriate. In particular, the question of *extensionality* comes into play. In mathematical logic, extensionality is the principle $\forall f \forall g (\forall x (f(x) = g(x)) \implies f = g)$, i.e. declaring every two functions equivalent if they agree on all their inputs. A similar idea can also be applied to program equivalence.

Extensional inductive theorems Extensional inductive theorems have been studied in existing literature [10]. For terms s, t of type $\sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota$ the equation $s \approx t$ is said to be an *extensional inductive theorem* if $(s \ x_1 \dots x_m) \gamma \leftrightarrow_{\mathcal{R}}^* (t \ x_1 \dots x_m) \gamma$ for all ground substitutions γ . So extensional inductive theorems are more liberal than ordinary inductive theorems: for example, $\text{map} ([+] 0) \approx \text{map} ([*] 1)$ is not an inductive theorem, but it is an extensional inductive theorem since $\text{map} ([+] 0) \ l \leftrightarrow_{\mathcal{R}}^* \text{map} ([*] 1) \ l$ for all ground terms l . Using extensional inductive theorems is intuitive, since functions are often viewed in an extensional way.

Unfortunately, as shown below in Example 3.1.3, it comes at a price, since this definition violates *monotonicity*: the property that if $s \approx t$ is an inductive theorem and C a context, then $C[s] \approx C[t]$ is also an inductive theorem.

Example 3.1.3. Let $\Sigma_{\text{terms}} = \{\text{add} :: \text{nat} \rightarrow \text{nat} \rightarrow \text{nat}, s :: \text{nat} \rightarrow \text{nat}, 0 :: \text{nat}, \text{fnil} :: \text{funclist}, \text{fcons} :: (\text{nat} \rightarrow \text{nat}) \rightarrow \text{funclist} \rightarrow \text{funclist}\}$, $\Sigma_{\text{theory}} = \emptyset$ and $\mathcal{R} = \{\text{add } x \ 0 \rightarrow x, \text{add } x \ (s \ y) \rightarrow s \ (\text{add } x \ y)\}$. Every ground term of type `nat` has a normal form of the shape $s^n 0$, for some $n \in \mathbb{N}$, so we can restrict to ground substitutions of the shape $\gamma_n = [x := s^n 0]$. Note that $(\text{add } (s \ 0) \ x) \gamma_n \xrightarrow{*}_{\mathcal{R}} s^{n+1} 0 \xleftarrow{*}_{\mathcal{R}} (s \ x) \gamma_n$. Hence, $\text{add } (s \ 0) \approx s$ is an extensional inductive theorem. However, for $C[\square] = \text{fcons } \square \ \text{fnil}$ we do *not* have $C[\text{add } (s \ 0)] \leftrightarrow_{\mathcal{R}}^* C[s]$.

This problem can be solved using *Higher-order Sufficient Completeness (HSC)* [10], at the cost of imposing restrictions on the rewrite systems under consideration. Aside from other consequences – which are similar to the ones we will consider in Section 3.3.2 – the restrictions essentially block constructors from taking higher-order arguments, thus for

instance disallowing lists of functions as used in Example 3.1.3. Since such constructs naturally occur in functional programs, we consider this restriction too severe, and have elected not to go into the extensional direction. Instead, we keep the first-order definition, which also makes sense in the higher-order setting and does satisfy monotonicity.

Definition 3.1.4 (Higher-order inductive theorems). An equation $s \approx t [\varphi]$ is a higher-order inductive theorem of an LCSTRS with rules $\mathcal{R}$ if $s\gamma \leftrightarrow_{\mathcal{R}}^* t\gamma$ for every ground substitution γ that respects this equation.

Discussion The choice whether to consider extensionality ties in to a larger discussion on the semantics of equations and (constrained higher-order) rewriting. Traditionally (in unconstrained rewriting), rules are seen as *oriented equations*. Ground terms may be interpreted in a *model*, and an equation $s \approx t$ *holds* in a model if for all ground instances $s\gamma \approx t\gamma$ of the equation, the interpretation of $s\gamma$ and $t\gamma$ is the same. We say that $\mathcal{R} \models s \approx t$ if $s \approx t$ holds in every model for which the rules of $\mathcal{R}$ all hold. In such a semantics, a term of higher type would typically be mapped to a function, so it is natural to use an extensional perspective where two terms are equivalent if their functional input-output behavior is equivalent.

One can define such a semantics for first-order LCTRSs [7], and prove the correspondence between ground convertibility and semantic validity; i.e., an equation $s \approx t [\varphi]$ is an inductive theorem (referred to as “CE-valid”) if and only if $\mathcal{R} \models s \approx t [\varphi]$. However, in *higher-order* rewriting, this does not typically hold – even if we include abstraction and the η rule scheme.

Existing work has studied higher-order equivalence [9], where “extensional theorems” (for unconstrained higher-order rewriting) are defined as equivalence in a particular model. This semantic approach is shown to correspond to syntactic equivalence of ground instances of equations in an inference system. Moreover, the approach solves the monotonicity problem [10] as monotonicity is built in, but it loses the direct correspondence to the convertibility relation $\leftrightarrow_{\mathcal{R}}^*$.

We have elected not to follow this example because our primary application domain is not equational reasoning – where a semantic definition is the natural choice – but functional programming, where the syntactic notion of convertibility seems preferable. Definition 3.1.4 has the benefit of minimality: any equivalence relation that includes $\rightarrow_{\mathcal{R}}$ must include $\leftrightarrow_{\mathcal{R}}^*$. Hence, any higher-order inductive theorem is also an extensional theorem, and the method of rewriting induction defined in this chapter can be used to derive extensional theorems as well.

3.1.3 Higher-order quasi-reductivity

Quasi-reductivity is an essential component of first-order RI, since it allows us to reduce any ground term to a constructor term. Yet in higher-order rewriting this property is hard to obtain, since it is usually possible for ground terms of function type to not be constructor terms; e.g., the term $[+] 1$, or the term init from Example 3.1.1. And Example 3.1.3, which seems relatively innocuous, even admits base-type ground non-constructor normal forms (e.g., $\text{fcons}(\text{add}(\text{s } 0)) \text{fnil}$).

Thus, we will update the first-order definition to include partially applied function symbols, both at the root and below constructors. For the notion of “partially applied” to make sense, we impose a mild restriction on the LCSTRSs we consider:

Definition 3.1.5 (Rule Arity). Every $f \in \mathcal{D}$ with $f :: \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota$, $\iota \in \mathcal{S}$ has a *rule arity* $k = ar(f) \leq m$, meaning that every rule in $\mathcal{R}_f$ is of the shape $f l_1 \dots l_k \rightarrow r [\varphi]$.

That is, we for example forbid having both a rule $\text{add}(\text{s } x) y \rightarrow \text{s}(\text{add } x y)$ where add takes two arguments, and a rule $\text{add } 0 \rightarrow \text{id}$ where it takes only one. This is not a significant restriction because we can simply alter the second rule to $\text{add } 0 x \rightarrow \text{id } x$ without changing its meaning. With this restriction, a symbol is partially applied if it has fewer than $ar(f)$ arguments. This allows us to define

Definition 3.1.6 (Semi-constructor terms). Let $\mathcal{L} = (\Sigma, \mathcal{R})$ be an LCSTRS. The *semi-constructor terms* over $\mathcal{L}$, notation $\mathcal{SCT}_{\mathcal{L}}$, are defined by

- (i). $\mathcal{V} \subseteq \mathcal{SCT}_{\mathcal{L}}$
- (ii). If $f \in \Sigma$ with $f :: \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota$, $\iota \in \mathcal{S}$ and $s_1 :: \sigma_1, \dots, s_n :: \sigma_n \in \mathcal{SCT}_{\mathcal{L}}$ with $n \leq m$, then $f s_1 \dots s_n \in \mathcal{SCT}_{\mathcal{L}}$ if one of the following holds
 - (a). $f \in \mathcal{C}$
 - (b). $f \in \mathcal{D}$ and $n < ar(f)$
 - (c). $f \in \Sigma_{theory} \setminus \mathcal{D}$ and $n < m$

The set $\mathcal{SCT}_{\mathcal{L}}^{\emptyset}$ refers to *ground* semi-constructor terms, built without (i).

Note that all constructor terms are semi-constructor terms, by (i) and (ii.a). In our previous examples, $[+] 1$ and init and $\text{fcons}(\text{add}(\text{s } 0)) \text{fnil}$, are all (ground) semi-constructor terms as well. Non-examples of semi-constructor terms are for instance $[+] 0 y$ and $\text{fcons}(\text{add}(\text{add } 0 0))$. Using (ii.b) we easily obtain:

Lemma 3.1.7. Every semi-constructor term is a normal form.

Now, ground semi-constructor terms are the higher-order counterpart of ground constructor terms, as intuitively, in an LCSTRS without missing cases in the pattern matching, the only ground normal forms are in $\mathcal{SCT}_{\mathcal{L}}^{\emptyset}$.

Definition 3.1.8 (Quasi-reductivity). An LCSTRS $\mathcal{L} = (\Sigma, \mathcal{R})$ is quasi-reductive if for every $t \in T(\Sigma, \emptyset)$ we have $t \in \mathcal{SCT}_{\mathcal{L}}^{\emptyset}$ or t reduces with $\rightarrow_{\mathcal{R}}$.

Note that for quasi-reductive LCSTRSs, we can limit the substitutions in Definition 3.1.4 to substitutions such that $im(\gamma) \subseteq \mathcal{SCT}_{\mathcal{L}}^{\emptyset}$, without loss of generality. We call such a substitution a *ground semi-constructor substitution*, abbreviated to *gsc substitution*.

3.2 Higher-Order Rewriting Induction

In this section, we introduce Higher-Order Rewriting Induction (HORI). Soundness of HORI is established by Theorem 3.2.1.

Theorem 3.2.1 (Soundness of HORI). Let $\mathcal{L}$ be a terminating, quasi-reductive LCSTRS and let $\mathcal{E}$ be a set of equations. If, by higher-order rewriting induction, $(\mathcal{E}, \emptyset) \vdash^* (\emptyset, \mathcal{H})$, for some set $\mathcal{H}$, then every equation in $\mathcal{E}$ is a higher-order inductive theorem of $\mathcal{L}$.

The proof of this theorem, being presented in Section 3.2.3, follows the first-order approach [22]: it proceeds by deriving an invariant being maintained by every deduction rule in HORI, and which is used to construct an induction proof on the well-founded relation $\rightarrow_{\mathcal{R} \cup \mathcal{H}}$ to eventually conclude $\leftrightarrow_{\mathcal{E}} \subseteq \leftrightarrow_{\mathcal{R}}$.

3.2.1 Deduction rules

We will now continue by introducing the deduction rules for HORI, shown in Figure 3.2. Compared to the RI system for LCTRSs, we introduce two new deduction rules, called (Generalize) and (Alter). This allows us to formulate the (Simplify) rule without *constrained reductions* [22].

As a running example, we continue on the equation from Example 3.1.1. Following Theorem 3.2.1, we show that there is a set $\mathcal{H}$ such that $(\mathcal{E}_0, \emptyset) \vdash^* (\emptyset, \mathcal{H})$ with

$$\mathcal{E}_0 = \{(\mathbf{A}) \quad \text{sumfun } f \ n \approx \text{fold } [+] \ 0 \ (\text{map } f \ (\text{init } n)) \ [n \geq 0]\}$$

Rewriting equations

The core deduction rule facilitates the application of a rule in $\mathcal{R} \cup \mathcal{R}_{calc} \cup \mathcal{H}$ to reduce an equation. We view an equation with variables as a way to represent all its *ground semi-constructor instances*. Similar to “constrained term reduction” [35], we can reduce equations with rewrite rules, taking the constraint of the equation into account.

(Simplify) Let $\ell \rightarrow r [\varphi] \in \mathcal{R} \cup \mathcal{R}_{calc} \cup \mathcal{H}$ with C a context, δ a substitution such that $\delta(\text{Var}(\varphi)) \subseteq \mathcal{Val} \cup \text{Var}(\psi)^1$, and $C[\ell\delta] \approx t [\psi]$ an equation. If the implication $\psi \implies \varphi\delta$

¹Compared to [27], we have eliminated the use of $\text{LVar}(\ell \rightarrow r [\varphi]) = \text{Var}(\varphi) \cup (\text{Var}(r) \setminus \text{Var}(\ell))$. This is possible due to the assumption $\text{Var}(r) \subseteq \text{Var}(\ell) \cup \text{Var}(\varphi)$, which implies that $\text{LVar}(\ell \rightarrow r [\varphi]) = \text{Var}(\varphi)$.

(Simplify)

$$\frac{(\mathcal{E} \uplus \{C[\ell\delta] \approx t[\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \{C[r\delta] \approx t[\psi]\}, \mathcal{H})} \quad \begin{array}{l} \ell \rightarrow r[\varphi] \in \mathcal{R} \cup \mathcal{R}_{calc} \cup \mathcal{H} \text{ and} \\ \delta(\text{Var}(\varphi)) \subseteq \mathcal{V}al \cup \text{Var}(\psi) \text{ and} \\ \psi \Rightarrow \varphi\delta \text{ valid} \end{array}$$

(Expansion)

$$\frac{(\mathcal{E} \uplus \{s \approx t[\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \text{Expd}(s \approx t[\psi], p), \mathcal{H} \cup \{s \rightarrow t[\psi]\})} \quad \begin{array}{l} p \text{ expansion-position of } s \text{ and} \\ \mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t[\psi]\} \text{ terminating} \end{array}$$

(Expansion2)

$$\frac{(\mathcal{E} \uplus \{s \approx t[\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \text{Expd}(s \approx t[\psi], p), \mathcal{H})} \quad p \text{ expansion-position of } s$$

(Generalize)/(Alter)

$$\frac{(\mathcal{E} \uplus \{s \approx t[\varphi]\}, \mathcal{H})}{(\mathcal{E} \cup \{u \approx v[\psi]\}, \mathcal{H})} \quad u \approx v[\psi] \text{ generalizes/alters } s \approx t[\varphi]$$

(Delete)

$$\frac{(\mathcal{E} \uplus \{s \approx t[\varphi]\}, \mathcal{H})}{(\mathcal{E}, \mathcal{H})} \quad \varphi \text{ unsatisfiable, or } s = t$$

(Semi-constructor)

$$\frac{(\mathcal{E} \uplus \{f \vec{s} \approx f \vec{t}[\varphi]\}, \mathcal{H})}{(\mathcal{E} \cup \{s_i \approx t_i[\varphi] \mid 1 \leq i \leq n\}, \mathcal{H})} \quad \begin{array}{l} \vec{s} = s_1 \dots s_n \text{ and } \vec{t} = t_1 \dots t_n \text{ and either:} \\ (1). f \in \mathcal{V} \cup \mathcal{C} \\ (2). f \in \mathcal{D} \text{ with } ar(f) > n \\ (3). f \in \Sigma_{calc} \setminus \mathcal{D} \text{ and } typeof(f) = \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota \text{ and } m > n \end{array}$$

Fig. 3.2 Derivation rules for HORI.

is valid, then we can perform a (Simplify) step:

$$(\mathcal{E} \uplus \{C[\ell\delta] \approx t[\psi]\}, \mathcal{H}) \vdash (\mathcal{E} \cup \{C[r\delta] \approx t[\psi]\}, \mathcal{H})$$

There is an analogous (Simplify) rule to reduce the right-hand side of an equation.

Example 3.2.2. Starting with the running example, simplifying on the right-hand side of (A) using rule (R8) yields $(\mathcal{E}_0, \emptyset) \vdash (\mathcal{E}_1, \emptyset)$ with

$$\mathcal{E}_1 = \{(\mathbf{B}) \text{ sumfun } f \ n \approx \text{fold } [+]\ 0 \ (\text{map } f \ (n : (\text{init } (n - 1)))) \ [n \geq 0]\}$$

Using subsequent (Simplify) steps with **(R7)** and **(R4)**, we have $(\mathcal{E}_1, \emptyset) \vdash^* (\mathcal{E}_3, \emptyset)$ with $\mathcal{E}_3 = \{(\mathbf{C}) \text{ sumfun } f \ n \approx \text{fold } [+] \ (0 + (f \ n)) \ (\text{map } f \ (\text{init } (n - 1))) \ [n \geq 0] \}$.

Expanding equations (doing a case analysis)

After a few simplifications, we typically end up in a state where nothing can be done without knowing how the variables are instantiated. The (Expansion) rule allows us to do a case analysis and create an induction hypothesis at the same time.

(Expansion) Let $s \approx t \ [\psi]$ be an equation. Let $p \in \text{Pos}_F(s)$ be an *expansion-position* of s , meaning that $s|_p = f \ s_1 \cdots s_n$ with $f \in \mathcal{D}$, $n \geq k = \text{ar}(f)$ and every argument s_i is a semi-constructor term. If additionally $\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t \ [\psi]\}$ is terminating then we can perform an (Expansion) step:

$$(\mathcal{E} \uplus \{s \approx t \ [\psi]\}, \mathcal{H}) \vdash (\mathcal{E} \cup \text{Expd}(s \approx t \ [\psi], p), \mathcal{H} \cup \{s \rightarrow t \ [\psi]\})$$

where

$$\text{Expd}(s \approx t \ [\psi], p) = \left\{ s[r \ s_{k+1} \cdots s_n]_p \delta \approx t \delta \ [(\varphi \delta) \wedge (\psi \delta)] \mid \begin{array}{l} \ell \rightarrow r \ [\varphi] \in \mathcal{R}, \ \delta = \text{mgu}(f \ s_1 \cdots s_k, \ell), \\ \delta(\text{Var}(\varphi) \cup \text{Var}(\psi)) \subseteq \mathcal{Val} \cup \mathcal{V} \end{array} \right\}$$

There is an analogous rule for performing (Expansion) on the right-hand side of an equation at an expansion-position of t . In that case, $t \rightarrow s \ [\varphi]$ is added to $\mathcal{H}$.

Example 3.2.3. In the running example, we expand the left-hand side of **(C)** at position ϵ . This gives $(\mathcal{E}_3, \emptyset) \vdash (\mathcal{E}_4, \mathcal{H}_1)$, where:

$$\mathcal{E}_4 = \left\{ \begin{array}{l} \mathbf{(D)} \quad f \ n \approx \text{fold } [+] \ (0 + (f \ n)) \ (\text{map } f \ (\text{init } (n - 1))) \ [n \geq 0 \wedge n \leq 0] \\ \mathbf{(E)} \quad (f \ n) + (\text{sumfun } f \ (n - 1)) \approx \\ \qquad \qquad \text{fold } [+] \ (0 + (f \ n)) \ (\text{map } f \ (\text{init } (n - 1))) \ [n \geq 0 \wedge n > 0] \end{array} \right\}$$

$$\mathcal{H}_1 = \{ \mathbf{(C')} \quad \text{sumfun } f \ n \rightarrow \text{fold } [+] \ (0 + (f \ n)) \ (\text{map } f \ (\text{init } (n - 1))) \ [n \geq 0] \}$$

Viewing an equation as a way to represent a set of ground equations, (Expansion) essentially splits this into multiple subsets, based on the possible instances at position p of s . Since we assume quasi-reductivity, there is at least one rule applicable at position p of $s\gamma$, for any ground semi-constructor substitution γ . For every such rule, the set $\text{Expd}(s \approx t \ [\psi], p)$ contains a resulting representative equation.

Note that, after applying (Expansion), the equation $s \approx t \ [\psi]$ becomes an induction hypothesis, because we add it to $\mathcal{H}$ as an oriented equation. Therefore, we can think of (Expansion) as starting an induction proof on the subterm $s|_p$.

(Expansion2) To be able to perform case-analysis without adding an induction hypothesis to $\mathcal{H}$ requires the following variant of the (Expansion) rule

$$\frac{(\mathcal{E} \uplus \{s \approx t [\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \text{Expd}(s \approx t [\psi], p), \mathcal{H})} \quad p \text{ expansion-position of } s$$

which is shown in Figure 3.2 as (Expansion2), but which was unintentionally omitted from the paper [27]. We include it here because in situations where we only require case analysis, (Expansion) imposes a redundant termination check, which may even cause proofs to get stuck.

Altering and generalizing equations

Intuitively, in Example 3.2.3 equation (D), we should be able to apply (Simplify), reducing $\text{init } (n - 1)$ to nil , because the constraint implies $n - 1 < 0$. However, the deduction rule does not allow this: the variable in the init rule must be instantiated by a value or variable. Nor can we apply (Simplify) with a calculation rule.

Of course, we could adapt the definition of (Simplify), but this problem is actually part of a larger pattern: since our derivation rules rely on the shape of an equation, it is often useful to alter an equation to a semantically equivalent one. That is, since an equation represents the set of its ground semi-constructor instances, we should be able to replace it by an equation that represents the same set.

We define two very similar derivation rules: one that lets us replace an equation by another equation that represents the *same* set, and a second that lets us replace an equation by another that may represent a *larger* set.

(Generalize) and (Alter) Assume that $u \approx v [\psi]$ *generalizes* $s \approx t [\varphi]$, meaning that for every ground semi-constructor substitution (gsc) substitution γ that respects $s \approx t [\varphi]$ there exists a substitution δ that respects $u \approx v [\psi]$ such that $s\gamma = u\delta$ and $t\gamma = v\delta$. We can then perform a (Generalize) step:

$$(\mathcal{E} \uplus \{s \approx t [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E} \cup \{u \approx v [\psi]\}, \mathcal{H})$$

If, moreover, for every gsc substitution δ that respects $u \approx v [\psi]$ there exists a substitution γ that respects $s \approx t [\varphi]$ such that $s\gamma = u\delta$ and $t\gamma = v\delta$, then we say that $u \approx v [\psi]$ *alters* $s \approx t [\varphi]$ and refer to this deduction step as (Alter) instead.

Example 3.2.4. We will apply (Alter) to equation (D) and (E)

- **(D):** as $n \geq 0 \wedge n \leq 0$ is logically equivalent to $-1 = n - 1$, we use (Alter) to obtain $f \ n \approx \text{fold } [+] \ (0 + (f \ n)) \ (\text{map } f \ (\text{init } (n - 1))) \ [-1 = n - 1]$. We then apply (Simplify)

using the calculation rule $x - y \rightarrow z [z = x - y]$ under substitution $[x := n, y := 1, z := -1]$ to obtain $f\ n \approx \text{fold } [+] (0 + (f\ n)) (\text{map } f (\text{init } (-1))) [-1 = n - 1]$

- **(E)**: since $n \geq 0 \wedge n > 0 \iff \exists m [n > 0 \wedge m = n - 1]$, we change the constraint into $[n > 0 \wedge m = n - 1]$, and apply two calculation steps to obtain $(f\ n) + (\text{sumfun } f\ m) \approx \text{fold } [+] (0 + (f\ n)) (\text{map } f (\text{init } m)) [n > 0 \wedge m = n - 1]$

We use (Simplify) a few more times to eventually end up at $(\{\mathbf{(F)}, \mathbf{(G)}\}, \mathcal{H}_1)$, where

$$\begin{aligned} \mathbf{(F)} \quad & f\ n \approx 0 + (f\ n) [-1 = n - 1] \\ \mathbf{(G)} \quad & (f\ n) + (\text{sumfun } f\ m) \approx \\ & \text{fold } [+] ((0 + (f\ n)) + (f\ m)) (\text{map } f (\text{init } (m - 1))) [n > 0 \wedge m = n - 1] \end{aligned}$$

Neither (Simplify) nor (Expansion) can be applied to **(F)**, and it is not obviously removable (see Section 3.2.1) since f could be instantiated by anything in $\mathcal{SCT}_{\mathcal{L}}^0$. However, with (Generalize) we replace **(F)** by $x \approx 0 + x$. Then, applying (Alter) and (Simplify) with a calculation rule, we obtain $(\{\mathbf{(F')}, \mathbf{(G)}\}, \mathcal{H}_1)$ with **(F')** $x \approx x [x = 0 + x]$.

Discussion Despite their similarity, (Alter) and (Generalize) are used in very different ways. The (Alter) rule is more innocent: replacing an equation by an equivalent one cannot harm the proof process – this in contrast to (Generalize), which can easily replace an equation that is an inductive theorem by one that is not.

In practice, (Alter) is typically used in combination with other derivation rules, e.g., to set up a (Simplify) step with a calculation as done in Example 3.2.4. We most often use (Alter) to replace $s \approx t [\varphi]$ by $u \approx v [\psi]$ in the following scenarios:

I. Replacing a constraint by an equivalent one

$s = u$ and $t = v$ and $(\exists \vec{x}. \varphi) \iff (\exists \vec{y}. \psi)$ is valid, where $\{\vec{x}\} = \text{Var}(\varphi) \setminus (\text{Var}(s) \cup \text{Var}(t))$ and $\{\vec{y}\} = \text{Var}(\psi) \setminus (\text{Var}(u) \cup \text{Var}(v))$.

This application of (Alter) is often done before a (Simplify) step, to put the constraint in the right shape.

II. Replacing variables/values by equivalent ones

$s = C[x_1, \dots, x_n]$ and $u = C[y_1, \dots, y_n]$ and $t = v$ and $\varphi = \psi$, if all x_i, y_i are values or variables in $\text{Var}(\varphi)$, and $\varphi \implies \bigwedge_{i=1}^n x_i = y_i$ is valid. For example, replacing $f\ n \approx f\ 0 [n = 0]$ by $f\ 0 \approx f\ 0 [n = 0]$.

This application of (Alter) is particularly useful when $t = C[y_1, \dots, y_n]$.

III. Adding safe variables to the constraint

$s = u$ and $t = v$ and $\psi = \varphi \wedge (x_1 = x_1) \wedge \dots \wedge (x_n = x_n)$, where $x_1, \dots, x_n \in (\text{Var}(s) \cup \text{Var}(u)) \setminus \text{Var}(\varphi)$, whose type is a theory sort $\iota \in \mathcal{S}_{\text{theory}}$ such that no

constructors of a type $\sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota$ exist other than values (and therefore, any ground semi-constructor instance of this variable must be a value).

On the other hand, (Generalize) is primarily used as a form of *lemma generation*: as we will see in Example 3.2.8, it is sometimes needed to generalize an equation to obtain a stronger induction hypothesis. Finding suitable lemmas is a core challenge in inductive theorem proving, being the central topic of Chapter 4. Generalization is also useful to abstract away from variable applications, as we did for $f\ n$ in Example 3.2.4.

Proof: (Alter) scenarios

We just claimed that we often use (Alter) in the scenarios I, II and III. We show that these scenarios indeed satisfy the requirements of the (Alter) rule.

Lemma 3.2.5 (Scenario I: replacing a constraint by an equivalent one). Let $s \approx t [\varphi]$ and $s \approx t [\psi]$ be equations such that $(\exists \vec{x}. \varphi) \iff (\exists \vec{y}. \psi)$ is valid, where $\{\vec{x}\} = \text{Var}(\varphi) \setminus (\text{Var}(s) \cup \text{Var}(t))$ and $\{\vec{y}\} = \text{Var}(\psi) \setminus (\text{Var}(s) \cup \text{Var}(t))$. Then

- (1). For every gsc substitution γ that respects $s \approx t [\varphi]$ there is a substitution δ that respects $s \approx t [\psi]$ such that $s\gamma = s\delta$ and $t\gamma = t\delta$.
- (2). For every gsc substitution δ that respects $s \approx t [\psi]$ there is a substitution γ that respects $s \approx t [\varphi]$ such that $s\gamma = s\delta$ and $t\gamma = t\delta$.

Proof. The cases are symmetric, so we only show (1).

Let $\vec{z} = (\text{Var}(\varphi) \cup \text{Var}(\psi)) \setminus \{\vec{x}, \vec{y}\}$, and write $\vec{x} = (x_1, \dots, x_n)$, $\vec{y} = (y_1, \dots, y_m)$ and $\vec{z} = (z_1, \dots, z_k)$. Validity of $\forall \vec{z}. ((\exists \vec{x}. \varphi) \iff (\exists \vec{y}. \psi))$ implies validity of the one-way implication $\forall \vec{z}. ((\exists \vec{x}. \varphi) \implies (\exists \vec{y}. \psi))$. This means that for all values $c_1, \dots, c_k$ of the right types: if there exist values $a_1, \dots, a_n$ such that $\varphi[x_1 := a_1, \dots, x_n := a_n, z_1 := c_1, \dots, z_k := c_k]$ is valid, then there exist values $b_1, \dots, b_m$ such that $\psi[y_1 := b_1, \dots, y_m := b_m, z_1 := c_1, \dots, z_k := c_k]$ is valid.

Let γ be a gsc substitution that respects $s \approx t [\varphi]$, i.e. $\gamma(\text{Var}(\varphi)) \subseteq \mathcal{V}al$, $\llbracket \varphi \gamma \rrbracket = \top$ and $\text{Var}(s) \cup \text{Var}(t) \subseteq \text{dom}(\gamma)$. Let $a_i := \gamma(x_i)$, and for $1 \leq j \leq k$: if $z_j \in \text{Var}(\varphi)$ let $c_j := \gamma(z_j)$, otherwise let c_j be an arbitrary value in $\mathcal{I}_{\text{typeof}(z_j)}$ (since we assumed that the sets $\mathcal{I}_i$ are non-empty, this can always be done). Moreover, let $\eta := [w := \gamma(w) \mid w \in (\text{Var}(s) \cup \text{Var}(t)) \setminus \text{Var}(\varphi)]$. Then $\text{dom}(\eta) \cap \{\vec{z}\} = \emptyset$.

Now, since γ respects the equation, $\varphi[x_1 := a_1, \dots, x_n := a_n, z_1 := c_1, \dots, z_k := c_k]$ is valid, and therefore by assumption we find values $y_1 := b_1, \dots, y_m := b_m$ such that $\psi[y_1 := b_1, \dots, y_m := b_m, z_1 := c_1, \dots, z_k := c_k]$ is valid. Since $\{\vec{y}\} \cap (\text{Var}(s) \cup \text{Var}(t)) = \emptyset$, we can define $\delta := \eta \cup [\vec{y} := \vec{b}, \vec{z} := \vec{c}]$.

Then clearly $\delta(\text{Var}(\psi)) \subseteq \{\vec{b}, \vec{c}\} \subseteq \mathcal{V}al$, and $\llbracket \psi \delta \rrbracket = \llbracket \psi[\vec{y} := \vec{b}, \vec{z} := \vec{c}] \rrbracket = \top$. All variables in s and t are either in $\{\vec{z}\}$ or in $\text{dom}(\eta)$, so δ respects $s \approx t [\psi]$, and maps each variable $w \in \text{Var}(s) \cup \text{Var}(t)$ to $\gamma(w)$. Hence, $s\gamma = s\delta$ and $t\gamma = t\delta$. $\square$

Lemma 3.2.6 (Scenario II: replacing variables/values by equivalent ones). Let $s \approx t [\varphi]$ and $u \approx t [\varphi]$ be equations such that $s = C[x_1, \dots, x_n]$, $u = C[y_1, \dots, y_n]$ and all x_i, y_i are values or variables in $\text{Var}(\varphi)$. Assume that $\varphi \implies \bigwedge_{i=1}^n x_i = y_i$ is valid. Then

- (1). For every gsc substitution γ that respects $s \approx t [\varphi]$ there exists a substitution δ that respects $u \approx t [\varphi]$ such that $s\gamma = u\delta$ and $t\gamma = t\delta$.
- (2). For every gsc substitution δ that respects $u \approx t [\varphi]$ there exists a substitution γ that respects $s \approx t [\varphi]$ such that $s\gamma = u\delta$ and $t\gamma = t\delta$.

Proof. The cases are symmetric, so we only show (1).

Let γ be a gsc substitution that respects $s \approx t [\varphi]$, i.e. $\gamma(\text{Var}(\varphi)) \subseteq \mathcal{Val}$, $\llbracket \varphi \gamma \rrbracket = \top$ and $\text{Var}(s) \cup \text{Var}(t) \subseteq \text{dom}(\gamma)$. Since all x_i, y_i are values or variables in φ , by definition of *respects* we know that each $x_i\gamma$ and $y_i\gamma$ is a value. By validity of $\varphi \implies \bigwedge_{i=1}^n x_i = y_i$, and the fact that $\llbracket \varphi \gamma \rrbracket = \top$, we obtain that $\llbracket x_i\gamma = y_i\gamma \rrbracket = \top$ for each i , which can only hold if they are the same values since the relation between values and the underlying set is one-to-one. Hence, $x_i\gamma = y_i\gamma$, so $s\gamma = u\gamma$. We are done by choosing $\delta = \gamma$. $\square$

Lemma 3.2.7 (Scenario III: adding safe variables to the constraint). Let $s \approx t [\varphi]$ and $s \approx t [\psi]$ be equations such that $\psi = \varphi \wedge (x_1 = x_1) \wedge \dots \wedge (x_n = x_n)$, where $x_1, \dots, x_n$ are variables in s, t that do not occur in φ , but whose type is a theory sort $\iota \in \mathcal{S}_{\text{theory}}$ such that no constructors of a type $\sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota$ exist other than values. Then

- (1). For every gsc substitution γ that respects $s \approx t [\varphi]$ there exists a substitution δ that respects $s \approx t [\psi]$ such that $s\gamma = s\delta$ and $t\gamma = t\delta$.
- (2). For every gsc substitution δ that respects $s \approx t [\psi]$ there exists a substitution γ that respects $s \approx t [\varphi]$ such that $s\gamma = s\delta$ and $t\gamma = t\delta$.

Proof. For (1), let γ be a gsc substitution that respects $s \approx t [\varphi]$, i.e. $\gamma(\text{Var}(\varphi)) \subseteq \mathcal{Val}$, $\llbracket \varphi \gamma \rrbracket = \top$ and $\text{Var}(s) \cup \text{Var}(t) \subseteq \text{dom}(\gamma)$. Consider $\gamma(x_i)$. Because $x_i \in \text{dom}(\gamma)$ and γ is a gsc substitution, $\gamma(x_i)$ must be a gsc term; having base type, it must have a form $c u_1 \dots u_k$ with c a constructor. Since there are no non-value constructors with a theory sort as output sort, we can only have $k = 0$ and c is a value. But then γ also respects $x_i = x_i$. Hence, we can choose $\delta := \gamma$.

For (2), we observe that any gsc substitution that respects ψ clearly also respects φ , so here too we can choose $\delta := \gamma$. $\square$

Removing equations

Thus far, we have only modified equations. We now introduce a deduction rule to remove an equation from $\mathcal{E}$.

(Delete) Let $s \approx t [\varphi]$ be such that either φ is unsatisfiable or $s = t$, then we can perform a (Delete) step:

$$(\mathcal{E} \uplus \{s \approx t [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E}, \mathcal{H})$$

Example 3.2.8. We continue on the proof state obtained in Example 3.2.4. We apply (Delete) to **(F')** and obtain $(\mathcal{E}_6, \mathcal{H}_1)$ with $\mathcal{E}_6 = \{(\mathbf{G}) (f\ n) + (\text{sumfun } f\ m) \approx \text{fold } [+]\ ((0 + (f\ n)) + (f\ m)) (\text{map } f\ (\text{init } (m - 1))) [n > 0 \wedge m = n - 1]\}$.

We have now defined all the inference rules necessary to complete the running example. The remaining part is *mostly* straightforward; we detail only the harder steps. First, in equation **(G)**, we use (Simplify) with the induction rule **(C')** $\text{sumfun } f\ n \rightarrow \text{fold } [+]\ (0 + (f\ n)) (\text{map } f\ (\text{init } (n - 1))) [n \geq 0]$ to obtain

$$(f\ n) + (\text{fold } [+]\ (0 + (f\ m)) (\text{map } f\ (\text{init } (m - 1)))) \approx \text{fold } [+]\ ((0 + (f\ n)) + (f\ m)) (\text{map } f\ (\text{init } (m - 1))) [n > 0 \wedge m = n - 1]$$

After (Alter) and a calculation step, we obtain

$$(f\ n) + (\text{fold } [+]\ (0 + (f\ m)) (\text{map } f\ (\text{init } k))) \approx \text{fold } [+]\ ((0 + (f\ n)) + (f\ m)) (\text{map } f\ (\text{init } k)) [n > 0 \wedge m = n - 1 \wedge k = m - 1]$$

Now, we could continue by performing additional (Expansion) steps, but this will result in a loop: none of the generated induction rules will be applicable. To avoid this issue, we instead use (Generalize) to obtain

$$x + (\text{fold } [+]\ (0 + y)\ l) \approx \text{fold } [+]\ ((0 + x) + y)\ l$$

In the next (Expansion) step, the termination requirement forces us to expand on the left rather than the right, creating an induction rule

$$x + (\text{fold } [+]\ y\ l) \rightarrow \text{fold } [+]\ z\ l [z = x + y]$$

This rule has a calculation symbol $+$ as the root symbol on the left, which is non-standard, but allowed in LCSTRSs, and termination can be proved. The rest of the proof is entirely straightforward.

Breaking up equations

We now introduce the (Semi-constructor) rule; a higher-order generalization of the (Constructor) rule [22]. Although this deduction rule was not needed in our running example, we sketch in Example 3.2.9 a hypothetical situation where it is necessary.

(Semi-constructor) Let $\vec{s} = s_1 \dots s_n$ and $\vec{t} = t_1 \dots t_n$ be terms with $n > 0$. If f is either a variable, a constructor, a defined symbol with $ar(f) > n$ or a symbol in $\Sigma_{calc} \setminus \mathcal{D}$ of type $\sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota$ with $m > n$ then we can perform a (Semi-constructor) step

$$(\mathcal{E} \uplus \{f \vec{s} \approx f \vec{t} [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E} \cup \{s_i \approx t_i [\varphi] \mid 1 \leq i \leq n\}, \mathcal{H})$$

Example 3.2.9. In an extension of $\mathcal{R}_{fold}$ containing additional symbols $g, h \in \Sigma$, we could e.g. have a deduction step $(\{\text{fold } g \ (h \ 0 \ x) \approx \text{fold } h \ (g \ 0 \ x)\}, \emptyset) \vdash (\{g \approx h, \ h \ 0 \ x \approx g \ 0 \ x\}, \emptyset)$.

3.2.2 Comparison to the first-order literature

Surprisingly few changes were needed to adapt the first-order definitions [22] to the higher-order setting. The most important changes are the new definitions of quasi-reductivity and semi-constructor terms. The proof was adapted to take these changes into account, but its overall structure remains the same as in the first-order case.

The most significant change could already have been made in [22] and considers the introduction of (Alter), and updating (Generalize) to quantify over ground *semi-constructor* substitutions, rather than *all* substitutions. In the first-order system [22], scenario I was combined with (Simplify), and a separate rule (EQ-DELETION) was used to handle II, but III was not supported – thus leaving it impossible to prove for instance $\text{init } (n + 1) \approx \text{init } (1 + n)$ if n was not in the constraint. This limitation is particularly relevant in the higher-order setting: due to proof states with higher-order variables, such as (F) from Example 3.2.4, the (Generalize) rule is needed much more often than in first-order RI, and to progress the proof further we need to be able to move the resulting variables into the constraint.

3.2.3 Proof: Soundness of HORI

In this section we will prove Theorem 3.2.1. The proof is quite elaborate, but not very new: it barely differs from its first-order counterpart [22]. Nevertheless, we supply it here to show that, using our higher-order adaptations, the same strategy still works.

The proof is split over two parts:

- (1). We introduce an invariant, being preserved by every HORI deduction step. We prove that this invariant implies soundness of HORI.
- (2). We then show that this invariant is indeed preserved by every deduction rule.

Part (1): the invariant

We fix a quasi-reductive, terminating LCSTRS $\mathcal{L}$ (with rules $\mathcal{R}$), and we assume that every equation in $\mathcal{E}$ is entirely over $\mathcal{L}$. We recall from Section 3.1.1 the symmetric

relation $\leftrightarrow_{\mathcal{E}}$ on a set of equations $\mathcal{E}$, where we now emphasize that all terms are built from $\mathcal{L}$ (as we will see in Section 3.3 this is precisely what changes in Global RI):

$$C[s\gamma] \leftrightarrow_{\mathcal{E}} C[t\gamma] \iff \begin{cases} C \text{ is a context over } \mathcal{L}, \\ s \approx t [\varphi] \in \mathcal{E} \text{ or } t \approx s [\varphi] \in \mathcal{E}, \\ \gamma \text{ respects } \varphi \end{cases}$$

Lemma 3.2.10. Suppose that $\leftrightarrow_{\mathcal{E}} \subseteq \leftrightarrow_{\mathcal{R}}^*$ on ground terms over $\mathcal{L}$. Then every equation in $\mathcal{E}$ is a higher-order inductive theorem of $\mathcal{L}$.

Proof. Let $s \approx t [\varphi] \in \mathcal{E}$ and γ a ground substitution over $\mathcal{L}$ which respects this equation. We need to prove that $s\gamma \leftrightarrow_{\mathcal{R}}^* t\gamma$. Since $s\gamma$ and $t\gamma$ are ground over $\mathcal{L}$, and satisfy $s\gamma \leftrightarrow_{\mathcal{E}} t\gamma$, we use the assumption to conclude $s\gamma \leftrightarrow_{\mathcal{R}}^* t\gamma$. $\square$

We continue by defining $\leftrightarrow_{\mathcal{E}}$ as the parallel application of zero or more $\leftrightarrow_{\mathcal{E}}$ steps (that is, $s \leftrightarrow_{\mathcal{E}} t$ if there exists a context $C[\square_1, \dots, \square_n]$ and terms $s_1, \dots, s_n, t_1, \dots, t_n$ with $s = C[s_1, \dots, s_n]$, $t = C[t_1, \dots, t_n]$ and $s_i \leftrightarrow_{\mathcal{E}} t_i$ for all $1 \leq i \leq n$).

We additionally define

$$u\gamma \leftrightarrow_{\mathcal{E}}^{\text{root,semi}} v\gamma \text{ if } u \approx v [\varphi] \in \mathcal{E} \text{ and } \gamma \text{ a gsc substitution over } \mathcal{L} \text{ that respects } \varphi$$

The following lemma expresses an invariant, satisfied by every HORI deduction rule:

Lemma 3.2.11 (Invariant). Suppose that $(\mathcal{E}, \mathcal{H}) \vdash (\mathcal{E}', \mathcal{H}')$. Then

- (1). $\leftrightarrow_{\mathcal{E} \setminus \mathcal{E}'}^{\text{root,semi}} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}'}^* \cdot \leftrightarrow_{\mathcal{E}'} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}'}^*$
- (2). $\rightarrow_{\mathcal{R} \cup \mathcal{H}} \subseteq \rightarrow_{\mathcal{R}} \cdot \rightarrow_{\mathcal{R} \cup \mathcal{H}'}^* \cdot \leftrightarrow_{\mathcal{E}} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}'}^*$ on ground terms over $\mathcal{L}$, implies
 $\rightarrow_{\mathcal{R} \cup \mathcal{H}'} \subseteq \rightarrow_{\mathcal{R}} \cdot \rightarrow_{\mathcal{R} \cup \mathcal{H}'}^* \cdot \leftrightarrow_{\mathcal{E}'} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}'}^*$ on ground terms over $\mathcal{L}$.

This lemma will be proven in part (2). For now, we just assume that it holds, to derive that if $(\mathcal{E}, \emptyset) \vdash^* (\emptyset, \mathcal{H})$ then $\leftrightarrow_{\mathcal{E}} \subseteq \leftrightarrow_{\mathcal{R}}^*$ holds on ground terms over $\mathcal{L}$ (by Lemma 3.2.10 it then follows that every equation in $\mathcal{E}$ is a higher-order inductive theorem of $\mathcal{L}$).

We first note:

Lemma 3.2.12. Suppose $\leftrightarrow_{\mathcal{E} \setminus \mathcal{E}'}^{\text{root,semi}} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}'}^* \cdot \leftrightarrow_{\mathcal{E}'} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}'}^*$.

Then $\leftrightarrow_{\mathcal{E}} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}'}^* \cdot \leftrightarrow_{\mathcal{E}'} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}'}^*$ on ground terms over $\mathcal{L}$.

Proof. Suppose $\leftrightarrow_{\mathcal{E} \setminus \mathcal{E}'}^{\text{root,semi}} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}'}^* \cdot \leftrightarrow_{\mathcal{E}'} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}'}^*$ and let s, t be ground terms such that $s \leftrightarrow_{\mathcal{E}} t$. That is, $s = C[s_1\gamma_1, \dots, s_n\gamma_n]$ and $t = C[t_1\gamma_1, \dots, t_n\gamma_n]$ where, for each $1 \leq i \leq n$, $s_i \approx t_i [\varphi_i] \in \mathcal{E}$ and γ_i respects φ_i . Consider an arbitrary i .

If $s_i \approx t_i [\varphi_i] \in \mathcal{E}'$, then clearly $s_i\gamma_i \leftrightarrow_{\mathcal{E}'}^* t_i\gamma_i$, so also $s_i\gamma_i \rightarrow_{\mathcal{R} \cup \mathcal{H}'}^* \cdot \leftrightarrow_{\mathcal{E}'} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}'}^* t_i\gamma_i$. Otherwise, $s_i \approx t_i [\varphi_i] \in \mathcal{E} \setminus \mathcal{E}'$. Since we have assumed that $\mathcal{L}$ is terminating, we can define $\gamma_i^\downarrow = [x := \gamma_i(x) \downarrow_{\mathcal{R}} \mid x \in \text{dom}(\gamma_i)]$ and since $\mathcal{L}$ is quasi-reductive, $\gamma_i^\downarrow$ is a gsc

substitution. Hence, by the assumption on $\leftrightarrow_{\mathcal{E} \setminus \mathcal{E}'}^{\text{root, semi}}$ and we have $s_i \gamma_i \rightarrow_{\mathcal{R}}^* s_i \gamma_i^\downarrow \rightarrow_{\mathcal{R} \cup \mathcal{H}'}^* \cdot \leftrightarrow_{\mathcal{E}'} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}'}^* t_i \gamma_i^\downarrow \leftarrow_{\mathcal{R}}^* t_i \gamma_i$.

We complete by sequentializing all $\rightarrow_{\mathcal{R} \cup \mathcal{H}'}$ reductions in s , and doing all $\leftrightarrow_{\mathcal{E}'}$ steps in parallel. $\square$

With the previous two lemmas we can prove the following:

Lemma 3.2.13. Suppose that $(\mathcal{E}_0, \emptyset) \vdash^* (\mathcal{E}_n, \mathcal{H}_n)$, and assume that Lemma 3.2.11 holds for any deduction step $(\mathcal{E}_{i-1}, \mathcal{H}_{i-1}) \vdash (\mathcal{E}_i, \mathcal{H}_i)$ in this deduction sequence. Then

- (1). $\leftrightarrow_{\mathcal{E}_0} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}_n}^* \cdot \leftrightarrow_{\mathcal{E}_n} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}_n}^*$ on ground terms over $\mathcal{L}$.
- (2). $\rightarrow_{\mathcal{R} \cup \mathcal{H}_n} \subseteq \rightarrow_{\mathcal{R}} \cdot \rightarrow_{\mathcal{R} \cup \mathcal{H}_n}^* \cdot \leftrightarrow_{\mathcal{E}_n} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}_n}^*$ on ground terms over $\mathcal{L}$.

Proof. We prove both statements by induction on n . In particular, we will use that $\mathcal{H}_i \subseteq \mathcal{H}_{i+1}$, for all $0 \leq i < n$. (This is a property of rewriting induction.)

If $n = 0$, then we are immediately done, since $\mathcal{E}_n = \mathcal{E}_0$ and $\rightarrow_{\mathcal{R} \cup \mathcal{H}_n} = \rightarrow_{\mathcal{R}}$ (and because a parallel step $\leftrightarrow_{\mathcal{E}_0}$ is allowed to be empty).

Now, assume $n > 0$ and the lemma holds for $n - 1$. So we have $\leftrightarrow_{\mathcal{E}_0} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}_{n-1}}^* \cdot \leftrightarrow_{\mathcal{E}_{n-1}} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}_{n-1}}^*$. By the combination of Lemma 3.2.11 and Lemma 3.2.12: $\leftrightarrow_{\mathcal{E}_{n-1}} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}_n}^* \cdot \leftrightarrow_{\mathcal{E}_n} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}_n}^*$. Therefore $\leftrightarrow_{\mathcal{E}_0} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}_{n-1}}^* \cdot (\rightarrow_{\mathcal{R} \cup \mathcal{H}_n}^* \cdot \leftrightarrow_{\mathcal{E}_n} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}_n}^*) \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}_{n-1}}^*$. We complete (1) because $\rightarrow_{\mathcal{R} \cup \mathcal{H}_{n-1}}^* \cdot \rightarrow_{\mathcal{R} \cup \mathcal{H}_n}^* \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}_n}^*$. As for (2): this follows immediately by induction hypothesis (2) and Lemma 3.2.11.(2). $\square$

We recall the following property from the literature, on arbitrary relations $\rightarrow_1$ and $\rightarrow_2$. We include a proof since the original proof is in Japanese.

Lemma 3.2.14 ([33, Lemma 3.4]). Let $\rightarrow_1 \subseteq \rightarrow_2$ be binary relations with $\rightarrow_2$ well-founded and $\rightarrow_2 \subseteq \rightarrow_1 \cdot \rightarrow_2^* \cdot \leftarrow_1^* \cdot \leftarrow_2^*$. Then $\leftarrow_1^* = \leftarrow_2^*$.

Proof. The direction $\leftarrow_1^* \subseteq \leftarrow_2^*$ is implied by $\rightarrow_1 \subseteq \rightarrow_2$. To prove $\leftarrow_2^* \subseteq \leftarrow_1^*$ it suffices to prove $\rightarrow_2 \subseteq \leftarrow_1^*$. So assume $s \rightarrow_2 t$. We use well-founded induction on $s \rightarrow_2$ to show $s \leftarrow_1^* t$. First, by assumption: there exist terms $a_1, \dots, a_n$ and $b_1, \dots, b_m$ such that

$$s \rightarrow_1 a_1 \rightarrow_2 a_2 \rightarrow_2 \dots \rightarrow_2 a_n \leftarrow_1^* b_m \leftarrow_2 \dots \leftarrow_2 b_2 \leftarrow_2 b_1 = t$$

Since $\rightarrow_1 \subseteq \rightarrow_2$, we see that $s \rightarrow_2^+ a_i$ for all $1 \leq i \leq n$, and $a_i \rightarrow_2 a_{i+1}$ for all $1 \leq i < n$. So the induction hypothesis gives us $a_i \leftarrow_1^* a_{i+1}$ for all $1 \leq i < n$. By assumption we have $s \rightarrow_2 t$, hence (again by induction hypothesis) $b_i \leftarrow_1^* b_{i+1}$ for all $1 \leq i < m$. This yields the desired sequence $s \rightarrow_1 a_1 \leftarrow_1^* a_2 \leftarrow_1^* \dots \leftarrow_1^* a_n \leftarrow_1^* b_m \leftarrow_1^* \dots \leftarrow_1^* b_2 \leftarrow_1^* b_1 = t$. $\square$

Now, we can prove Theorem 3.2.1, conditional on Lemma 3.2.11.

Theorem 3.2.1 (conditionally). Let $\mathcal{E}$ be a set of equations. Assume that, by HORI, $(\mathcal{E}, \emptyset) \vdash^* (\emptyset, \mathcal{H})$, for some set $\mathcal{H}$, and Lemma 3.2.11 holds for every single step that occurs in this deduction sequence. Then every equation in $\mathcal{E}$ is a higher-order inductive theorem of $\mathcal{L}$.

Proof. We use Lemma 3.2.13 with $\mathcal{E}_0 = \mathcal{E}$, $\mathcal{E}_n = \emptyset$ and $\mathcal{H}_n = \mathcal{H}$ to obtain

- (1). $\hookrightarrow_{\mathcal{E}} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}}^* \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}}^*$ on ground terms over $\mathcal{L}$
- (2). $\rightarrow_{\mathcal{R} \cup \mathcal{H}} \subseteq \rightarrow_{\mathcal{R}} \cdot \rightarrow_{\mathcal{R} \cup \mathcal{H}}^* \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}}^*$ on ground terms over $\mathcal{L}$

Combine (2) with Lemma 3.2.14 to conclude $\leftrightarrow_{\mathcal{R}}^* = \leftrightarrow_{\mathcal{R} \cup \mathcal{H}}^*$ on ground terms over $\mathcal{L}$ (take $\rightarrow_1 := \rightarrow_{\mathcal{R}}$ and $\rightarrow_2 := \rightarrow_{\mathcal{R} \cup \mathcal{H}}$). Then use (1) to conclude that:

$$\leftrightarrow_{\mathcal{E}} \subseteq \hookrightarrow_{\mathcal{E}} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}}^* \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}}^* \subseteq \leftrightarrow_{\mathcal{R} \cup \mathcal{H}}^* \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}}^* = \leftrightarrow_{\mathcal{R} \cup \mathcal{H}}^* = \leftrightarrow_{\mathcal{R}}^*$$

on ground terms over $\mathcal{L}$. Finally, use Lemma 3.2.10 to conclude Theorem 3.2.1. $\square$

Part (2): every deduction rule preserves the invariant

It remains to be seen that the invariant from Lemma 3.2.11 holds for every deduction rule. Note that whenever $\mathcal{H} = \mathcal{H}'$ – as is the case for every deduction rule other than (Expansion) – we only need to prove (1), because then (2) is implied by (1).

Hence, in all cases other than (Expansion) we only have to show that

$$\leftrightarrow_{\mathcal{E} \setminus \mathcal{E}'}^{root, semi} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}}^* \cdot \hookrightarrow_{\mathcal{E}'} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}}^*$$

(Simplify) We first introduce a helper lemma.

Lemma 3.2.15. Let $\mathcal{L} = (\Sigma, \mathcal{R})$ be terminating and suppose that $\mathcal{X} \supseteq \mathcal{R}$ is terminating on $T(\Sigma, \mathcal{V})$. Assume an equation $C[\ell\delta] \approx t[\psi]$, for some rule $\ell \rightarrow r[\varphi] \in \mathcal{X} \cup \mathcal{R}_{calc}$, context C and substitution δ , both over $\mathcal{L}$, such that $\delta(\text{Var}(\varphi)) \subseteq \mathcal{V}al \cup \text{Var}(\psi)$ and $\psi \implies \varphi\delta$ valid. Then for any substitution γ over $\mathcal{L}$ which respects ψ we have $C[\ell\delta]\gamma \rightarrow_{\mathcal{X}} C[r\delta]\gamma$.

Proof. Let γ be a substitution over $\mathcal{L}$ respecting ψ and define $\eta := \gamma \circ \delta$. We check that $C[\ell\delta]\gamma \rightarrow_{\mathcal{X}} C[r\delta]\gamma$ with substitution η and rule $\ell \rightarrow r[\varphi]$.

Let $C' = C\gamma$. Then $C[\ell\delta]\gamma = C'[\ell\delta\gamma] = C'[\ell\eta]$ and $C[r\delta]\gamma = C'[r\delta\gamma] = C'[r\eta]$. Therefore, by definition of the rewrite relation, it suffices to show that η respects $\ell \rightarrow r[\varphi]$, i.e. $\eta(\text{Var}(\varphi)) \subseteq \mathcal{V}al$ and $\llbracket \varphi\eta \rrbracket = \top$. As γ respects ψ we have $\gamma(\text{Var}(\psi)) \subseteq \mathcal{V}al$ and $\llbracket \psi\gamma \rrbracket = \top$. So $\eta(\text{Var}(\varphi)) = \gamma(\delta(\text{Var}(\varphi))) \subseteq \gamma(\mathcal{V}al \cup \text{Var}(\psi)) = \gamma(\mathcal{V}al) \cup \gamma(\text{Var}(\psi)) = \mathcal{V}al \cup \gamma(\text{Var}(\psi)) = \mathcal{V}al$ and, because of $\psi \implies \varphi\delta$, we have $\top = \llbracket (\varphi\delta)\gamma \rrbracket = \llbracket \varphi\eta \rrbracket$. $\square$

Lemma 3.2.16. Assume that $(\mathcal{E} \uplus \{C[\ell\delta] \approx t[\psi]\}, \mathcal{H}) \vdash (\mathcal{E} \cup \{C[r\delta] \approx t[\psi]\}, \mathcal{H})$ by (Simplify), using the rule $\ell \rightarrow r[\varphi]$. Then

$$\leftrightarrow_{\{C[\ell\delta] \approx t[\psi]\}}^{root, semi} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}}^* \cdot \hookrightarrow_{\mathcal{E} \cup \{C[r\delta] \approx t[\psi]\}} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}}^*$$

Proof. Suppose $C[\ell\delta]\gamma \leftrightarrow_{\{C[\ell\delta] \approx t[\psi]\}}^{root, semi} t\gamma$, with γ a gsc substitution over $\mathcal{L}$ that respects ψ . Then, by definition of (Simplify), we have $\delta(\text{Var}(\varphi)) \subseteq \mathcal{V}al \cup \text{Var}(\psi)$ and $\psi \implies \varphi\delta$. Now, Lemma 3.2.15 (take $\mathcal{X} = \mathcal{R} \cup \mathcal{H}$) guarantees $C[\ell\delta]\gamma \rightarrow_{\mathcal{R} \cup \mathcal{H}} C[r\delta]\gamma$. Therefore, $C[\ell\delta]\gamma \rightarrow_{\mathcal{R} \cup \mathcal{H}} C[r\delta]\gamma \hookrightarrow_{\{C[r\delta] \approx t[\psi]\}} t\gamma$. $\square$

(Generalize) and (Alter) Since (Alter) is a special case of (Generalize), we can handle both derivation rules at once.

Lemma 3.2.17. If $(\mathcal{E} \uplus \{s \approx t [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E} \cup \{u \approx v [\psi]\}, \mathcal{H})$ by (Generalize) or (Alter), then $\leftrightarrow_{\{s \approx t [\varphi]\}}^{\text{root,semi}} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}}^* \cdot \Vdash_{\mathcal{E} \cup \{u \approx v [\psi]\}} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}}^*$.

Proof. Suppose $s\gamma \leftrightarrow_{\{s \approx t [\varphi]\}}^{\text{root,semi}} t\gamma$, with γ a gsc substitution over $\mathcal{L}$ that respects φ . Then, by definition of the Generalize and Alter rules, there exists a substitution δ that respects $u \approx v [\psi]$ such that $s\gamma = u\delta$ and $t\gamma = v\delta$. Since δ respects ψ we have $u\delta \leftrightarrow_{\{u \approx v [\psi]\}} v\delta$. Therefore, $s\gamma \leftrightarrow_{\{u \approx v [\psi]\}} t\gamma$. $\square$

(Delete)

Lemma 3.2.18. If $(\mathcal{E} \uplus \{s \approx t [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E}, \mathcal{H})$ by (Delete) then

$$\leftrightarrow_{\{s \approx t [\varphi]\}}^{\text{root,semi}} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}}^* \cdot \Vdash_{\mathcal{E}} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}}^*$$

Proof. Assume $s\gamma \leftrightarrow_{\{s \approx t [\varphi]\}}^{\text{root,semi}} t\gamma$, with γ a gsc substitution over $\mathcal{L}$ that respects φ . Then necessarily $s = t$, because in the other case (the case φ being unsatisfiable) there would not exist such a substitution. But then the result trivially holds. $\square$

(Semi-constructor)

Lemma 3.2.19. Assume $(\mathcal{E} \uplus \{c \vec{s} \approx c \vec{t} [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E} \cup \{s_i \approx t_i [\varphi] \mid 1 \leq i \leq n\}, \mathcal{H})$ by (Semi-constructor). Then

$$\leftrightarrow_{\{c \vec{s} \approx c \vec{t} [\varphi]\}}^{\text{root,semi}} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}}^* \cdot \Vdash_{\mathcal{E} \cup \{s_i \approx t_i [\varphi] \mid 1 \leq i \leq n\}} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}}^*$$

Proof. Suppose that $(c \vec{s})\gamma \leftrightarrow_{\{c \vec{s} \approx c \vec{t} [\varphi]\}}^{\text{root,semi}} (c \vec{t})\gamma$, with γ a gsc substitution over $\mathcal{L}$ that respects φ . By definition, we have $s_i\gamma \leftrightarrow_{\mathcal{E} \cup \{s_i \approx t_i [\varphi] \mid 1 \leq i \leq n\}} t_i\gamma$. As all s_i occur in parallel, it follows that $(c \vec{s})\gamma \Vdash_{\mathcal{E} \cup \{s_i \approx t_i [\varphi] \mid 1 \leq i \leq n\}} (c \vec{t})\gamma$. $\square$

(Expansion) Since (Expansion) adds a rewrite rule to $\mathcal{H}$, part (2) of Lemma 3.2.11 is not automatically implied by part (1). Hence, we have to prove both statements. We start with proving (1), but first we introduce two helper lemmas.

Lemma 3.2.20. Let $f s_1 \cdots s_n$ be a ground term over $\mathcal{L}$ such that $f \in \mathcal{D}$, $n \geq ar(f) = k$ and every $s_i \in \mathcal{SCT}_{\mathcal{L}}$. Then there is a rule $\ell \rightarrow r [\psi] \in \mathcal{R}$ and a substitution δ over $\mathcal{L}$ respecting this rule such that $f s_1 \cdots s_k = \ell\delta$.

Proof. Quasi-reductivity of $\mathcal{L}$ implies that $f s_1 \cdots s_n$ reduces ($n \geq ar(f)$ so it is not a semi-constructor term). Since every s_i is a semi-constructor term, the only possible way this reduction can happen is at the root position. So there is a rule $\ell \rightarrow r [\psi] \in \mathcal{R}$ and a substitution δ over $\mathcal{L}$ respecting this rule such that $f s_1 \cdots s_k = \ell\delta$. $\square$

Lemma 3.2.21. Let $s \approx t [\varphi]$ be an equation such that $s|_p = f s_1 \cdots s_n$ with $f \in \mathcal{D}$, $n \geq ar(f)$ and every s_i a semi-constructor term over $\mathcal{L}$. Then for any gsc substitution γ over $\mathcal{L}$ which respects $s \approx t [\varphi]$ we have $s\gamma \rightarrow_{\mathcal{R}} \cdot \leftarrow \Vdash_{\mathcal{E} \cup \text{Expd}(s \approx t [\varphi], p)} t\gamma$.

Proof. Let γ be a gsc substitution over $\mathcal{L}$ which respects φ . By Lemma 3.2.20, $s|_p\gamma$ reduces at root position with some rule $\ell \rightarrow r [\psi] \in \mathcal{R}$ and substitution δ over $\mathcal{L}$ respecting ψ . We can assume the variables in $\ell \rightarrow r [\psi]$ are named so as not to overlap with the variables in the equation. So then we can let $\delta' := \gamma \cup \delta$ and have $s\gamma = s\delta' = s[\ell s_{k+1} \cdots s_n]_p \delta'$ and $t\gamma = t\delta'$, where δ' respects both φ and ψ . And since δ' respects ψ , we have $s[\ell s_{k+1} \cdots s_n]_p \delta' \rightarrow_{\mathcal{R}} s[r s_{k+1} \cdots s_n]_p \delta'$.

Since $f s_1 \cdots s_k$ and ℓ are unifiable with unifier δ' , there is a most general unifier η , where “most general” implies that $\delta' = \eta\zeta$ for some substitution ζ . Hence, $\text{Expd}(s \approx t [\varphi], p)$ includes an equation $s[r s_{k+1} \cdots s_n]_p \eta \approx t\eta$. Conclude $s\gamma = s[\ell s_{k+1} \cdots s_n]_p \delta' \rightarrow_{\mathcal{R}} s[r s_{k+1} \cdots s_n]_p \delta' = s[r s_{k+1} \cdots s_n]_p \eta\zeta \leftarrow \Vdash_{\text{Expd}(s \approx t [\varphi], p)} t\eta\zeta = t\gamma$. $\square$

Now part (1) of Lemma 3.2.11 is proved by the following.

Lemma 3.2.22.

If $(\mathcal{E} \uplus \{s \approx t [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E} \uplus \text{Expd}(s \approx t [\varphi], p), \mathcal{H} \cup \{s \rightarrow t [\varphi]\})$ then

$$\leftrightarrow_{\{s \approx t [\varphi]\}}^{\text{root, semi}} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t [\varphi]\}}^* \cdot \leftarrow \Vdash_{\mathcal{E} \uplus \text{Expd}(s \approx t [\varphi], p)} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t [\varphi]\}}^*$$

Proof. Suppose $s\gamma \leftrightarrow_{\{s \approx t [\varphi]\}}^{\text{root, semi}} t\gamma$, with γ a gsc substitution over $\mathcal{L}$ that respects φ . By Lemma 3.2.21 we have $s\gamma \rightarrow_{\mathcal{R}} \cdot \leftarrow \Vdash_{\text{Expd}(s \approx t [\varphi], p)} t\gamma$. $\square$

And part (2) of Lemma 3.2.11 is proved by the following.

Lemma 3.2.23.

If $(\mathcal{E} \uplus \{s \approx t [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E} \cup \text{Expd}(s \approx t [\varphi], p), \mathcal{H} \cup \{s \rightarrow t [\varphi]\})$, and $\rightarrow_{\mathcal{R} \cup \mathcal{H}} \subseteq \rightarrow_{\mathcal{R}} \cdot \rightarrow_{\mathcal{R} \cup \mathcal{H}}^* \cdot \leftarrow \Vdash_{\mathcal{E} \cup \{s \approx t [\varphi]\}} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}}^*$ on ground terms over $\mathcal{L}$. Then

$$\rightarrow_{\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t [\varphi]\}} \subseteq \rightarrow_{\mathcal{R}} \cdot \rightarrow_{\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t [\varphi]\}}^* \cdot \leftarrow \Vdash_{\mathcal{E} \cup \text{Expd}(s \approx t [\varphi], p)} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t [\varphi]\}}^*$$

holds on ground terms over $\mathcal{L}$.

Proof. Suppose $\rightarrow_{\mathcal{R} \cup \mathcal{H}} \subseteq \rightarrow_{\mathcal{R}} \cdot \rightarrow_{\mathcal{R} \cup \mathcal{H}}^* \cdot \leftarrow \Vdash_{\mathcal{E} \cup \{s \approx t [\varphi]\}} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}}^*$ on ground terms over $\mathcal{L}$. Then by Lemma 3.2.22 and Lemma 3.2.12

$$\begin{aligned} \rightarrow_{\mathcal{R} \cup \mathcal{H}} \subseteq \rightarrow_{\mathcal{R}} \cdot \rightarrow_{\mathcal{R} \cup \mathcal{H}}^* \cdot \left(\rightarrow_{\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t [\varphi]\}}^* \cdot \right. \\ \left. \leftarrow \Vdash_{\mathcal{E} \cup \text{Expd}(s \approx t [\varphi], p)} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t [\varphi]\}}^* \right) \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}}^* \end{aligned}$$

Or equivalently

$$\rightarrow_{\mathcal{R} \cup \mathcal{H}} \subseteq \rightarrow_{\mathcal{R}} \cdot \rightarrow_{\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t [\varphi]\}}^* \cdot \leftarrow \Vdash_{\mathcal{E} \cup \text{Expd}(s \approx t [\varphi], p)} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t [\varphi]\}}^*$$

Therefore, it suffices to show that on ground terms over $\mathcal{L}$ we have

$$\rightarrow_{\{s \rightarrow t \ [\varphi]\}} \subseteq \rightarrow_{\mathcal{R}} \cdot \rightarrow_{\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t \ [\varphi]\}}^* \cdot \leftarrow_{\mathcal{E} \cup \text{Expd}(s \approx t \ [\varphi], p)} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t \ [\varphi]\}}^*$$

So suppose $C[s\gamma]$, $C[t\gamma]$ are ground terms over $\mathcal{L}$ such that $C[s\gamma] \rightarrow_{\{s \rightarrow t \ [\varphi]\}} C[t\gamma]$, for some ground substitution γ over $\mathcal{L}$ that respects $s \rightarrow t \ [\varphi]$. Then $\gamma^\downarrow$ with $\gamma^\downarrow(x) = \gamma(x) \downarrow_{\mathcal{R}}$ for all $x \in \text{dom}(\gamma)$ is a gsc substitution over $\mathcal{L}$, so by Lemma 3.2.21 we have $s\gamma^\downarrow \rightarrow_{\mathcal{R}} \cdot \leftarrow_{\text{Expd}(s \approx t \ [\varphi], p)} t\gamma^\downarrow$. Therefore

$$C[s\gamma] \rightarrow_{\mathcal{R}}^* C[s\gamma^\downarrow] \rightarrow_{\mathcal{R}} \cdot \leftarrow_{\text{Expd}(s \approx t \ [\varphi], p)} C[t\gamma^\downarrow] \leftarrow_{\mathcal{R}}^* C[t\gamma]$$

□

(Expansion2) Since $\mathcal{H} = \mathcal{H}'$ we only need to prove part (1) of Lemma 3.2.11. We will now show that this is a trivial consequence of the proof of part (1) for (Expansion).

Lemma 3.2.24.

If $(\mathcal{E} \uplus \{s \approx t \ [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E} \uplus \text{Expd}(s \approx t \ [\varphi], p), \mathcal{H})$ then

$$\leftrightarrow_{\{s \approx t \ [\varphi]\}}^{\text{root}, \text{semi}} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}}^* \cdot \leftarrow_{\mathcal{E} \uplus \text{Expd}(s \approx t \ [\varphi], p)} \cdot \leftarrow_{\mathcal{R} \cup \mathcal{H}}^*$$

Proof. We recall the reasoning from the proof of Lemma 3.2.22, which still applies here: if $s\gamma \leftrightarrow_{\{s \approx t \ [\varphi]\}}^{\text{root}, \text{semi}} t\gamma$, with γ a gsc substitution over $\mathcal{L}$ that respects φ then $s\gamma \rightarrow_{\mathcal{R}} \cdot \leftarrow_{\text{Expd}(s \approx t \ [\varphi], p)} t\gamma$, which immediately proves the desired result. □

3.3 Global Rewriting Induction

A very desirable property not being satisfied by higher-order inductive theorems and the proof system HORI is *extensibility*. This means that if an equation is an inductive theorem in $\mathcal{R}$, it remains an inductive theorem in any reasonable (i.e., adding defined symbols, not constructors) LCSTRS extending $\mathcal{R}$. In terminology of functional programming: it should be possible to import functions from external modules without breaking any equivalence. Extensibility allows for more local reasoning: to prove properties about a small part of a larger system, it is very desirable to only have to consider the rules that are directly related. This property is also used in some existing methods for program transformations [19]. Example 3.3.1 comes from existing literature [9] and demonstrates a rewrite system with a non-extensible inductive theorem.

Example 3.3.1. Let $\Sigma_{\text{terms}} = \{\text{zero} :: (\text{nat} \rightarrow \text{nat}) \rightarrow \text{nat}, \text{add} :: \text{nat} \rightarrow \text{nat} \rightarrow \text{nat}, s :: \text{nat} \rightarrow \text{nat}, 0 :: \text{nat}\}$, $\Sigma_{\text{theory}} = \emptyset$ and let $\mathcal{R} = \mathcal{R}_{\text{add}} \cup \mathcal{R}_{\text{zero}}$ be defined by

$$\mathcal{R}_{\text{add}} = \{\text{add } 0 \ y \rightarrow y, \text{add } (s \ x) \ y \rightarrow s \ (\text{add } x \ y)\} \quad \text{and} \quad \mathcal{R}_{\text{zero}} = \{\text{zero } s \rightarrow 0\}$$

Then $\text{add } x \ y \approx \text{add } y \ x$ is an inductive theorem, since $(\text{add } x \ y)\gamma \leftrightarrow_{\mathcal{R}}^* (\text{add } y \ x)\gamma$ holds for any ground substitution γ . However, in the extension $\mathcal{R}' = \mathcal{R} \cup \{f \ x \rightarrow 0\}$ where $f :: \text{nat} \rightarrow \text{nat}$ takes the role of a new defined symbol, $\text{add } x \ y \approx \text{add } y \ x$ is no longer an inductive theorem, because $\text{add } 0 \ (\text{zero } f) \leftrightarrow_{\mathcal{R}'}^* \text{add } (\text{zero } f) \ 0$ does not hold.

3.3.1 Strong quasi-reductivity

A key problem in Example 3.3.1 is that the extension breaks quasi-reductivity: by importing f we create a missing pattern in $\mathcal{R}_{\text{zero}}$. This is caused by *pattern matching on a function*: the last rule in $\mathcal{R}$ matches on the expression s which is a non-variable term of type $\text{nat} \rightarrow \text{nat}$. If we now import any new symbol of this type, no matter how innocent, it creates a new pattern; and thus quasi-reductivity is lost. From the perspective of functional programming, rules like this seem very unnatural; it is not typically *allowed* for a pattern to have a higher-order subterm that is not a variable. Thus, we argue that the original system is inherently problematic. To prevent such pathological examples, we extend the definition of quasi-reductivity to exclude this program structure.

Definition 3.3.2 (CHV term). Let $\mathcal{L}$ be an LCSTRS. A *Constructor term with (only) Higher-order Variables* (CHV term) over $\mathcal{L}$ is a constructor term s over $\mathcal{L}$ such that $\text{Var}(s)$ contains only variables of higher type.

Definition 3.3.3 (Strong quasi-reductivity). An LCSTRS $\mathcal{L} = (\Sigma, \mathcal{R})$ with defined symbols $\mathcal{D}$ is *strong quasi-reductive* if any term of the form $f s_1 \cdots s_n$ with $f \in \mathcal{D}$, $n \geq \text{ar}(f)$ and each s_i a CHV term over $\mathcal{L}$ reduces with $\rightarrow_{\mathcal{R}}$.

Any strong quasi-reductive LCSTRS is also quasi-reductive (see Lemma 3.3.5 below). An LCSTRS is certainly strong quasi-reductive if it has exhaustive pattern matching, left-linear rules and all strict higher-order subterms of left-hand sides are variables. Strong quasi-reductivity is close to (and implies) quasi-reducibility [9]. On the other hand, strong quasi-reductivity is weaker than Higher-order Sufficient Completeness [10].

Proof: strong quasi-reductivity implies quasi-reductivity

We just claimed that strong quasi-reductivity implies general quasi-reductivity. To prove this statement, we first introduce a helper function μ , that replaces every higher-order subterm in a ground semi-constructor term by a variable.

Definition 3.3.4 (μ). For a ground semi-constructor term s , choose variables F_p for every $p \in \text{Pos}_{\mathcal{F}}(s)$ with $s|_p$ of higher type. Then, let $\mu(s) := \mu_{\epsilon}(s)$, where for subterms t of s at position p , we let:

- if t has an arrow type, $\mu_p(t) := F_p$
- if t has base type, then t necessarily has a shape $c t_1 \cdots t_m$ with c a constructor, so we let $\mu_p(t) := c \mu_{p \cdot 1}(t_1) \cdots \mu_{p \cdot m}(t_m)$

Note that $\mu(s)$ is a constructor term with only higher-order variables, and moreover it is linear (no variable occurs more than once), and the *only* subterms of higher type are variables. We clearly have the property that $\mu(s)[F_p := s|_p \mid p \in \text{Pos}_{\mathcal{F}}(s) \wedge s|_p \text{ has a higher type}] = s$. With this, we easily prove the desired result.

Lemma 3.3.5. Any strong quasi-reductive LCSTRS is quasi-reductive.

Proof. Suppose an LCSTRS $\mathcal{L} = (\Sigma, \mathcal{R})$ is strong quasi-reductive, and towards a contradiction, suppose $s \in T(\Sigma, \emptyset)$ is irreducible but not a semi-constructor term. This can only be the case if s has a subterm $f s_1 \cdots s_n$ for some $n \geq ar(f)$, where $f \in \mathcal{D}$ and all s_i are semi-constructor terms, but $f s_1 \cdots s_n$ does not reduce. However, by strong quasi-reductivity, $f \mu_1(s_1) \cdots \mu_n(s_n)$ does reduce! But then $(f \mu_1(s_1) \cdots \mu_n(s_n))[F_p := s|_p \mid p \in \text{Pos}_F(s) \wedge s|_p \text{ has higher type}] = f s_1 \cdots s_n$ must also reduce. This gives the required contradiction. $\square$

3.3.2 Natural extensions and global inductive theorems

Unfortunately, as shown below in Example 3.3.6, limiting interest to strong quasi-reductive systems does not suffice to obtain extensibility.

Example 3.3.6. Let $\mathcal{L} = (\Sigma, \mathcal{R})$ be the LCSTRS defined by $\Sigma_{\text{terms}} = \{a :: A, b :: A, c :: C, f :: (C \rightarrow A) \rightarrow A, g :: C \rightarrow A\}$, $\Sigma_{\text{theory}} = \emptyset$ and $\mathcal{R} = \{f F \rightarrow F c, g x \rightarrow b\}$. Then $f F \approx b$ is an inductive theorem, since the only ground term that can instantiate F is g , and indeed $f g \rightarrow_{\mathcal{R}} g c \rightarrow_{\mathcal{R}} b$. However, it is not an inductive theorem in the extended rewrite system $\mathcal{L}' = (\Sigma', \mathcal{R}')$, with signature $\Sigma' = \Sigma \cup \{h :: C \rightarrow A\}$ and $\mathcal{R}' = \mathcal{R} \cup \{h x \rightarrow a\}$.

Here, $f F \approx b$ is an inductive theorem because the reasoning is restricted to the given rewrite system $\mathcal{L} = (\Sigma, \mathcal{R})$, where the only possible instance of F is g . This of course may not be true in an extension, like $\mathcal{L}' = (\Sigma', \mathcal{R}')$. To avoid examples like this, we follow a similar idea to that in existing work [9] and directly define a kind of inductive theorems that are preserved under reasonable extensions.

Definition 3.3.7 (Natural extensions). An LCSTRS $\mathcal{L}'$ (generated by $\mathcal{S}', \Sigma', \mathcal{R}'$, etc.) is a natural extension of $\mathcal{L}$ (generated by $\mathcal{S}, \Sigma, \mathcal{R}$, etc.) if

- (a). $\mathcal{S}' \supseteq \mathcal{S}$ and $\mathcal{S}'_{\text{theory}} \supseteq \mathcal{S}_{\text{theory}}$ and $\Sigma'_{\text{theory}} \supseteq \Sigma_{\text{theory}}$ and $\Sigma'_{\text{terms}} \supseteq \Sigma_{\text{terms}}$ and $\mathcal{V}' \supseteq \mathcal{V}$ and $\mathcal{R}' \supseteq \mathcal{R}$
- (b). $\mathcal{I}'_i = \mathcal{I}_i$ for all $i \in \mathcal{S}_{\text{theory}}$, and $\llbracket f \rrbracket' = \llbracket f \rrbracket$ for all $f \in \Sigma_{\text{theory}}$
- (c). for all $a \in \Sigma \cup \mathcal{V}$: $\text{typeof}'(a) = \text{typeof}(a)$
- (d). for all $f \in \Sigma$: $\mathcal{R}'_f = \mathcal{R}_f$ (so $\mathcal{R}' \setminus \mathcal{R}$ does not define any of the constructors in $\mathcal{L}$, nor add cases to a defined symbol or calculation symbol)
- (e). for all $f :: \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota \in \Sigma'$, all i : there is a ground term of type σ_i
- (f). for all constructor symbols $c :: \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota \in C' \setminus C$ we have $\iota \notin \mathcal{S}$

Hence, a natural extension can add more rules, but cannot interfere with the original LCSTRS, nor add new patterns to its sorts. By (f), any ground constructor term of a sort ι that occurs in the original signature can only use constructors in this signature.

Example 3.3.8. Let $\Sigma = \{\text{add} :: \text{nat} \rightarrow \text{nat} \rightarrow \text{nat}, \text{s} :: \text{nat} \rightarrow \text{nat}, 0 :: \text{nat}\}$ and $\mathcal{R} = \{\text{add } 0 \ y \rightarrow y, \text{add } (\text{s } x) \ y \rightarrow \text{s } (\text{add } x \ y)\}$. The system defined by $\Sigma' = \Sigma \cup \{\text{double} :: \text{nat} \rightarrow \text{nat}\}$ and $\mathcal{R}' = \mathcal{R} \cup \{\text{double } x \rightarrow \text{add } x \ x\}$ is a natural extension of $(\Sigma, \mathcal{R})$.

Definition 3.3.9 (Global inductive theorems). An equation $s \approx t \ [\varphi]$ over a terminating, strong quasi-reductive LCSTRS $\mathcal{L}$ is a global inductive theorem of $\mathcal{L}$ if for every terminating, quasi-reductive natural extension $\mathcal{L}'$ with rules $\mathcal{R}'$ and every ground substitution γ over $\mathcal{L}'$ that respects this equation: $s\gamma \leftrightarrow_{\mathcal{R}'}^* t\gamma$.

Discussion At this point, it is worthwhile to compare ground convertibility to another frequently used equivalence notion in the literature: *contextual equivalence* (also known as behavioral equivalence) [42]. The notion is motivated by the desire to consider two expressions s and t equivalent if any occurrence of s and t in any other program $C[\square]$ can be interchanged without affecting the results of executing that program, meaning that $C[s]$ and $C[t]$ should evaluate to the same value with respect to the pre-defined operational semantics. The definition of contextual equivalence therefore involves quantifying over all contexts $C[\square]$ in the given programming language.

The specifically used programming language may vary between different papers: e.g., PCFL [42] is a higher-order functional language (with lambda-abstraction) with recursive functions and lazy lists, while the `while` language [16] has a more imperative flavor. These languages also support non-terminating programs: PCFL contains a fixed-point operator (which can e.g. be used to produce potentially infinite data structures such as infinite lists) and `while` supports while-loop constructions.

Since contextual equivalence is built on a quite different mathematical machinery than ground convertibility, it is hard to directly compare them (doing this properly is definitely beyond the scope of this thesis). However, for terminating programs, we can approximately express contextual equivalence using the terminology of term rewriting:

$$s \sim_{\text{ctx}} t \quad \text{iff} \quad \text{For every normal form } n, \text{ context } C \text{ and substitution } \gamma \text{ in an arbitrary signature extension we have: } C[s\gamma] \rightarrow_{\mathcal{R}}^* n \iff C[t\gamma] \rightarrow_{\mathcal{R}}^* n$$

Although contextual equivalence does not involve quantification over arbitrary signature extensions (the term formation signature is fixed), languages with a fixed-point operator allow new recursive functions to be created on the spot and therefore likely include a form of extensibility. It is shown that contextual equivalence is also extensional [42].

Proving contextual equivalence directly is challenging, requiring induction proofs on the term structure where finding induction hypotheses can be complicated. To overcome this issue, several proof methods have been developed, such as denotational semantics and coinductive methods based on bisimilarity [42]. Semantic methods have

also been developed for proving weakened versions of contextual equivalence (without quantifying over all contexts) [2, 16], which may be better suited to certain applications.

Contextual equivalence and related proof methods constitute relevant contributions to the field of program equivalence, but are beyond the scope of this thesis. It remains future work to investigate to what degree rewriting induction can be unified with contextual equivalence: this likely requires weakening the termination requirements, for which the Bounded RI approach in Chapter 5 may be helpful.

3.3.3 Oracle extensions

We now aim to extend HORI in such a way that it proves equations to be *global* inductive theorems. Largely, this is straightforward, as we can mostly ignore rules whose defined symbols do not occur inside the equation. However, a major problem arises with the (Expansion) rule: we now have to prove termination of $\mathcal{R}' \cup \mathcal{H}$ for *any* natural extension $\mathcal{R}'$ of $\mathcal{R}$. This is in general not possible.

To handle this issue, we use a specific, more manageable kind of extension.

Definition 3.3.10 (Oracle extension). An *oracle extension* of an LCSTRS $\mathcal{L} = (\Sigma, \mathcal{R})$ is a quasi-reductive and terminating natural extension $\mathcal{Q} = (\Sigma^{\mathcal{Q}}, \mathcal{R}^{\mathcal{Q}})$ such that all rules in $\mathcal{R}^{\mathcal{Q}} \setminus \mathcal{R}$ have a form $f \ v_1 \cdots v_m \rightarrow w$, where $f :: \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota$, with $\iota \in \mathcal{S}^{\mathcal{Q}}$ and

- (i). v_i is a ground term, for all $1 \leq i \leq m$
- (ii). w is a ground semi-constructor term over $\mathcal{Q}$ that contains no defined symbols of $\mathcal{R}$

Thus, an oracle extension adds functions that, given ground arguments, compute a semi-constructor result in exactly one step. Moreover, their right-hand sides do not use defined symbols in $\mathcal{R}$, thus removing any dependency. We call the rules of $\mathcal{R}^{\mathcal{Q}} \setminus \mathcal{R}$ *oracle rules*. There are typically infinitely many.

The idea is that a natural extension $\mathcal{L}'$ of $\mathcal{L}$ may be translated into an oracle extension by taking, for every defined symbol f of $\mathcal{R}' \setminus \mathcal{R}$ and ground terms $v_1, \dots, v_m$, the normal form w of $f \ v_1 \cdots v_m$, and including $f \ v_1 \cdots v_m \rightarrow w$ as a rule. To ensure that the right-hand sides of the oracle rules do not contain defined symbols of $\mathcal{R}$, we use fresh (primed) copies of the non-constructor symbols of the natural extension. The full construction is in Section 3.3.6, but Example 3.3.11 illustrates the idea.

Example 3.3.11. The natural extension $(\Sigma', \mathcal{R}')$ in Example 3.3.8 gives rise to oracle rules by taking ground instances of $\text{double } x \rightarrow \text{add } x \ x$ and normalizing the resulting right-hand side to a semi-constructor term, using a fresh copy double' . For example, we include $\text{double}' \ (s \ 0) \rightarrow s \ (s \ 0)$ because $\text{double} \ (s \ 0) \rightarrow_{\mathcal{R}}^* s \ (s \ 0)$. In contrast to the rules in $\mathcal{R}'$, the right-hand sides of the oracle rules do not contain add . Thereby the oracle rules have no $\mathcal{D}$ -dependency on the original system.

So intuitively, in terms of programming, while a natural extension may fully implement a function imported from an external module, an oracle extension deliberately hides this implementation, directly specifying its input-output behaviour.

Lemma 3.3.12. An equation $s \approx t [\varphi]$ over a terminating, strong quasi-reductive LCSTRS $\mathcal{L}$ is a global inductive theorem of $\mathcal{L}$ if for every oracle extension Q and ground substitution γ over Q that respects this equation: $s\gamma \leftrightarrow_{\mathcal{R}^Q}^* t\gamma$.

3.3.4 Global Rewriting Induction

We now update HORI in such a way that we can prove global inductive theorems of terminating, strong quasi-reductive LCSTRSs. Since equations and rules only contain function symbols from the original signature, the (Simplify) rule is unchanged. The (Delete) and (Semi-constructor) rules are also the same. For (Alter) and (Generalize), we now quantify over all gsc substitutions in the extended signature, but scenarios I–III all still apply. Hence, the only rule that changes is (Expansion).

(Global Expansion) Let $s \approx t [\varphi]$ be an equation and $p \in \text{Pos}_F(s)$ a position such that $s|_p = f s_1 \cdots s_n$ with $f \in \mathcal{D}$, $n \geq k = ar(f)$ and for all $1 \leq i \leq k$, $q \in \text{Pos}_F(s_i)$: if $s_i|_q$ has base type and is not a variable, then $s_i|_q$ has a form $c t_1 \cdots t_m$ with c a constructor symbol. If $\mathcal{R}^Q \cup \mathcal{H} \cup \{s \rightarrow t [\varphi]\}$ is terminating for every oracle extension Q of $\mathcal{L}$ then

$$(\mathcal{E} \uplus \{s \approx t [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E} \cup \text{Expd}(s \approx t [\varphi], p), \mathcal{H} \cup \{s \rightarrow t [\varphi]\})$$

Compared to the original (Expansion) rule, the requirement on the shape of the s_i is weaker than before; this is possible due to the strong quasi-reductivity requirement. While the termination requirement is harder to check, this could be done either through dynamic dependency pairs [36, 37] (since the oracle rules do not generate any dependency pairs), or, if certain (reasonable) restrictions on the original system are satisfied, using static dependency pairs. An automated variation of the latter approach is available for LCSTRSs [24]. We use oracle extensions, rather than arbitrary (terminating, strong quasi-reductive) natural extensions, because the extra rules do not depend on the defined symbols of $\mathcal{L}$, which is what makes it feasible to prove termination results.

Definition 3.3.13 (Global RI). We define *Global Rewriting Induction* (Global RI) as the proof system obtained from the following deduction rules

- (Simplify), (Delete) and (Semi-constructor) the same as in HORI
- (Generalize) and (Alter) where the occurring gsc substitutions are quantified over all signatures of all oracle extensions

- (Global Expansion)

Theorem 3.3.14 (Soundness of Global RI). Let $\mathcal{L}$ be a terminating, strong quasi-reductive LCSTRS and let $\mathcal{E}$ be a set of equations. If, by Global RI, $(\mathcal{E}, \emptyset) \vdash^* (\emptyset, \mathcal{H})$, for some set $\mathcal{H}$, then every equation in $\mathcal{E}$ is a global inductive theorem of $\mathcal{L}$.

Before proving this, we first comment on how to prove oracle termination in practice.

3.3.5 Proving oracle termination

In Section 3.3.4, we claimed that static dependency pairs could be used to prove termination of all oracle extensions of a given LCSTRS. To see this, note that static dependency pairs are based around the notion of *computability*. Following the definitions for LCSTRSs [25] (which are based on older techniques for unconstrained systems in the literature), we assume given a quasi-ordering $\sqsupseteq$ on the sorts, whose strict part $\sqsubset$ is well-founded, and define the following relation between sorts and arbitrary types

$$\begin{aligned} \iota \sqsupseteq^+ \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \kappa & \text{ if } \iota \sqsupseteq \kappa \wedge \forall i \in \{1, \dots, m\}. \iota \sqsubset^- \sigma_i \\ \iota \sqsubset^- \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \kappa & \text{ if } \iota \sqsubset \kappa \wedge \forall i \in \{1, \dots, m\}. \iota \sqsupseteq^+ \sigma_i \end{aligned}$$

For all symbols $f :: \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota$, we choose a set $\text{Acc}(f) \subseteq \{1, \dots, m\}$ such that $\iota \sqsupseteq^+ \sigma_i$ for all $i \in \text{Acc}(f)$. We let $g \ s_1 \cdots s_n \succeq_{\text{Acc}} t$ if $t = g \ s_1 \cdots s_n$ or $s_i \succeq_{\text{Acc}} t$ for some $i \in \text{Acc}(g)$. With this, we can define a computability predicate on terms such that

- (i). if $s :: \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota$, then s is computable if and only if $s \ t_1 \cdots t_m$ is computable for all computable terms $t_i :: \sigma_i$
- (ii). if $s :: \iota$, then s is computable if and only if
 - for all t such that $s \rightarrow_{\mathcal{R}} t$ also t is computable
 - if $s :: f \ s_1 \cdots s_m$, then for all $i \in \text{Acc}(f)$ also s_i is computable.
- (iii). a computable term is guaranteed to be terminating

Let $\mathcal{L}'$ be a natural extension of $\mathcal{L}$. Given a sort ordering and accessibility function Acc , the static dependency pair framework can be used to prove that *if* all function symbols in $\Sigma' \setminus \Sigma$ are computable, then so are all function symbols in Σ , without further knowledge of Σ' or $\mathcal{R}'$. This proves termination of the extension using the method and assumptions on the extension described in existing work on termination of LCSTRSs [25]. Thus, to use this method, we only have to prove computability of all oracle symbols.

Now, we assume given an oracle extension $Q = (\Sigma^Q, \mathcal{R}^Q)$ of $\mathcal{L}$, and a sort ordering and accessibility function on the sorts and function symbols of the original signature $\mathcal{L}$. We impose the additional restriction on the choice of $\sqsupseteq$ and Acc that for any constructor

$c :: \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota$ of the original signature, if $i \in \text{Acc}(c)$ and σ_i is an arrow type $\tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow \kappa$, then $\iota \sqsupset \kappa$. This restriction still allows us to for instance use lists of functions. We then assign $\text{Acc}(f) = \emptyset$ for all $f \in \Sigma^Q \setminus \Sigma$, and extend the computability notion to terms over Q . This gives us the following property.

Lemma 3.3.15. Fix a sort ι . Suppose that for all $f :: \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \kappa$ in $\Sigma^Q \setminus \Sigma$ with $\iota \sqsupset \kappa$, and all terminating ground terms $s_1 :: \sigma_1, \dots, s_m :: \sigma_m$, the term $f s_1 \dots s_m$ is computable. Let s be a ground semi-constructor term of type ι' that is equal to ι in the sort ordering. Then s is computable.

Proof. By induction on the size of s . Since s is a ground semi-constructor term of base type, we can write $s = c s_1 \dots s_m$ with $c :: \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota'$ a constructor. Since a gsc term does not reduce, we only need to show that s_i is computable for $i \in \text{Acc}(c)$. If c is in $\Sigma^Q \setminus \Sigma$, this is immediate because then $\text{Acc}(c) = \emptyset$; so assume that $c \in \Sigma$. Let $i \in \text{Acc}(c)$, so $\kappa \sqsupset^+ \sigma_i$ by definition of Acc . If σ_i is a sort that is equal to ι in the sort ordering, then we conclude computability of s_i by the induction hypothesis. Otherwise, write $s_i = g u_1 \dots u_n$ with $g :: \tau_1 \rightarrow \dots \rightarrow \tau_p \rightarrow \kappa$, with $\sigma_i = \tau_{n+1} \rightarrow \dots \rightarrow \tau_p \rightarrow \kappa$. Then due to the additional restriction (or because $n = p$ and we already covered the “both sorts are equivalent” case) $\iota \sqsupset \kappa$. We observe that all u_j are ground semi-constructor terms, and therefore terminating. Moreover, all computable $u_{n+1} :: \tau_{n+1}, \dots, u_p :: \tau_p$ are terminating by the computability notion. Hence, by the induction hypothesis, $g u_1 \dots u_p$ is computable for all computable $u_{n+1}, \dots, u_p$, which implies computability of $g u_1 \dots u_p = s_i$ as desired. $\square$

This allows us to prove computability of all the extra function symbols.

Lemma 3.3.16. If $f :: \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota \in \Sigma^Q \setminus \Sigma$, and $s_1 :: \sigma_1, \dots, s_m :: \sigma_m$ are computable terms, then $f s_1 \dots s_m$ is computable.

Proof. Let $s_1, \dots, s_m$ be terminating terms under $\rightarrow_{RQ}$ (by definition of computability, computable terms are terminating), and prove that $f s_1 \dots s_m$ is computable by induction first on ι (ordered with $\sqsupset$), second by $(s_1, \dots, s_m)$ (ordered placewise with $\rightarrow_{RQ}$).

Since $\text{Acc}(f) = \emptyset$ and s has base type, we only need to prove that t is computable whenever $s \rightarrow_{RQ} t$. If a reduction step is taken in one of the s_i , then we complete by the second induction hypothesis. Otherwise, t is a ground semi-constructor term of type ι and we complete by Lemma 3.3.15 (since the induction hypothesis gives the prerequisites to apply the lemma). $\square$

3.3.6 Proof: Soundness of Global RI

In this section we will prove Theorem 3.3.14. This follows a very similar outline as the soundness proof of HORI: for an arbitrary oracle extension Q of $\mathcal{L}$ we use induction

on $\rightarrow_{\mathcal{R}^Q \cup \mathcal{H}}$ to prove that $\leftrightarrow_{\mathcal{E}} \subseteq \leftrightarrow_{\mathcal{R}^Q}^*$. With Lemma 3.3.12, we then obtain the desired result of ground convertibility in any quasi-reductive and terminating natural extension. Compared to the proof of HORI, the most important changes are:

- In every deduction step, we consider the oracle extension rather than $\mathcal{L}$ directly.
- To prove that (Global Expansion) maintains the invariants, we use the definition of strong quasi-reductivity to show that a ground base-type term of the shape $f s_1 \cdots s_n$ (with $f \in \mathcal{D}$) can be reduced at the root if $s_i|_q$ has a constructor as head symbol whenever $s_i|_q$ has base type.

The content of this section is structured in three parts

- (1). We first show how a terminating quasi-reductive natural extension of a strong quasi-reductive LCSTRS can be turned into an oracle extension. From this, we will prove Lemma 3.3.12.
- (2). We then introduce a similar invariant as for HORI. The difference is that every occurrence of $\mathcal{R}$ is replaced by $\mathcal{R}^Q$, for arbitrary *oracle extension* Q of $\mathcal{L}$.
- (3). Finally, we show that the invariant is indeed satisfied by every deduction rule.

Part (1): Lemma 3.3.12

Towards a proof of Lemma 3.3.12, we show how an oracle extension can be constructed from an arbitrary terminating quasi-reductive natural extension of a strong quasi-reductive LCSTRS.

Definition 3.3.17 (Oracle variant of a natural extension). Let $\mathcal{L}$ be an LCSTRS and $\mathcal{L}'$ be a quasi-reductive, terminating natural extension of $\mathcal{L}$. We define the *oracle variant* Q_{orc} of $\mathcal{L}'$, using the following construction:

Let

$$\mathcal{P} = \left\{ (f v_1 \cdots v_m, w) \mid \begin{array}{l} f \in \Sigma' \\ f \text{ is a defined or calculation symbol in } \mathcal{L}' \\ \text{typeof}'(f) = \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota \ (\iota \in \mathcal{S}') \\ v_1, \dots, v_m, w \text{ are ground terms} \\ f v_1 \cdots v_m \rightarrow_{\mathcal{R}'}^+ w \text{ in } \mathcal{L}' \\ w \text{ is a normal form in } \mathcal{L}' \end{array} \right\}$$

and

- $\mathcal{S}^Q = \mathcal{S}'$ and $\mathcal{V}^Q = \mathcal{V}'$
- $\mathcal{S}_{\text{theory}}^Q = \mathcal{S}_{\text{theory}}$ and $\Sigma_{\text{theory}}^Q = \Sigma_{\text{theory}}$ and $\mathcal{I}^Q = \mathcal{I}$ and $\llbracket \cdot \rrbracket^Q = \llbracket \cdot \rrbracket$
So we use the theory of the *original* signature, not the extension.

- $\Sigma_{terms}^Q = \Sigma_{terms} \cup \{f' \mid f \in \Sigma'_{terms} \setminus C\}$
So there is a symbol f' for each $f \in \Sigma'_{terms}$ that is not a constructor in the original LCSTRS $\mathcal{L}$.
- $typeof^Q(x) = typeof'(x)$ for $x \in \mathcal{V}'$
 $typeof^Q(f) = typeof'(f) = typeof(f)$ for $f \in \Sigma_{terms}$
 $typeof^Q(f') = typeof'(f)$ for $f \in \Sigma'_{terms} \setminus C$

For a term $s \in T(\Sigma^Q, \mathcal{V}^Q)$, we define $\zeta(s) \in T(\Sigma', \mathcal{V}')$ as the term obtained from s by replacing each f' by the corresponding f . For $t \in T(\Sigma', \mathcal{V}')$ or in $T(\Sigma^Q, \mathcal{V}^Q)$, we define $\chi(t) \in T(\Sigma^Q, \mathcal{V}^Q)$ as the term obtained from t by replacing each f that occurs in $\Sigma'_{terms} \setminus C$ by f' (note that all elements of C are in Σ^Q , so leaving them unchanged does not cause problems). Note that $\zeta(\chi(t)) = t$ if $t \in T(\Sigma', \mathcal{V}')$, and $\chi(\zeta(s)) = \chi(s)$ for $s \in T(\Sigma^Q, \mathcal{V}^Q)$. We define

$$\mathcal{R}_{orc} = \left\{ f' s_1 \cdots s_m \rightarrow \chi(w) \mid \begin{array}{l} s_1, \dots, s_m, w \in T(\Sigma^Q, \emptyset) \\ (f \zeta(s_1) \cdots \zeta(s_m), w) \in \mathcal{P} \end{array} \right\}$$

The oracle variant of $\mathcal{L}'$ is the LCSTRS $\mathcal{Q}_{orc}$ generated by $\mathcal{S}^Q, \mathcal{V}^Q, \mathcal{S}_{theory}^Q, \Sigma_{theory}^Q, I^Q, \llbracket \cdot \rrbracket^Q, \Sigma_{terms}^Q, typeof^Q$ and rules $\mathcal{R} \cup \mathcal{R}_{orc}$.

We now first prove several lemmas, to eventually prove Lemma 3.3.12.

Lemma 3.3.18. Suppose $\mathcal{L}'$ is terminating and quasi-reductive. Then the oracle variant of $\mathcal{L}'$ is also a natural extension of $\mathcal{L}$.

Proof. We will check that the requirements of Definition 3.3.7 hold. By definition, all inclusions of (a) are satisfied. Furthermore, $\mathcal{R}_{orc}$ only defines symbols f' , which do not occur in $\mathcal{L}$, so (d) holds. It is easy to see that (b) and (c) hold. Also (e) holds: for all symbols $g :: \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota$, $\zeta(g)$ is a function symbol in $\mathcal{L}'$, so for $1 \leq i \leq m$ we have that σ_i is inhabited in $\mathcal{L}'$; but if $u :: \sigma_i$ is a ground term in $\mathcal{L}$ then $\chi(u)$ is a ground term of type σ_i in $\mathcal{Q}_{orc}$. Finally, we show (f): assume $c :: \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota$ is a constructor of $\mathcal{Q}_{orc}$. There are three possibilities

- If c occurs in Σ_{terms} , then c is also a constructor wrt $\mathcal{L}$.
- If $c = f'$ for some $f \in \Sigma'_{terms} \setminus C$ and f is a constructor of Σ'_{terms} , then ι cannot occur in $\mathcal{S}$ because $\mathcal{L}'$ is a natural extension of $\mathcal{L}$.
- If $c = f'$ for some $f \in \Sigma'_{terms} \setminus C$ that is *not* a constructor of Σ'_{terms} , then we obtain a contradiction, as f' cannot be a constructor of $\mathcal{R}_{orc}$. To see this, note that by the assumption on inhabitation, there exist ground terms $s_1 :: \sigma_1, \dots, s_m :: \sigma_m$, and by quasi-reductivity of $\mathcal{L}'$, the term $f s_1 \cdots s_m$ must reduce. By termination, it has a normal form w , so $(f s_1 \cdots s_m, w) \in \mathcal{P}$. Therefore, $f' \chi(s_1) \cdots \chi(s_m) \rightarrow \chi(w)$ is a rule of $\mathcal{R}_{orc}$. Hence, f' is a defined symbol of $\mathcal{R}_{orc}$, giving the required contradiction. $\square$

Lemma 3.3.19. If $s \rightarrow_{\mathcal{R} \cup \mathcal{R}_{\text{orc}}} t$ in $\mathcal{Q}_{\text{orc}}$ (s, t ground), then $\zeta(s) \rightarrow_{\mathcal{R}}^+ \zeta(t)$ in $\mathcal{L}'$.

Proof. If $s \rightarrow_{\mathcal{R} \cup \mathcal{R}_{\text{orc}}} t$, then $s = C[s']$, $t = C[t']$, and one of the following holds:

- $s' = \ell\gamma$ and $t' = r\gamma$ for some $\ell \rightarrow r [\varphi] \in \mathcal{R}$ and substitution γ that respects this rule. Then since the rules of $\mathcal{R}$ do not use any of the copied symbols ℓ' , we have $\zeta(s') = \ell\gamma^\zeta$ and $\zeta(t') = r\gamma^\zeta$, where $\gamma^\zeta(x) = \zeta(\gamma(x))$. And since $\zeta(v) = v$ for any value v (as values are in $\Sigma_{\text{theory}}^Q = \Sigma_{\text{theory}}$), we have that γ^ζ also respects $\ell \rightarrow r [\varphi]$. Thus, $\zeta(s') = \ell\gamma^\zeta \rightarrow_{\mathcal{R}} r\gamma^\zeta = \zeta(t')$ in $\mathcal{L}'$.
- $s' \rightarrow t' \in \mathcal{R}_{\text{orc}}$, so $\mathcal{P}$ contains a pair $(\zeta(s'), w)$ with $t' = \chi(w)$, which implies $w = \zeta(t')$. By definition of $\mathcal{P}$, $\zeta(s') \rightarrow_{\mathcal{R}}^+ \zeta(t')$.
- $s' = f v_1 \cdots v_m$ for $f :: \iota_1 \rightarrow \dots \rightarrow \iota_m \rightarrow \kappa$ a calculation symbol and all v_i values, and t' is the value of this theory term; since $\Sigma_{\text{theory}}^Q = \Sigma_{\text{theory}} \subseteq \Sigma'_{\text{theory}}$, we have $\zeta(s') = s' \rightarrow_{\mathcal{R}} t' = \zeta(t')$ by a calculation rule.

In all cases, we immediately obtain $\zeta(s) = \zeta(C)[\zeta(s')] \rightarrow_{\mathcal{R}}^+ \zeta(C)[\zeta(t')] = \zeta(t)$. $\square$

Corollary 3.3.20. If $\mathcal{L}'$ is terminating, so is $\mathcal{Q}_{\text{orc}}$.

Lemma 3.3.21. If $\mathcal{L}$ is strong quasi-reductive and $\mathcal{L}'$ is both terminating and quasi-reductive, then $\mathcal{Q}_{\text{orc}}$ is quasi-reductive.

Proof. We prove by induction on the size of a ground term s in $\mathcal{Q}_{\text{orc}}$: if s does not reduce in $\mathcal{Q}_{\text{orc}}$, then s is a semi-constructor term. We can always denote $s = g s_1 \cdots s_n$ with $g :: \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota$ ($m \geq n$, $\iota \in \mathcal{S}'$). If s does not reduce, then neither do its arguments s_i , so by the induction hypothesis, these are semi-constructor terms. We consider the possibilities:

- If g is a constructor symbol in $\mathcal{R} \cup \mathcal{R}_{\text{orc}}$, then we are immediately done.
- If g is defined in $\mathcal{R}_{\text{orc}}$ but $n < m$ then we are done, since all left-hand sides of rules in $\mathcal{R}_{\text{orc}}$ are maximally applied.
- If g is defined in $\mathcal{R}_{\text{orc}}$ and $n = m$, then $g = f'$ for some defined or calculation symbol $f \in \mathcal{L}'$. Either way, note that $\zeta(s) = f \zeta(s_1) \cdots \zeta(s_n)$ is not a semi-constructor term in $\mathcal{L}'$, so by quasi-reductivity of $\mathcal{L}'$ it reduces; by termination there is a normal form w . But then $s \rightarrow \chi(w) \in \mathcal{R}_{\text{orc}}$, so s must reduce.
- If g is an undefined calculation symbol, then we are immediately done if $n < m$. If $n = m$, then note that $g \in \Sigma_{\text{theory}}^Q = \Sigma_{\text{theory}}$, so $\zeta(s) = g \zeta(s_1) \cdots \zeta(s_n)$. By definition of calculation symbol, the arguments of g all have a theory sort, so for $1 \leq i \leq m$, s_i has a form $c t_1 \cdots t_l$ with c a constructor symbol in $\mathcal{Q}_{\text{orc}}$ (since s_i is a ground semi-constructor term of base type), so $\zeta(s_i) = \zeta(c) \zeta(t_1) \cdots \zeta(t_l)$.

Claim: $\zeta(c)$ is a constructor in $\mathcal{L}'$.

This means that any reduct of $\zeta(s_i)$ still has $\zeta(c)$ as root symbol. Since $\mathcal{R}'_g = \mathcal{R}_g$, there are no rules in $\mathcal{R}'$ to reduce $g(\zeta(s_1) \downarrow_{\mathcal{R}'} \cdots (\zeta(s_m) \downarrow_{\mathcal{R}'}))$ and yet it is not a semi-constructor term; which means that the calculation rule must apply, so $\zeta(c)$ can only be a value. But if $\zeta(c)$ is a value, then c is a value, so we see that all s_i are values. Hence, $s = g s_1 \cdots s_m$ reduces (using a calculation rule).

To prove the claim, suppose that $\zeta(c)$ is a calculation symbol or defined symbol; as $\zeta(s_i)$ has base type, it is clearly not a semi-constructor term, so $\zeta(s_i)$ must reduce by quasi-reductivity; and due to the termination requirement we know that $(\zeta(s_i), w) \in \mathcal{P}$ for some w , and hence that c is a defined symbol in $\mathcal{R}_{\text{orc}}$ – and therefore s_i cannot be a semi-constructor term in $\mathcal{Q}_{\text{orc}}$.

- Finally, if g is defined in $\mathcal{R}$, then we are done if $n < ar(g)$. If $n \geq ar(g)$, then consider $\mu\zeta(s) = g \mu(\zeta(s_1)) \cdots \mu(\zeta(s_n))$. The sorts occurring in the argument types of g are all in $\mathcal{S}$, as are the sorts occurring in the argument types of any constructor of $\mathcal{L}$; therefore, $\mu(\zeta(s_i)) \in T(\Sigma, \mathcal{V})$ for all i – and $\mu(\zeta(s_i))$ is a CHV term.

Now observe that for constructors c in Σ_{terms} , there is no marked version c' ; hence, $\mu(s_i) = \mu(\zeta(s_i))$, and $s_i = \mu(s_i)[F_p := s_i|_p \mid p \in \text{Pos}_F(s_i) \wedge s_i|_p \text{ has higher type}]$.

By strong quasi-reductivity, there is a rule in $\mathcal{R}$ that reduces $g \mu(s_1) \cdots \mu(s_n)$ at the head. But then the same rule also reduces its instance $g s_1 \cdots s_n$ in $\mathcal{Q}_{\text{orc}}$. $\square$

Lemma 3.3.22. Suppose $\mathcal{L}$ is strong quasi-reductive and $\mathcal{L}'$ is terminating and quasi-reductive. Then the oracle variant of $\mathcal{L}'$ is an oracle extension of $\mathcal{L}$.

Proof. By Lemma 3.3.18, it is a natural extension. By Corollary 3.3.20, it is terminating. By Lemma 3.3.21, it is quasi-reductive.

All rules in $\mathcal{R}_{\text{orc}}$ have a shape $f' s_1 \cdots s_m \rightarrow \chi(w)$ with f' a defined symbol of $\mathcal{R}_{\text{orc}}$ and all s_i and $\chi(w)$ ground. By definition of χ , the only symbols in $\chi(w)$ are constructors and symbols f' , so not defined symbols of $\mathcal{L}$. Since w is a ground normal form, by quasi-reductivity it is a semi-constructor term, and since $ar(f) = ar^Q(f)$ for $f \in \Sigma_{\text{terms}}$, and $ar(f) \leq ar^Q(f')$ for $f \in \Sigma'_{\text{terms}} \setminus C$, this means $\chi(w)$ is a semi-constructor term for $\mathcal{Q}$. $\square$

Lemma 3.3.23. Let $s \approx t[\varphi]$ be an equation over $\mathcal{L}$, and γ a ground substitution over $\mathcal{L}'$ that respects this equation. Then γ^χ also respects the equation, and if $s\gamma^\chi \leftrightarrow_{\mathcal{R} \cup \mathcal{R}_{\text{orc}}}^* t\gamma^\chi$, then $s\gamma \leftrightarrow_{\mathcal{R}}^* t\gamma$.

Proof. For γ to respect the equation, three things should be satisfied

- The domain of γ should include all variables occurring in the equation. This is clearly also satisfied for γ^χ (as this has the same domain).

- For all variables x in the constraint φ , $\gamma(x)$ must be a value. But note that the variables in the constraint have a sort occurring in the original signature, so (since no new constructors of the original sorts are added, which implies no new values) $\gamma(x) \in C$, so $\gamma^x(x) = \gamma(x)$.
- $\llbracket \varphi \gamma \rrbracket = \top$; since $\gamma^x(x) = \gamma(x)$ for all $x \in \text{Var}(\varphi)$, clearly also $\llbracket \varphi \gamma^x \rrbracket = \top$.

Now suppose $s\gamma^x \leftrightarrow_{\mathcal{R} \cup \mathcal{R}_{\text{orc}}}^* t\gamma^x$. We see by Lemma 3.3.19 that $\zeta(s\gamma^x) \leftrightarrow_{\mathcal{R}'}^* \zeta(t\gamma^x)$, and since $\zeta(s\gamma^x) = s\gamma$ and $\zeta(t\gamma^x) = t\gamma$, this is exactly what we need. $\square$

Thus, we finally have the prerequisites to prove Lemma 3.3.12.

Lemma 3.3.12. *An equation $s \approx t[\varphi]$ over a terminating, strong quasi-reductive LCSTRS $\mathcal{L}$ is a global inductive theorem of $\mathcal{L}$ if for every oracle extension Q and ground substitution γ over Q that respects this equation: $s\gamma \leftrightarrow_{\mathcal{R}^Q}^* t\gamma$.*

Proof. To prove that the equation is a global inductive theorem of $\mathcal{L}$, let $\mathcal{L}'$ be an arbitrary natural extension of $\mathcal{L}$; we must show that for every ground substitution δ over $\mathcal{L}'$ that respects the equation we have $s\delta \leftrightarrow_{\mathcal{R}'}^* t\delta$.

Let Q_{orc} be the oracle variant of $\mathcal{L}'$. By Lemma 3.3.22, Q_{orc} is an oracle extension of $\mathcal{L}$, and by Lemma 3.3.23, δ^x respects the equation. Hence, by the assumptions in the lemma (choosing δ^x for γ and $\mathcal{R} \cup \mathcal{R}_{\text{orc}}$ for $\mathcal{R}^Q$), $s\delta^x \leftrightarrow_{\mathcal{R} \cup \mathcal{R}_{\text{orc}}}^* t\delta^x$. Again by Lemma 3.3.23, this implies the required property $s\delta \leftrightarrow_{\mathcal{R}'}^* t\delta$. $\square$

Part (2): the invariant

The rest of this section is devoted to the proof of the following statement.

Theorem 3.3.14. *Let $\mathcal{L}$ be a terminating, strong quasi-reductive LCSTRS and let $\mathcal{E}$ be a set of equations. If, by Global RI, $(\mathcal{E}, \emptyset) \vdash^* (\emptyset, \mathcal{H})$, for some set $\mathcal{H}$, then every equation in $\mathcal{E}$ is a global inductive theorem of $\mathcal{L}$.*

By Lemma 3.3.12, it suffices to show that all elements of $\mathcal{E}$ are inductive theorems in an arbitrary oracle extension of $\mathcal{L}$. So let us fix an arbitrary oracle extension $Q = (\Sigma^Q, \mathcal{R}^Q)$ of $\mathcal{L}$. The proof is very similar to the proof of HORI – Theorem 3.2.1 in Section 3.2.3.

- We define $\leftrightarrow_{\mathcal{E}}$ as in Section 3.2.3, but with the difference that C is now a context over Q , and γ a substitution over Q . We do the same for $\leftrightarrow_{\mathcal{E}}^{\parallel}$ and $\leftrightarrow_{\mathcal{E}}^{\text{root, semi}}$.
- We adjust Lemma 3.2.10 to this new setting, to say: if $\leftrightarrow_{\mathcal{E}} \subseteq \leftrightarrow_{\mathcal{R}^Q}^*$ on ground terms over Q , then every equation in $\mathcal{E}$ is an inductive theorem of Q .

Using the same reasoning as in Section 3.2.3 (but with Q in place of $\mathcal{L}$), it is straightforward to check that Theorem 3.3.14 holds if the following invariant holds.

Lemma 3.3.24 (Invariant). Suppose that $(\mathcal{E}, \mathcal{H}) \vdash (\mathcal{E}', \mathcal{H}')$, using Global RI. Then

- (1) $\leftrightarrow_{\mathcal{E} \setminus \mathcal{E}'}^{root, semi} \subseteq \rightarrow_{\mathcal{R}^Q \cup \mathcal{H}'}^* \cdot \leftrightarrow_{\mathcal{E}'} \cdot \leftarrow_{\mathcal{R}^Q \cup \mathcal{H}'}^*$
- (2) $\rightarrow_{\mathcal{R}^Q \cup \mathcal{H}} \subseteq \rightarrow_{\mathcal{R}^Q} \cdot \rightarrow_{\mathcal{R}^Q \cup \mathcal{H}}^* \cdot \leftrightarrow_{\mathcal{E}} \cdot \leftarrow_{\mathcal{R}^Q \cup \mathcal{H}}^*$ on ground terms over Q , implies
 $\rightarrow_{\mathcal{R}^Q \cup \mathcal{H}'} \subseteq \rightarrow_{\mathcal{R}^Q} \cdot \rightarrow_{\mathcal{R}^Q \cup \mathcal{H}'}^* \cdot \leftrightarrow_{\mathcal{E}'} \cdot \leftarrow_{\mathcal{R}^Q \cup \mathcal{H}'}^*$ on ground terms over Q .

Part (3): every deduction rule preserves the invariant

Similar to the proof in Section 3.2.3, we show that Lemma 3.3.24 holds for every deduction rule, but this time we have to work with Q instead of $\mathcal{L}$. Except for (Expansion) – which now becomes (Global Expansion) – it is not difficult to see that the proofs are easily adapted to this new situation: we replace the gsc substitutions over $\mathcal{L}$ by gsc substitutions over Q , and we are allowed to reduce with $\mathcal{R}^Q$ instead of only $\mathcal{R}$. Aside from this

- In (Simplify), we also observe that every rule in $\mathcal{R}$ is also an element of $\mathcal{R}^Q$, and that every calculation rule still exists in Q .
- In (Generalize) and (Alter), we use an adapted version of these derivation rules that quantifies over all gsc substitutions over Q instead of those over $\mathcal{L}$.
- For (Delete), note that if a constraint φ built over the original signature is not satisfiable in $\mathcal{L}$, then it is also not satisfiable in Q : any variable that occurs in φ is in the original sort set $\mathcal{S}$, so by definition of Q being a natural extension, there are no values of this sort in Q that do not also occur in $\mathcal{L}$.
- No additional observations are needed for (Semi-constructor).

Therefore, we only need to show correctness of (Global Expansion).

Similar to (Expansion), we need to prove both (1) and (2) of Lemma 3.3.24. To prove (1), we first introduce two helper lemmas.

Lemma 3.3.25. Let $f s_1 \cdots s_n$ be a ground term over Q such that

- $f \in \mathcal{D}$ (the defined symbols of the *original* LCSTRS) with $n \geq ar(f) = k$
- for all $1 \leq i \leq n$, $q \in \text{Pos}_F(s_i)$: if $s_i|_q$ has base type, then $s_i|_q$ has a form $c t_1 \cdots t_m$ with c a constructor symbol in Q

Then there is a rule $\ell \rightarrow r$ [φ] $\in \mathcal{R} \subseteq \mathcal{R}^Q$ and a substitution γ over Q respecting this rule such that $f s_1 \cdots s_k = l\gamma$.

Proof. Let $s := f s_1 \cdots s_n$, $s' := f \mu_1(s_1) \cdots \mu_n(s_n)$ and $\delta := [F_p := s|_p \mid \text{all variables } F_p \text{ in } \text{Var}(s')]$. Since $f \in \mathcal{D}$, the argument sorts of f occur in the original LCSTRS $\mathcal{L}$, as do the argument sorts of any constructor. So, by definition of a natural extension, $\mu_i(s_i)$ uses

only constructors of $\mathcal{L}$. Therefore, $s'\delta = s$, and s' is a term over $\mathcal{L}$. Note that every $\mu_i(s_i)$ is a CHV term.

Since $s' = f \mu_1(s_1) \cdots \mu_n(s_n)$ is headed by a defined symbol, $n \geq ar(f)$ and all $\mu(s_i)$ are CHV terms over the original signature, strong quasi-reductivity implies that s' reduces in the original LCSTRS, and since the $\mu_i(s_i)$ are constructor terms, it can only reduce at the head. So there is a rule $f \ell_1 \cdots \ell_k \rightarrow r [\varphi] \in \mathcal{R}$ and a substitution η over $\mathcal{L}$ respecting this rule such that $f \mu_1(s_1) \cdots \mu_k(s_k) = f \ell_1 \cdots \ell_k$. We are done choosing $\gamma := \eta\delta$. $\square$

Lemma 3.3.26. If $(\mathcal{E} \uplus \{s \approx t [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E} \cup \text{Expd}(s \approx t [\varphi], p), \mathcal{H} \cup \{s \rightarrow t [\varphi]\})$ by (Global Expansion), and γ is a ground semi-constructor substitution over $\mathcal{Q}$ that respects the equation $s \approx t [\varphi]$, then $s\gamma \rightarrow_{\mathcal{R}} \cdot \leftarrow^*_{\mathcal{E} \cup \text{Expd}(s \approx t [\varphi], p)} t\gamma$.

Proof. Let γ be a ground semi-constructor substitution over $\mathcal{Q}$ that respects $s \approx t [\varphi]$. We have $s|_p = f s_1 \cdots s_n$ for $f \in \mathcal{D}$ (in the original signature), $n \geq ar(f) =: k$ and for all $1 \leq i \leq n$, $q \in \text{Pos}_F(s_i)$: if $s_i|_q$ has base type and is not a variable, then $s_i|_q$ has a form $c t_1 \cdots t_m$ with c a constructor symbol.

But then any subterm $(s_i\gamma)|_q$ of base type is of the shape $c (t_1\gamma) \cdots (t_m\gamma)$ as well, either because $q \in \text{Pos}_F(s_i)$ and $(s_i\gamma)|_q = s_i|_q\gamma$, or because $(s_i\gamma)|_q = \gamma(x)|_{q'}$ for some $x \in \mathcal{V}(s_i)$ and $q' \in \text{Pos}_F(\gamma(x))$ (and γ is a gsc substitution). Therefore, by Lemma 3.3.25, $f (s_1\gamma) \cdots (s_n\gamma) = \ell\delta$ for some $\ell \rightarrow r [\psi] \in \mathcal{R}$ and substitution δ over $\mathcal{Q}$ that respects ψ . Now, we finish the proof exactly as in Lemma 3.2.21. $\square$

Part (1) of Lemma 3.3.24 is now proved by the following.

Corollary 3.3.27. If $(\mathcal{E} \uplus \{s \approx t [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E} \cup \text{Expd}(s \approx t [\varphi], p), \mathcal{H} \cup \{s \rightarrow t [\varphi]\})$ by (Global Expansion) then

$$\begin{aligned} \leftarrow^{\text{root, semi}}_{\{s \approx t [\varphi]\}} &\subseteq \rightarrow^*_{\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t [\varphi]\}} \cdot \leftarrow^*_{\mathcal{E} \cup \text{Expd}(s \approx t [\varphi], p)} \cdot \leftarrow^*_{\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t [\varphi]\}} \\ &\subseteq \rightarrow^*_{\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t [\varphi]\}} \cdot \leftarrow^*_{\mathcal{E} \cup \text{Expd}(s \approx t [\varphi], p)} \cdot \leftarrow^*_{\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t [\varphi]\}} \end{aligned}$$

And part (2) of Lemma 3.3.24 is proved by the following.

Lemma 3.3.28.

Suppose $(\mathcal{E} \uplus \{s \approx t [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E} \cup \text{Expd}(s \approx t [\varphi], p), \mathcal{H} \cup \{s \rightarrow t [\varphi]\})$ by (Global Expansion), and $\rightarrow_{\mathcal{R} \cup \mathcal{H}} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}} \cdot \rightarrow^*_{\mathcal{R} \cup \mathcal{H}} \cdot \leftarrow^*_{\mathcal{E} \cup \{s \approx t [\varphi]\}} \cdot \leftarrow^*_{\mathcal{R} \cup \mathcal{H}}$ on ground terms over $\mathcal{Q}$. Then

$$\rightarrow_{\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t [\varphi]\}} \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{H}} \cdot \rightarrow^*_{\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t [\varphi]\}} \cdot \leftarrow^*_{\mathcal{E} \cup \text{Expd}(s \approx t [\varphi], p)} \cdot \leftarrow^*_{\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t [\varphi]\}}$$

holds on ground terms over $\mathcal{Q}$.

Proof. With a similar reasoning as in the proof of Lemma 3.2.23: because of Corollary 3.3.27 it suffices to show that on ground terms over $\mathcal{Q}$ we have

$$\rightarrow_{\{s \rightarrow t \ [\varphi]\}} \subseteq \rightarrow_{\mathcal{R}^Q} \cdot \rightarrow_{\mathcal{R}^Q \cup \mathcal{H} \cup \{s \rightarrow t \ [\varphi]\}}^* \cdot \leftarrow_{\mathcal{E} \cup \text{Expd}(s \approx t \ [\varphi], p)} \cdot \leftarrow_{\mathcal{R}^Q \cup \mathcal{H} \cup \{s \rightarrow t \ [\varphi]\}}^*$$

So suppose $C[s\gamma], C[t\gamma]$ are ground terms over $\mathcal{Q}$ such that $C[s\gamma] \rightarrow C[t\gamma]$ with $s \rightarrow t \ [\varphi]$ for some ground substitution γ over $\mathcal{Q}$ respecting this rule. Then $\delta = \gamma \downarrow_{\mathcal{R}^Q}$ is a ground semi-constructor substitution over $\mathcal{Q}$, so by Lemma 3.3.26 (and the $\mathcal{Q}$ -variant of Lemma 3.2.12) we have $s\delta \rightarrow_{\mathcal{R}} \cdot \leftarrow_{\text{Expd}(s \approx t \ [\varphi], p)} t\delta$. Therefore,

$$C[s\gamma] \rightarrow_{\mathcal{R}^Q}^* C[s\delta] \rightarrow_{\mathcal{R}} \cdot \leftarrow_{\text{Expd}(s \approx t \ [\varphi], p)} C[t\delta] \leftarrow_{\mathcal{R}^Q}^* C[t\gamma]$$

Hence

$$C[s\gamma] \rightarrow_{\mathcal{R}^Q}^* C[s\delta] \rightarrow_{\mathcal{R}^Q} \cdot \leftarrow_{\text{Expd}(s \approx t \ [\varphi], p)} C[t\delta] \leftarrow_{\mathcal{R}^Q}^* C[t\gamma] \quad \square$$

3.4 Conclusion and future work

HORI provides the first RI system for higher-order rewriting, generalizing RI for LCTRSs via a higher-order generalization of first-order quasi-reductivity. This provides a theoretical foundation for further developments, such as Bounded RI (defined in Chapter 5).

However, HORI is not implemented in a rewriting tool and is not intended for practical use: as discussed in Chapter 5, its induction strategy, like that of the first-order system [22], often imposes unnecessarily strong termination requirements. Bounded RI (implemented in Cora), avoids these issues and strictly extends HORI, since every equation provable by HORI is also provable by Bounded RI.

We propose two directions for future work:

Proving and disproving global inductive theorems Apart from proving inductive theorems, Bounded RI can also disprove equations that are not inductive theorems. However, we do not yet have a rewriting tool for proving global inductive theorems or disproving equations that are not global inductive theorems. A challenging question is whether we can also establish that an equation is not a global inductive theorem when it *is* an inductive theorem. Future work could focus on incorporating Global RI into Bounded RI, and extensions for disproving global inductive theorems.

Extensional RI A second idea is to admit extensionality; that is, to allow an (induction) rule $s \ x_1 \cdots x_n \rightarrow t \ x_1 \cdots x_n \ [\varphi]$ to be used to reduce a term $C[s\gamma]$. This may allow us to develop a corresponding rewriting induction approach for proving extensional inductive theorems. If we restrict constructors to have base-type arguments, we conjecture that such a deduction rule is also sound in our setting (perhaps under additional restrictions like ground confluence).

Chapter 4

Lemma Generation

In Section 3.2.1 we already illustrated the importance of generalizations in RI proofs. For instance, in Example 3.2.4, we used (Generalize) to transform $f\ n \approx 0 + (f\ n)\ [-1 = n - 1]$ into $x \approx 0 + x$. In this case, (Generalize) is used to replace a variable application by a fresh integer variable, which was necessary to eventually remove the equation.

In this chapter, we focus on another use case of (Generalize), namely as a way to introduce induction hypotheses. This we have also already encountered in Section 3.2.1: in Example 3.2.8, we came across the equation

$$(E) \quad \begin{array}{l} (f\ n) + (\text{fold } [+]\ (0 + (f\ m))\ (\text{map } f\ (\text{init } k))) \approx \\ \text{fold } [+]\ ((0 + (f\ n)) + (f\ m))\ (\text{map } f\ (\text{init } k))\ [n > 0 \wedge m = n - 1 \wedge k = m - 1] \end{array}$$

To prove (E), we replaced it by the more general equation $x + (\text{fold } [+]\ (0 + y)\ l) \approx \text{fold } [+]\ ((0 + x) + y)\ l$ so that the subsequent (Expansion) step produces an induction rule $x + (\text{fold } [+]\ y\ l) \rightarrow \text{fold } [+]\ z\ l\ [z = x + y]$ by which we can finish the proof. On the other hand, applying (Expansion) directly to (E) does not produce an applicable induction rule. In this context, we refer to $x + (\text{fold } [+]\ (0 + y)\ l) \approx \text{fold } [+]\ ((0 + x) + y)\ l$ as a *lemma*. In particular, a lemma is always itself an inductive theorem.

So far, we have not addressed the question of how to obtain lemmas. Lemma generation is a core challenge in inductive theorem proving, and will be the central topic of this chapter. Before we dive into lemma generation, we first address the question: how can we tell from an ongoing RI proof that a lemma is needed?

Divergence Consider the RI proof for $\text{sumIter } x \approx \text{sumRec } x$ in the following LCSTRS:

$$\begin{array}{lll} \text{sumIter } x \rightarrow \text{while } x\ 1\ 0 & \text{sumRec } x \rightarrow 0 & [x \leq 0] \\ \text{while } x\ i\ z \rightarrow \text{while } x\ (i + 1)\ (z + i) & [i \leq x] & \text{sumRec } x \rightarrow x + (\text{sumRec } (x - 1))\ [x > 0] \\ \text{while } x\ i\ z \rightarrow z & [i > x] & \end{array}$$

We first apply (Simplify) to obtain $\text{while } x \ 1 \ 0 \approx \text{sumRec } x$. Since the reduction behavior of both sides depends on whether $x \leq 0$ or $x > 0$ holds, we continue with a case analysis: we apply (Expansion) on $\text{sumRec } x$ to start an induction proof. This yields

$$\begin{aligned} \mathcal{E} &= \left\{ \begin{array}{ll} \text{(E1)} & \text{while } x \ 1 \ 0 \approx 0 \quad [x \leq 0] \\ \text{(E2)} & \text{while } x \ 1 \ 0 \approx x + (\text{sumRec } (x - 1)) \quad [x > 0] \end{array} \right\} \\ \mathcal{H} &= \{ \text{(H1)} \quad \text{sumRec } x \rightarrow \text{while } x \ 1 \ 0 \} \end{aligned}$$

Equation (E1) immediately simplifies to $0 \approx 0$, which we remove with (Delete). Equation (E2) can be transformed into (E2)' while $x \ 2 \ 1 \approx x + (\text{sumRec } x')$ $[x > 0 \wedge x' = x - 1]$. We apply (H1) to (E2)' and obtain

$$\mathcal{E} = \{ \text{while } x \ 2 \ 1 \approx x + (\text{while } x' \ 1 \ 0) \quad [x > 0 \wedge x' = x - 1] \}$$

Note that the induction hypothesis effectively removed the dependency on $\mathcal{R}_{\text{sumRec}}$. However, the proof is not yet finished. To continue, we again start a combined case analysis and induction with (Expansion). This yields

$$\begin{aligned} \mathcal{E} &= \left\{ \begin{array}{ll} \text{(E3)} & \text{while } x \ (2 + 1) \ (1 + 2) \approx x + (\text{while } x' \ 1 \ 0) \quad [2 \leq x \wedge x > 0 \wedge x' = x - 1] \\ \text{(E4)} & 1 \approx x + (\text{while } x' \ 1 \ 0) \quad [2 > x \wedge x > 0 \wedge x' = x - 1] \end{array} \right\} \\ \mathcal{H} &= \left\{ \begin{array}{ll} \text{(H1)} & \text{sumRec } x \rightarrow \text{while } x \ 1 \ 0 \\ \text{(H2)} & \text{while } x \ 2 \ 1 \rightarrow x + (\text{while } x' \ 1 \ 0) \quad [x > 0 \wedge x' = x - 1] \end{array} \right\} \end{aligned}$$

Equation (E4) is easily removed. Equation (E3) is transformed into

$$\text{(E3)'} \quad \text{while } x \ 3 \ 3 \approx x + (\text{while } x' \ 2 \ 1) \quad [x \geq 2 \wedge x' = x - 1]$$

From here on, the straightforward application of RI deduction rules does not yield further progress: an additional application of (Simplify) with a rule in $\mathcal{R}$ is not possible, nor can we apply (H2) to reduce the left-hand side of (E3)'. We can still perform an additional (Expansion) but this will again produce a non-applicable induction rule.

What we encounter here is the first stage of a diverging proof: an infinitely repeating cycle of deduction steps, each cycle being initiated by (Expansion), and producing a non-applicable induction rule. After four cycles, $\mathcal{H}$ contains

$$\begin{aligned} &\text{while } x \ 2 \ 1 \rightarrow x + \text{while } x' \ 1 \ 0 \quad [x \geq 1 \wedge x' = x - 1] \\ &\text{while } x \ 3 \ 3 \rightarrow x + \text{while } x' \ 2 \ 1 \quad [x \geq 2 \wedge x' = x - 1] \\ &\text{while } x \ 4 \ 6 \rightarrow x + \text{while } x' \ 3 \ 3 \quad [x \geq 3 \wedge x' = x - 1] \\ &\text{while } x \ 5 \ 10 \rightarrow x + \text{while } x' \ 4 \ 6 \quad [x \geq 4 \wedge x' = x - 1] \end{aligned}$$

Such a sequence of induction rules we call a *divergence pattern*. In the construction of a divergence pattern, the most recently added induction rule always lags one step behind on the proof goal in $\mathcal{E}$. For example,

- After adding the induction rule $\text{while } x \ 2 \ 1 \rightarrow x + \text{while } x' \ 1 \ 0 \ [x \geq 1 \wedge x' = x - 1]$ the proof goal in $\mathcal{E}$ becomes $\text{while } x \ 3 \ 3 \approx x + \text{while } x' \ 2 \ 1 \ [x \geq 2 \wedge x' = x - 1]$.
- After adding the induction rule $\text{while } x \ 3 \ 3 \rightarrow x + \text{while } x' \ 2 \ 1 \ [x \geq 2 \wedge x' = x - 1]$ the proof goal in $\mathcal{E}$ becomes $\text{while } x \ 4 \ 6 \approx x + \text{while } x' \ 3 \ 3 \ [x \geq 3 \wedge x' = x - 1]$.
- And so on . . .

So, in a diverging proof no induction rule in the divergence pattern is ever applicable.

Generalization To stop this infinite (Expansion) loop, we can apply (Generalize) and introduce a lemma that simultaneously generalizes all induction rules in the divergence pattern. Here, we can identify the following lemma

$$\text{while } x \ i \ a \approx x + (\text{while } x' \ j \ b) \ [x \geq j \wedge x' = x - 1 \wedge j = i - 1 \wedge a = b + j]$$

Indeed, the lemma simultaneously generalizes all induction rules in the divergence pattern: they all satisfy $x \geq j$, $x' = x - 1$, $j = i - 1$ and $a = b + j$. We refer to such theory-term expressions as *invariants*. To prove the lemma, we start by (Expansion), producing the induction rule **(H)** $\text{while } x \ i \ a \rightarrow x + (\text{while } x' \ j \ b) \ [x \geq j \wedge x' = x - 1 \wedge j = i - 1 \wedge a = b + j]$. The two corresponding (Expansion) equations reduce to the following

$$\text{(E1)} \quad a \approx x + b \ [i > x \wedge x \geq j \wedge x' = x - 1 \wedge j = i - 1 \wedge a = b + j]$$

$$\text{(E2)} \quad \text{while } x \ i' \ a' \approx x + (\text{while } x' \ j' \ b') \ [i \leq x \wedge i' = i + 1 \wedge a' = a + i \wedge j' = j + 1 \wedge b' = b + j \wedge x \geq j \wedge x' = x - 1 \wedge j = i - 1 \wedge a = b + j]$$

Equation **(E1)** can be removed, since the constraint implies $a = x + b$. For **(E2)**, we apply **(H)** γ with $\gamma = [i := i', a := a', j := j', b := b']$ to reduce the left side to $x + (\text{while } x' \ j' \ b')$, after which the equation can be removed, as both sides are syntactically equal.

Generalization in equational reasoning The necessity of lemmas – or the generalization of equations before they can serve as induction hypotheses – is a well-known topic in equational reasoning [18, 31, 32, 45, 46]. The methods developed in this area are very suitable for unconstrained rewriting where we can reason about programs operating on inductive data structures through pattern matching (like natural numbers and lists). They can also be applied in constrained rewriting. For example, a useful generalization heuristic is to generalize maximal subterms occurring multiple times in an equation [39]. This is exactly what we used to transform $f \ n \approx 0 + (f \ n) \ [-1 = n - 1]$ into $x \approx 0 + x$ by replacing the maximal multiple occurring subterm $f \ n$ with a fresh variable x .

Generalization in constrained rewriting Although we can apply generalization heuristics from equational reasoning in constrained rewriting, they are typically not sufficient. The situation in constrained rewriting is more involved, as we have to deal with constraint-based calculations on non-inductive data structures (theory terms), which are treated semantically. When constructing lemmas in constrained rewriting, the semantical dependencies between theory terms are expressed in the constraint by invariants. For example, we just explained that to prove $\text{sumIter } x \approx \text{sumRec } x$ we use the lemma $\text{while } x \ i \ a \approx x + (\text{while } x' \ j \ b) [x \geq j \wedge x' = x - 1 \wedge j = i - 1 \wedge a = b + j]$, in which the constraint includes the invariants $x \geq j$, $x' = x - 1$, $j = i - 1$ and $a = b + j$.

To the best of our knowledge, only two existing works [22, 39] have studied lemma generation in constrained rewriting. Attempts to adapt or extend existing methods from equational reasoning to constrained rewriting have so far been unsuccessful [39]. This is not entirely surprising, as the validity of an invariant depends on the chosen semantics, hence we cannot expect to generate them by purely syntactically oriented generalization methods. These works [22, 39] constitute two initial attempts at automatic lemma generation in constrained rewriting. In particular, *Initialization Generalization* (InGen) [22] marks a step in the right direction, being able to produce non-trivial invariants. Nevertheless, InGen remains limited in practice, as in many examples it only produces a fraction of the required invariants. Another issue is that InGen may produce non-invariants.

4.1 Initialization Generalization

Initialization Generalization (InGen) [22] is a generalization method in which theory term invariants are derived via a RI deduction, by tracking $\mathcal{R}_{calc}$ reductions. The method replaces the original LCSTRS by an equivalent one, in which all value initializations are replaced by variables. For example, we may replace the rule $\text{sumIter } x \rightarrow \text{while } x \ 1 \ 0$ by the equivalent rule $\text{sumIter } x \rightarrow \text{while } x \ i_0 \ a_0 [i_0 = 1 \wedge a_0 = 0]$ because 1 and 0 are the values used to initialize the while loop $\text{while } x \ i_0 \ a_0$. We do not consider 0 on the right hand side of $\text{sumRec } x \rightarrow 0 [x \leq 0]$ as an initialization, as this merely returns a value.

The intuition behind the InGen strategy is that, by replacing values with variables, the RI deduction is able to *symbolically* derive computational dependencies between the involved theory terms (obtained through the application of $\mathcal{R}_{calc}$ rules), thereby abstracting away from their specific values. The symbolic dependencies are often invariants required to produce a lemma. We will not repeat the full formal discussion of InGen, but informally explain the procedure through Example 4.1.1.

Example 4.1.1. The first published method for lemma generation in constrained rewriting [39] was not able to generate a lemma to prove the equivalence $\text{dblRec } x \approx \text{dblIter } x$ in an LCTRS that is very similar to

$$\begin{array}{ll} \text{dblRec } x \rightarrow 0 & [x \leq 0] \\ \text{dblRec } x \rightarrow (\text{dblRec } (x - 1)) + 2 & [x > 0] \end{array} \quad \begin{array}{ll} \text{dblIter } x \rightarrow u \ x \ 1 \ 0 & \\ u \ x \ i \ z \rightarrow z & [i > x] \\ u \ x \ i \ z \rightarrow u \ x \ (i + 1) \ (z + 2) & [i \leq x] \end{array}$$

The equation results in a diverging RI proof

$$\begin{array}{l} (u \ x_1 \ 1 \ 0) + 2 \rightarrow u \ x \ 2 \ 2 \quad [x > 0 \wedge x_1 = x - 1] \\ (u \ x_1 \ 2 \ 2) + 2 \rightarrow u \ x \ 3 \ 4 \quad [x > 1 \wedge x_1 = x - 1] \\ (u \ x_1 \ 3 \ 4) + 2 \rightarrow u \ x \ 4 \ 6 \quad [x > 2 \wedge x_1 = x - 1] \\ (u \ x_1 \ 4 \ 6) + 2 \rightarrow u \ x \ 5 \ 8 \quad [x > 3 \wedge x_1 = x - 1] \\ \vdots \end{array}$$

We now show that InGen can produce a lemma. The method proceeds in two stages.

Stage 1: Replace value initializations by variables. For every $\ell \rightarrow r \ [\varphi] \in \mathcal{R}$: the rule is transformed into an equivalent one by replacing every value-initialization v in r with a fresh variable x and adding $x = v$ to the constraint

We have value initializations 1 and 0 in the rewrite rule $\text{dblIter } x \rightarrow u \ x \ 1 \ 0$. We replace them by fresh variables i_0 and z_0 , and assign their values $i_0 = 1$ and $z_0 = 0$ in the constraint. This yields the following rule

$$\text{dblIter } x \rightarrow u \ x \ i_0 \ z_0 \quad [i_0 = 1 \wedge z_0 = 0]$$

Stage 2: Apply RI and generalize by removing initializations from the constraint. To obtain a lemma candidate, we start a RI deduction using the adapted LCSTRS, obtained in Stage 1. At any, unspecified, moment we may apply the generalization strategy: remove all initialization-equalities $x = v$ from the constraint of the diverging equation.

We start a RI deduction on the equation $\text{dblRec } x \approx \text{dblIter } x$, and at some point remove $i_0 = 1$ and $z_0 = 0$ from the constraint of the diverging equation, thereby abstracting away from the values 1 and 0, while preserving the symbolic calculations derived from them. The strategy involves a certain ambiguity: *when* in the RI proof should we perform this generalization? There is no general answer to this question, but as a rule of thumb: a minimal requirement is that we should have completed at least one (Expansion) cycle in the divergence pattern.

Here, it suffices to complete precisely one (Expansion) cycle, after which we obtain the following equation, corresponding to the second rule in the divergence pattern

$$\begin{aligned}
& u \ x_1 \ i_1 \ z_1 + 2 \approx u \ x \ i_2 \ z_2 \\
& [i_0 = 1 \wedge z_0 = 0 \wedge x > 0 \wedge x_1 = x - 1 \wedge \\
& i_1 = i_0 + 1 \wedge z_1 = z_0 + 2 \wedge x > 1 \wedge i_2 = i_1 + 1 \wedge z_2 = z_1 + 2]
\end{aligned}$$

Removing the initializations $i_0 = 1$ and $z_0 = 0$ yields the following lemma

$$\begin{aligned}
& u \ x_1 \ i_1 \ z_1 + 2 \approx u \ x \ i_2 \ z_2 \\
& [x > 0 \wedge x_1 = x - 1 \wedge i_1 = i_0 + 1 \wedge z_1 = z_0 + 2 \wedge x > 1 \wedge i_2 = i_1 + 1 \wedge z_2 = z_1 + 2]
\end{aligned}$$

As a final note, InGen does often not create the most esthetically optimized lemma: here, we have redundant invariants $x > 0$, $x > 1$, $i_1 = i_0 + 1$ and $z_1 = z_0 + 2$. Removing them gives us a more user-friendly lemma

$$u \ x_1 \ i_1 \ z_1 + 2 \approx u \ x \ i_2 \ z_2 \quad [x_1 = x - 1 \wedge i_2 = i_1 + 1 \wedge z_2 = z_1 + 2]$$

We will now present an example, where InGen fails to generate a suitable lemma.

Example 4.1.2. We consider $up \ x \approx down \ x$ in the following LCSTRS

$ \begin{aligned} & up \ x \rightarrow u \ x \ 1 \\ & u \ x \ k \rightarrow u \ x \ (k + 1) \quad [k \leq x] \\ & u \ x \ k \rightarrow x \quad [k > x] \end{aligned} $	$ \begin{aligned} & down \ x \rightarrow d \ x \ x \\ & d \ x \ l \rightarrow d \ x \ (l - 1) \quad [l > 0] \\ & d \ x \ l \rightarrow x \quad [l \leq 0] \end{aligned} $
---	---

We obtain a diverging RI proof

$$\begin{aligned}
& u \ x \ 1 \rightarrow d \ x \ x \\
& u \ x \ 2 \rightarrow d \ x \ x_1 \quad [x \geq 1 \wedge x_1 = x - 1] \\
& u \ x \ 3 \rightarrow d \ x \ x_2 \quad [x \geq 2 \wedge x_2 = x - 2] \\
& u \ x \ 4 \rightarrow d \ x \ x_3 \quad [x \geq 3 \wedge x_3 = x - 3] \\
& \vdots
\end{aligned}$$

In Stage 1, InGen generates the rewrite rule $up \ x \rightarrow u \ x \ k_0 \ [k_0 = 1]$. In Stage 2, we remove the initialization $k_0 = 1$ after completing the first divergence cycle. This yields

$$u \ x \ k_1 \approx d \ x \ l \quad [x \geq 1 \wedge k_1 = k_0 + 1 \wedge l = x - 1]$$

which is not a valid lemma. It is easily seen that the equality $l = x - 1$ is not an invariant of the divergence pattern. The reason it is not recognized as a non-invariant is that InGen only considers value initializations. Consequently, the second x in $d \ x \ x$ on the right-hand side of $down \ x \rightarrow d \ x \ x$ is not treated as an initialization, even though it is. It has been shown [17] that InGen can be extended in a way to fix this issue, being able

to produce the initialized rule $\text{down } x \rightarrow \text{d } x \ l_0 \ [l_0 = x]$. If we then remove $k_0 = 1$ and $l_0 = x$ after the first divergence cycle, we obtain

$$\text{u } x \ k_1 \approx \text{d } x \ l_1 \ [x \geq 1 \wedge k_1 = k_0 + 1 \wedge l_1 = l_0 - 1]$$

Nevertheless, although this allows us to remove the non-invariant, this does not yield a lemma. In general, this extension is not very powerful: a serious deficiency is that it fails to discover the only invariant that is actually required, $k_1 + l_1 = x + 1$. (The invariants $x \geq 1$, $k_1 = k_0 + 1$ and $l_1 = l_0 - 1$ are redundant and can be removed.) After renaming $k_1 := k$ and $l_1 := l$, we then obtain the following optimized and user-friendly lemma:

$$\text{u } x \ k \approx \text{d } x \ l \ [k + l = x + 1]$$

Remark: we note that InGen (implicitly) successfully discovered the consistent progression $k \mapsto k + 1$ and $l \mapsto l - 1$ throughout the divergence pattern, via the expressions $k_1 = k_0 + 1$ and $l_1 = l_0 - 1$. However, it requires some additional reasoning to derive the invariant $k + l = x + 1$. From $k \mapsto k + 1$ and $l \mapsto l - 1$ we, by human reasoning, immediately conclude that $k + l$ remains constant throughout the divergence pattern. To see that this constant equals $x + 1$ one may, for example, use the initialization $k_0 = 1$ and $l_0 = x$. A systematic treatment of such symbolic reasoning will be the topic of Section 4.2.

4.2 Matrix Invariants

In this section we present *Matrix Invariants*: an algebraic method for generating loop invariants for integer programs. It applies to non-nested loops in which each loop iteration uses a consistent assignment to update the loop variables. In the setting of rewriting induction, such behavior arises in a *simple divergence pattern*: an infinite sequence of induction rules generated by repeated (Expansion) cycles, each of which produces a rule with the same term structure – called the *divergence shape* – and where the involved loop variables are updated according to a fixed *divergence substitution*. In the examples discussed so far, we only encountered simple divergence patterns.

Simple divergence In Example 4.1.2 we can identify a divergence shape $\text{u } x \ k \approx \text{d } x \ l$ with loop variables k and l . We refer to k and l as *divergence variables*. The first induction rule appearing in the divergence pattern initializes $k := 1$ and $l := x$. In each subsequent (Expansion) cycle, the terms substituted for k and l are consistently updated in accordance with the application of the recursive rules $\text{u } x \ k \rightarrow \text{u } x \ (k + 1) \ [k \leq x]$ and $\text{d } x \ l \rightarrow \text{d } x \ (l - 1) \ [l > 0]$, which increases k by 1 and decreases l by 1, while keeping

x constant. As we will explain in Section 4.2.1, this can be described by a divergence substitution $\chi = [k := k + 1, l := l - 1]$. This substitution is the core ingredient for describing the corresponding simple divergence pattern. Intuitively, the divergence substitution tells us how to compute the next terms to be substituted for k and l from the current terms that were substituted for k and l . For example, from the initialization $k := 1, l := x$ we can compute the next terms by applying χ , as follows:

$$\chi(k)[k := 1] = 1 + 1 \xrightarrow{\mathcal{R}_{calc}} 2 \quad \text{and} \quad \chi(l)[l := x] = x - 1$$

So this yields $k := 2$ and $l := x - 1$. From this, we can compute the next pair of terms

$$\chi(k)[k := 2] = 2 + 1 \xrightarrow{\mathcal{R}_{calc}} 3 \quad \text{and} \quad \chi(l)[l := (x - 1)] = (x - 1) - 1$$

and so on.

Integer theory terms as polynomials We assume a definition of Σ_{theory} in which ground integer theory terms are interpreted as elements in $\mathbb{Z}$, in the standard way. Then it makes sense to interpret non-ground integer theory terms (i.e. terms in $T(\Sigma_{theory}, \mathcal{V})$ of type `int`) as polynomials with coefficients in $\mathbb{Z}$. For example, the theory term $(x - 1) - 1$ can be interpreted as the polynomial $x - 2 \in \mathbb{Z}[x]$ and $x + y + x * y + 1$ can be interpreted as the polynomial $x + y + xy + 1 \in \mathbb{Z}[x, y]$.

This interpretation is not only intuitive, but also very useful, because it opens opportunities for applying algebraic methods to compute invariants by exploiting the algebraic structure of polynomial rings. To explain this, we consider again Example 4.1.2. We combine both initializations $k := 1$ and $l := x$ into a single pair of polynomials $(k_0, l_0) = (1, x) \in \mathbb{Z}[x]^2$. Moreover, we interpret χ as a function $\chi : \mathbb{Z}[x]^2 \rightarrow \mathbb{Z}[x]^2$ defined by $\chi(k, l) = (k + 1, l - 1)$. This description is very convenient from a computational point of view, as it allows us to reason about the evolution of k and l in a purely algebraic way. For example, the second (k, l) -pair is computed directly from the initialization by applying χ , i.e. $(2, x - 1) = \chi(1, x)$. In this way, we also avoid the issue that multiple theory terms may represent the same polynomial. For example, we just showed that the second application of the divergence substitution χ yields $l := (x - 1) - 1$, which does *not* reduce to $x - 2$ with $\mathcal{R}_{calc}$. However, with the interpretation of χ as a function on $\mathbb{Z}[x]^2$, we immediately obtain $\chi(2, x - 1) = (3, x - 2)$.

In particular, the trajectory of (k, l) can now be described as a recurrence relation

$$\begin{cases} (k_0, l_0) = (1, x) \\ (k_i, l_i) = \chi(k_{i-1}, l_{i-1}) \quad \text{for } i > 0 \end{cases}$$

producing an infinite set $\mathcal{Div} = \{(1, x), (2, x-1), (3, x-2), \dots\} \subseteq \mathbb{Z}[x]^2$. The description of a divergence pattern as a subset of $\mathbb{Z}[x]^2$ provides a mathematical setting suitable for applying algebraic methods to compute invariants [3, 15, 30]. In particular, the Matrix Invariant method that we will define is built on this particular interpretation.

4.2.1 Simple divergence

We will now formally define what we mean with a simple divergence pattern. In the remainder of this chapter we assume that theory terms of type `int` only contain calculation symbols from $+$, $-$ and $*$. Then every such term has a polynomial interpretation.

Definition 4.2.1 (Simple divergence). Let $n \geq 1$. A *simple divergence pattern* in n parameters involves

- ▷ A sequence $(\rho_i)_{i \in \mathbb{N}}$ of rewrite rules, called the *divergence pattern*.
- ▷ An equation $\ell \approx r$, called the *divergence shape*.
- ▷ A substitution χ , called the *divergence substitution*.
- ▷ A non-empty subset $\mathcal{V}_{div} = \{x_1, \dots, x_n\} \subseteq \text{Var}(\ell) \cup \text{Var}(r)$ of *divergence variables* such that $\mathcal{V}_{div} = \text{dom}(\chi)$. (That is, $\chi(x) \neq x$ for all $x \in \mathcal{V}_{div}$)
- ▷ A set $\mathcal{V}_{stb}$ of *stable variables* such that $\mathcal{V}_{stb} \cap \mathcal{V}_{div} = \emptyset$.¹

such that

- (1). For all i there exist $y_{1,i}, \dots, y_{n,i} \in \mathcal{V}$ and a constraint φ_i such that

$$\rho_i = \ell[x_1 := y_{1,i}, \dots, x_n := y_{n,i}] \rightarrow r[x_1 := y_{1,i}, \dots, x_n := y_{n,i}] [\varphi_i]$$

- (2). For all i and $1 \leq j \leq n$, there is a term $e_{j,i} \in T(\Sigma_{theory}, \mathcal{V}_{stb})$ such that φ_i contains a conjunct $y_{j,i} = e_{j,i}$.
- (3). Each $e_{j,i+1}$ is obtained from $e_{1,i}, \dots, e_{n,i}$ by applying χ . This is expressed as follows: for every $i \in \mathbb{N}$ and $1 \leq j \leq n$

$$e_{j,i+1} \stackrel{p}{=} \chi(x_j)[x_1 := e_{1,i}, \dots, x_n := e_{n,i}]$$

where $\stackrel{p}{=}$ denotes *polynomial equivalence*, referring to the equivalence relation on $T(\Sigma_{theory}, \mathcal{V}_{stb})$ defined by

$$s \stackrel{p}{=} t \iff s \text{ and } t \text{ represent the same polynomial in } \mathbb{Z}[\mathcal{V}_{stb}]$$

¹Note: we do not necessarily have $\mathcal{V}_{stb} \subseteq \text{Var}(\ell) \cup \text{Var}(r)$. Furthermore, $\mathcal{V}_{stb} = \emptyset$ is allowed.

The equivalence relation $\stackrel{p}{=}$ is an extension of $\leftrightarrow_{\mathcal{R}_{calc}}^*$ which is used, for instance, to treat $x - 2$ and $(x - 1) - 1$ as equivalent terms.

Example 4.2.2. To see that Example 4.1.2 exhibits a simple divergence pattern, we first recast to the following equivalent representation:

$$\begin{aligned}\rho_0 &= u \ x \ k_0 \rightarrow d \ x \ l_0 \quad [k_0 = 1 \wedge l_0 = x] \\ \rho_1 &= u \ x \ k_1 \rightarrow d \ x \ l_1 \quad [k_1 = 2 \wedge l_1 = x - 1 \wedge x \geq 1] \\ \rho_2 &= u \ x \ k_2 \rightarrow d \ x \ l_2 \quad [k_2 = 3 \wedge l_2 = x - 2 \wedge x \geq 2] \\ \rho_3 &= u \ x \ k_3 \rightarrow d \ x \ l_3 \quad [k_3 = 4 \wedge l_3 = x - 3 \wedge x \geq 3] \\ &\vdots\end{aligned}$$

We have divergence shape $u \ x \ k \approx d \ x \ l$, divergence variables $\mathcal{V}_{div} = \{k, l\}$, stable variables $\mathcal{V}_{stb} = \{x\}$, and divergence substitution $\chi = [k := k + 1, l := l - 1]$.

Note that, although the divergence variables k and l do occur in the divergence shape, they do not appear in the divergence pattern. Rather, they are used as placeholders for substituting the variables k_i and l_i . On the other hand, the variable x does appear in the divergence pattern and is never used for substituting other terms (hence remains stable).

We will check the requirements (1), (2) and (3) of Definition 4.2.1

- (1). For all $i \in \mathbb{N}$ there are variables $y_{1,i} = k_i$ and $y_{2,i} = l_i$ and a constraint φ_i such that

$$\rho_i = (u \ x \ k)[k := k_i] \rightarrow (d \ x \ l)[l := l_i] \ [\varphi_i]$$

- (2). We see that for all $i \in \mathbb{N}$ and $1 \leq j \leq 2$, there is a term $e_{j,i} \in T(\Sigma_{theory}, \mathcal{V}_{stb})$ such that φ_i contains a conjunct $y_{j,i} = e_{j,i}$. For example, for $i = 2$ we see that φ_2 contains conjuncts $y_{1,2} = k_2 = 3$ and $y_{2,2} = l_2 = x - 2$. So $e_{1,2} = 3$ and $e_{2,2} = x - 2$.

- (3). We should check that for all i the following holds

$$e_{1,i+1} \stackrel{p}{=} \chi(k)[k := e_{1,i}, l := e_{2,i}] \quad \text{and} \quad e_{2,i+1} \stackrel{p}{=} \chi(l)[k := e_{1,i}, l := e_{2,i}]$$

Using that $\chi(k) = k + 1$ and $\chi(l) = l - 1$ this is equivalent to saying that

$$e_{1,i+1} \stackrel{p}{=} (k + 1)[k := e_{1,i}] \quad \text{and} \quad e_{2,i+1} \stackrel{p}{=} (l - 1)[l := e_{2,i}]$$

These are indeed satisfied: let us demonstrate this for the transition $i = 0$ to $i = 1$. At stage $i = 0$, the theory terms are initialized by $e_{1,0} = 1$ and $e_{2,0} = x$. Therefore

$$e_{1,1} = 2 \stackrel{p}{=} 1 + 1 = (k + 1)[k := e_{1,0}] \quad \text{and} \quad e_{2,1} = x - 1 = (l - 1)[l := e_{2,0}]$$

4.2.2 Divergence space

Finding invariants of a simple divergence pattern is all about finding relations between the terms $e_{j,i}$, for $1 \leq j \leq n$, that hold for all $i \in \mathbb{N}$. Before we continue this discussion, we introduce the notation $\vec{e}_i = (e_{1,i}, \dots, e_{n,i})$ and refer to this as the *divergence vector* at stage i . Then requirement Definition 4.2.1.(3) is represented more compactly

$$\vec{e}_{i+1} \stackrel{p}{=} \chi(\vec{x})[\vec{x} := \vec{e}_i]$$

Here, it is understood that a substitution applied to a vector refers to point wise application, i.e. $\chi(\vec{x}) = (\chi(x_1), \dots, \chi(x_n))$, and $(a_1, \dots, a_n)[\vec{x} := \vec{e}] = (a_1[\vec{x} := \vec{e}], \dots, a_n[\vec{x} := \vec{e}])$.

By requirement (3), the sequence $(\vec{e}_i)_{i \in \mathbb{N}}$ is fully determined once we know $\vec{e}_0$ and χ . By definition, every $\vec{e}_i$ is a vector in $T(\Sigma_{theory}, \mathcal{V}_{stb})^n$, but using the polynomial interpretation, we will identify $\vec{e}_i$ as a vector in $\mathbb{Z}[\mathcal{V}_{stb}]^n$.

Definition 4.2.3 (Divergence space). Let $(\rho_i)_{i \in \mathbb{N}}$ be a simple divergence pattern in n parameters with divergence substitution χ and initial divergence vector $\vec{e}_0$. We define *divergence space*

$$\mathcal{Div} = \{\vec{e}_i \mid i \in \mathbb{N}\}$$

We will identify $\mathcal{Div}$ as a subset of $\mathbb{Z}[\mathcal{V}_{stb}]^n$ and we interpret χ as a function $\chi : \mathbb{Z}[\mathcal{V}_{stb}]^n \rightarrow \mathbb{Z}[\mathcal{V}_{stb}]^n$. Then $\vec{e}_i = \chi^i(\vec{e}_0)$, and therefore $\mathcal{Div} = \{\chi^i(\vec{e}_0) \mid i \in \mathbb{N}\}$.

Example 4.2.4. Continuing on Example 4.2.2: we have initial divergence vector $\vec{e}_0 = (1, x)$ and divergence substitution $\chi = [k := k + 1, l := l - 1]$, yielding the function $\chi : \mathbb{Z}[x]^2 \rightarrow \mathbb{Z}[x]^2$ defined as $\chi(k, l) = (k + 1, l - 1)$. We compute the divergence space

$$\mathcal{Div} = \{ \underbrace{(1, x)}_{\vec{e}_0}, \underbrace{(2, x-1)}_{\vec{e}_1 = \chi(\vec{e}_0)}, \underbrace{(3, x-2)}_{\vec{e}_2 = \chi^2(\vec{e}_0)}, \underbrace{(4, x-3), \dots}_{\vec{e}_3 = \chi^3(\vec{e}_0)} \} \subseteq \mathbb{Z}[x]^2$$

4.2.3 Matrix Invariants

As discussed before, an invariant describes a relation between the entries of the vector $\vec{e}_i = (e_{1,i}, \dots, e_{n,i}) \in \mathbb{Z}[\mathcal{V}_{stb}]^n$ that holds for all i . We may think of such a relation as describing a function $f : \mathbb{Z}[\mathcal{V}_{stb}]^n \rightarrow \mathbb{Z}[\mathcal{V}_{stb}]$ that cannot distinguish between elements from $\mathcal{Div}$. That is, f should be a *constant* function when restricting its domain to $\mathcal{Div}$. Without loss of generality, we can take this constant to be 0.

Definition 4.2.5 (Invariant). Assume a simple divergence pattern in n parameters. An *invariant* is a non-zero function $f : \mathbb{Z}[\mathcal{V}_{stb}]^n \rightarrow \mathbb{Z}[\mathcal{V}_{stb}]$ with $f(\vec{e}) = 0$ for all $\vec{e} \in \mathcal{Div}$.

We will now restrict attention to polynomial invariants. A polynomial invariant $f : \mathbb{Z}[\mathcal{V}_{stb}]^n \rightarrow \mathbb{Z}[\mathcal{V}_{stb}]$ can be represented by a polynomial $f \in \mathbb{Z}[\mathcal{V}_{stb}][\mathcal{V}_{div}]$ where

the variables $\mathcal{V}_{div} = \{x_1, \dots, x_n\}$ are used to insert input from $\mathbb{Z}[\mathcal{V}_{stb}]^n$. For example, the polynomial $f = x + 1 - k - l \in \mathbb{Z}[x][k, l]$ can be seen as a function $f : \mathbb{Z}[x]^2 \rightarrow \mathbb{Z}[x]$ where the variables k and l are used to insert input from $\mathbb{Z}[x]^2$. For example, $\vec{e}_1 = (2, x - 1) \in \mathbb{Z}[x]^2$ is evaluated as $f(2, x - 1) = x + 1 - 2 - (x - 1) = 0$.

For illustrative purposes, we will for the moment restrict to $\mathcal{V}_{div}$ -affine polynomials:

Definition 4.2.6 (Affine invariant). Let $(\rho_i)_{i \in \mathbb{N}}$ be a simple divergence pattern in n parameters, and assume $\mathcal{V}_{div} = \{x_1, \dots, x_n\}$. An *affine invariant* is an invariant $f : \mathbb{Z}[\mathcal{V}_{stb}]^n \rightarrow \mathbb{Z}[\mathcal{V}_{stb}]$ representable by a polynomial in $\mathbb{Z}[\mathcal{V}_{stb}][x_1, \dots, x_n]$ of the shape

$$f = a_0 + \sum_{1 \leq j \leq n} a_j x_j \quad \text{with } a_j \in \mathbb{Z}[\mathcal{V}_{stb}] \text{ for all } 0 \leq j \leq n$$

Example 4.2.7. Continuing on Example 4.2.4, an affine invariant is a polynomial $f \in \mathbb{Z}[x][k, l]$ of the shape $f = a_0 + a_1 k + a_2 l$ with $a_0, a_1, a_2 \in \mathbb{Z}[x]$, not all equal to 0, such that $f(\vec{e}_i) = 0$ for all $i \in \mathbb{N}$. Restricting to the first three divergence vectors $\vec{e}_0, \vec{e}_1, \vec{e}_2$, yields a system of 3 equations in 3 unknowns

$$\begin{aligned} f(\vec{e}_0) &= f(1, x) &= a_0 + a_1 \cdot 1 + a_2 \cdot x &= 0 \\ f(\vec{e}_1) &= f(2, x - 1) &= a_0 + a_1 \cdot 2 + a_2 \cdot (x - 1) &= 0 \\ f(\vec{e}_2) &= f(3, x - 2) &= a_0 + a_1 \cdot 3 + a_2 \cdot (x - 2) &= 0 \end{aligned}$$

This system of equations is represented equivalently as a matrix equation

$$\underbrace{\begin{pmatrix} 1 & 1 & x \\ 1 & 2 & x - 1 \\ 1 & 3 & x - 2 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}}_{\vec{a}} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

That is, every affine invariant yields a non-zero vector $\vec{a}$ such that $\mathbf{A}\vec{a} = \vec{0}$.

Definition 4.2.8 (Affine invariant matrix). Let $(\rho_i)_{i \in \mathbb{N}}$ be a simple divergence pattern in n parameters with divergence vectors $\vec{e}_0, \dots, \vec{e}_n$. We define the *affine invariant matrix* $\mathbf{A}$ of size $(n + 1) \times (n + 1)$ with entries in $\mathbb{Z}[\mathcal{V}_{stb}]$ as follows

$$\mathbf{A} = \begin{pmatrix} 1 & \vec{e}_0 \\ 1 & \vec{e}_1 \\ \vdots & \vdots \\ 1 & \vec{e}_n \end{pmatrix} = \begin{pmatrix} 1 & e_{1,0} & e_{2,0} & \cdots & e_{n,0} \\ 1 & e_{1,1} & e_{2,1} & \cdots & e_{n,1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & e_{1,n} & e_{2,n} & \cdots & e_{n,n} \end{pmatrix}$$

In Example 4.2.7, we observed that every affine invariant yields a vector $\vec{a}$ such that $\mathbf{A}\vec{a} = \vec{0}$. In algebraic terminology, the set of all such vectors is called the *kernel* of $\mathbf{A}$,

denoted by $\ker(\mathbf{A})$. In the special case $\mathcal{V}_{stb} = \emptyset$ this can be computed with standard Gaussian elimination from linear algebra. In Section 4.5 we will explain that:

- ▷ In the general case of $\mathcal{V}_{stb} \neq \emptyset$, like in Example 4.2.7, more advanced computational methods are required to compute $\ker(\mathbf{A})$ using Gröbner basis computations. This can be done with the Buchberger algorithm [20], which can be seen as a special instance of the Knuth-Bendix completion algorithm, instantiated to term rewriting systems on polynomial rings [13].
- ▷ $\ker(\mathbf{M})$ is finitely generated for any arbitrary $m \times n$ matrix $\mathbf{M}$ with entries from a Noetherian ring (such as $\mathbb{Z}[\mathcal{V}_{stb}]$). In particular, $\ker(\mathbf{A})$ is finitely generated. This also tells us that the matrix invariant method extends to arbitrary polynomial templates, because even if we include non-affine monomials, we can always express the invariant condition as a matrix equation $\mathbf{M}\vec{c} = \vec{0}$, as explained in Section 4.4.
- ▷ Except in the special case that $\mathcal{V}_{stb} = \emptyset$, the specific choice of constructing a square matrix is merely a heuristic.

Example 4.2.9. Continuing on Example 4.2.7, we compute that $\ker(\mathbf{A})$ is generated by a single vector $(a_0, a_1, a_2) = (x + 1, -1, -1)$. This yields an affine invariant $f = (x + 1) + (-1) \cdot k + (-1) \cdot l = x + 1 - k - l$. By definition of an invariant, this means that $f(\vec{e}_i) = 0$ for all $i \in \mathbb{N}$, hence by equating $f = 0$ we obtain the corresponding relation between the divergence variables: $x + 1 - k - l = 0$.

We note that it may not be true that every element in $\ker(\mathbf{A}) \setminus \{\vec{0}\}$ is an affine invariant. The reason is that, when constructing $\mathbf{A}$, we restrict the invariant condition $f(\vec{e}_i) = 0$ to finitely many indices, namely $0 \leq i \leq n$. Consequently, $\ker(\mathbf{A})$ may contain solutions that satisfy the finite system but fail to hold for some $i > n$.

Still, for isolated candidates $a_0 + \sum_{1 \leq j \leq n} a_j x_j = 0$ we may be able to verify whether it is preserved under application of χ , by checking the validity of

$$a_0 + \sum_{1 \leq j \leq n} a_j x_j = 0 \rightarrow a_0 + \sum_{1 \leq j \leq n} a_j \chi(x_j) = 0$$

As long as we stay with affine invariants, this can be automated with SMT solving. For example, in Example 4.2.9 we check that the candidate $x + 1 - k - l = 0$ is preserved under χ by checking that the following implication is a valid formula

$$x + 1 - k - l = 0 \rightarrow x + 1 - (k + 1) - (l - 1) = 0$$

However, this strategy will not be sufficient when invariants are mutually dependent, i.e. when the preservation of an invariant under χ can only be concluded in the presence of

other invariants. That being said, in all examples we present in this thesis, we have not encountered a case where $\ker(\mathbf{A})$ contains non-invariants.

Heuristic 4.2.10 (Lemma generation). Let $(\rho_i)_{i \in \mathbb{N}}$ be a simple divergence pattern with divergence shape $\ell \approx r$, divergence variables $\mathcal{V}_{div} = \{x_1, \dots, x_n\}$ and affine invariant matrix $\mathbf{A}$ such that $\ker(\mathbf{A})$ is finitely generated by $\mathcal{G} \subseteq \mathbb{Z}[\mathcal{V}_{stb}]^n$. Then we propose the following lemma candidate

$$\ell \approx r \quad \left[\bigwedge_{\vec{a} \in \mathcal{G}} \left(a_0 + \sum_{j=1}^n a_j x_j = 0 \right) \right]$$

Example 4.2.11. In Example 4.2.2 we found a divergence shape $u \ x \ k \approx d \ x \ l$ for which, in Example 4.2.9, we computed an affine invariant matrix $\mathbf{A}$ whose kernel is generated by a single vector $(x + 1, -1, -1)$. Then Heuristic 4.2.10, proposes the lemma candidate $u \ x \ k \approx d \ x \ l \ [x + 1 - k - l = 0]$. This is a valid lemma, allowing us to complete the RI proof of $up \ x \approx down \ x$.

4.2.4 Computing initial vector and divergence substitution

Thus, for a simple divergence pattern $(\rho_i)_{i \in \mathbb{N}}$, we can compute the affine invariant matrix $\mathbf{A}$ as soon as we know the initial divergence vector $\vec{e}_0$ and divergence substitution χ . In practice, however, it may not be entirely straightforward to identify $\vec{e}_0$, because ρ_0 is not necessarily the first induction rule that fits the divergence shape.

Example 4.2.12. Consider the following variant of Example 4.1.2

$up \ x \rightarrow u \ x \ 1$ $u \ x \ k \rightarrow u \ x \ (k + 2) \quad [k \leq x]$ $u \ x \ k \rightarrow x \quad [k > x]$	$down \ x \rightarrow d \ x \ x$ $d \ x \ l \rightarrow d \ x \ (l - 1) \quad [l > 0]$ $d \ x \ l \rightarrow x \quad [l \leq 0]$
---	---

The proof of $up \ x \approx down \ x$ yields a divergence pattern with divergence shape $u \ x \ k \approx d \ x \ l$, divergence variables $\mathcal{V}_{div} = \{k, l\}$ and stable variables $\mathcal{V}_{stb} = \{x\}$.

$$\begin{array}{lll}
 u \ x \ k_{-1} \rightarrow d \ x \ l_{-1} & [k_{-1} = 1 \wedge l_{-1} = x] \\
 \rho_0 & u \ x \ k_0 \rightarrow d \ x \ l_0 & [k_0 = 3 \wedge l_0 = x - 1 \wedge x \geq 1] \\
 \rho_1 & u \ x \ k_1 \rightarrow d \ x \ l_1 & [k_1 = 5 \wedge l_1 = x - 3 \wedge x \geq 3] \\
 \rho_2 & u \ x \ k_2 \rightarrow d \ x \ l_2 & [k_2 = 7 \wedge l_2 = x - 5 \wedge x \geq 5] \\
 & \vdots &
 \end{array}$$

Even though the first induction rule $u \ x \ k_{-1} \rightarrow d \ x \ l_{-1} \ [k_{-1} = 1 \wedge l_{-1} = x]$ fits the divergence shape, it does not belong to the divergence pattern.

To see why this is the case, we note that each above induction rule corresponds to an (Expansion) cycle, each with a fixed procedure: for all $i \geq -1$ we have

- (1). The induction rule $u \ x \ k_i \rightarrow d \ x \ l_i \ [\varphi_i]$ was created by (Expansion) on the left side of the equation $(\mathbf{E}_i) \ u \ x \ k_i \approx d \ x \ l_i \ [\varphi_i]$.
- (2). This yields a set of equations $\mathcal{E}_i = \text{Expd}((\mathbf{E}_i), \epsilon)$, based on all possible ways to reduce $u \ x \ k_i$ with a rule in $\mathcal{R}$ (as explained in Section 3.2.1). However, there is precisely one equation in $\mathcal{E}_i$ that initiates the next (Expansion) cycle, which is

$$(\mathbf{E}_i^{\text{div}}) \quad u \ x \ (k_i + 2) \approx d \ x \ l_i \ [k_i \leq x \wedge \varphi_i]$$

and which corresponds to the *divergence rule* $\rho_{\text{div}} = u \ x \ k \rightarrow u \ x \ (k + 2) \ [k \leq x]$.

- (3). After this, we further reduce $(\mathbf{E}_i^{\text{div}})$ by applying one or more (Simplify) steps on its right-hand side, using $d \ x \ l \rightarrow d \ x \ (l - 1) \ [l > 0]$, to obtain $(\mathbf{E}_{i+1})$.

Although the deduction steps required in (1) and (2) are the same for every (Expansion) cycle, this does not apply to (3) because the number of times that we can apply $d \ x \ l \rightarrow d \ x \ (l - 1) \ [l > 0]$ may vary. This is precisely the reason why the first induction rule does not belong to the divergence pattern. To see this, we note that

- $i = -1$: the induction rule ρ_{-1} was obtained from (Expansion) on the equation $(\mathbf{E}_{-1}) \ u \ x \ k_{-1} \approx d \ x \ l_{-1} \ [k_{-1} = 1 \wedge l_{-1} = x]$. The corresponding divergence equation in $\text{Expd}((\mathbf{E}_{-1}), \epsilon)$ is given by

$$(\mathbf{E}_{-1}^{\text{div}}) \quad u \ x \ (k_{-1} + 2) \approx d \ x \ l_{-1} \ [k_{-1} \leq x \wedge k_{-1} = 1 \wedge l_{-1} = x]$$

In step (3) we can only apply the rule $d \ x \ l \rightarrow d \ x \ (l - 1) \ [l > 0]$ once to obtain

$$(\mathbf{E}_{-1}^{\text{div}})' \quad u \ x \ (k_{-1} + 2) \approx d \ x \ (l_{-1} - 1) \ [k_{-1} \leq x \wedge k_{-1} = 1 \wedge l_{-1} = x]$$

The net effect of this (Expansion) cycle on the variables can be presented by a substitution on $\mathcal{V}_{\text{div}}$, namely $[k := k + 2, l := l - 1]$

- $i \geq 0$: one can check that step (3) involves two applications of $d \ x \ l \rightarrow d \ x \ (l - 1) \ [l > 0]$ on $(\mathbf{E}_i^{\text{div}})$. Therefore, each such (Expansion) cycle will have a net effect on $\mathcal{V}_{\text{div}}$, described by $[k := k + 2, l := l - 2]$

So $i = -1$ can be seen as a pre-processing case for entering the simple divergence, and should therefore be discarded. We identify $\chi = [k := k + 2, l := l - 2]$ and $\vec{e}_0 = (3, x - 1)$. We can then compute a correct matrix invariant $k + l - x - 2 = 0$ and a corresponding lemma $u \ x \ k \approx d \ x \ l \ [k + l - x - 2 = 0]$ by which we can complete the RI proof.

In Heuristic 4.2.13, we provide a procedure to construct candidates $\vec{e}_0$ and χ for a given sequence of induction rules. Intuitively, it should be seen as a search procedure, looking for the first i such that (3) requires the same number of (Simplify) steps in both $(\mathbf{E}_i^{\text{div}})$ and $(\mathbf{E}_{i+1}^{\text{div}})$.

Heuristic 4.2.13 (Divergence substitution & initial vector). Let $(\rho_i)_{i \in \mathbb{N}}$ be a sequence of induction rules that satisfy condition (1) and (2) of Definition 4.2.1. So $\rho_i = \ell[\vec{x} := \vec{y}_i] \rightarrow r[\vec{x} := \vec{y}_i] [\varphi_i]$ and for all $1 \leq j \leq n$ there is a term $e_{j,i} \in T(\Sigma_{\text{theory}}, \mathcal{V}_{\text{stb}})$ such that φ_i contains a conjunct $y_{j,i} = e_{j,i}$.

Furthermore, in analogy to Example 4.2.12 we assume that:

- (1). Each ρ_i comes from (Expansion) in some $p \in \text{Pos}_F(\ell[\vec{x} := \vec{y}_i])$ in the equation

$$(\mathbf{E}_i) \quad \ell[\vec{x} := \vec{y}_i] \approx r[\vec{x} := \vec{y}_i] [\varphi_i]$$

- (2). There is precisely one equation $(\mathbf{E}_i^{\text{div}})$ in $\mathcal{E}_i = \text{Expd}((\mathbf{E}_i), p)$ that initiates the next (Expansion) cycle, corresponding to what we will refer to as the *divergence rule*, using the notation ρ_{div} .
- (3). The equation $(\mathbf{E}_i^{\text{div}})$ can be reduced, using finitely many (Simplify) steps with $\mathcal{R}$. We write s_i for this number of steps. For each (Simplify) step we will keep track of which rule is used, and on which position it is applied.

We define $m = \min\{i \mid s_i = s_{i+1}\}$, if it exists. (If not, the procedure will run indefinitely) The heuristic is based on the assumption that $s_j = s_m$ for all $j > m$. (If not, the heuristic will produce an incorrect output.) Define a candidate initial vector

$$\vec{e}_0 = (e_{1,m}, \dots, e_{n,m})$$

To derive a candidate divergence substitution χ , we reproduce the deduction steps in the (Expansion) cycle of $(\mathbf{E}_m)$ on the divergence shape $\ell \approx r$. At the end of this (Expansion) cycle we can deduce a candidate χ . We will not formally define how this should be done, but use Example 4.2.14 as an illustrative example.

Example 4.2.14. To continue on Example 4.2.12: we have $m = 0$ (and $s_m = 2$), which gives us a candidate $\vec{e}_0 = (3, x - 1)$ from the equation

$$(\mathbf{E}_0) \quad u \times k_0 \approx d \times l_0 \quad [k_0 = 3 \wedge l_0 = x - 1 \wedge x \geq 1]$$

To compute a candidate χ , we note that the (Expansion) cycle of $(\mathbf{E}_m)$, which reduces $(\mathbf{E}_m)$ to $(\mathbf{E}_{m+1})$, involves the following steps:

- (i). (Expansion) on $(\mathbf{E}_0)$ to obtain $(\mathbf{E}_0^{\text{div}})$

- (ii). The execution of $s_m = 2$ (Simplify) steps with $\mathsf{d} \ x \ l \rightarrow \mathsf{d} \ x \ (l - 1) \ [l > 0]$, both on position ϵ of the right-hand side

We reproduce the same steps on the divergence shape $(\mathbf{E}) \ u \ x \ k \approx \mathsf{d} \ x \ l$:

- (i). After performing (Expansion) we obtain

$$(\mathbf{E}^{\text{div}}) \ u \ x \ (k + 2) \approx \mathsf{d} \ x \ l \quad [k \leq x]$$

- (ii). To perform the 2 rewrite steps we cannot actually use (Simplify): the constraint of $(\mathbf{E}^{\text{div}})$ does not even allow us to apply $\mathsf{d} \ x \ l \rightarrow \mathsf{d} \ x \ (l - 1) \ [l > 0]$ once. Instead, we just pretend that we are allowed to perform these (Simplify) steps. The first one yields

$$(\mathbf{E}^{\text{div}})' \ u \ x \ k' \approx \mathsf{d} \ x \ l' \quad [l' = l - 1 \wedge k' = k + 1 \wedge k \leq x]$$

The second application yields

$$(\mathbf{E}^{\text{div}})'' \ u \ x \ k' \approx \mathsf{d} \ x \ l'' \quad [l'' = l' - 1 \wedge l' = l - 1 \wedge k' = k + 1 \wedge k \leq x]$$

Finally, to obtain a candidate substitution χ , we use the constraint of $(\mathbf{E}^{\text{div}})''$ to replace k' and l'' by terms in $T(\Sigma_{\text{theory}}, \mathcal{V}_{\text{div}})$.

$$k' = k + 1 \quad \text{and} \quad l'' = l' - 1 = (l - 1) - 1 \stackrel{p}{=} l - 2$$

We define $\chi = [k := k + 1, l := l - 2]$.

4.2.5 Constrained divergence shape

We consider the equivalence $\text{sumIt } x \approx \text{sumRec } x$ in the system

$$\begin{array}{ll} \text{sumIt } x \rightarrow u \ x \ 1 \ 0 & \text{sumRec } x \rightarrow x + (\text{sumRec } (x - 1)) \quad [x > 0] \\ u \ x \ i \ a \rightarrow u \ x \ (i + 1) \ (a + i) \quad [i \leq x] & \text{sumRec } x \rightarrow 0 \quad [x \leq 0] \\ u \ x \ i \ a \rightarrow a & [i > x] \end{array}$$

This yields the following simple divergence

$$\begin{array}{ll} u \ x \ 1 \ 0 \leftarrow \text{sumRec } x & \\ \rho_0 \quad u \ x \ 2 \ 1 \rightarrow x + (u \ x_1 \ 1 \ 0) \quad [x_1 = x - 1 \wedge x \geq 1] & \\ \rho_1 \quad u \ x \ 3 \ 3 \rightarrow x + (u \ x_1 \ 2 \ 1) \quad [x_1 = x - 1 \wedge x \geq 2] & \\ \rho_2 \quad u \ x \ 4 \ 6 \rightarrow x + (u \ x_1 \ 3 \ 3) \quad [x_1 = x - 1 \wedge x \geq 3] & \end{array}$$

with divergence shape $u \ x \ i \ a \approx x + (u \ x_1 \ j \ b)$. We have $\mathcal{V}_{\text{div}} = \{i, a, j, b\}$, $\mathcal{V}_{\text{stb}} = \{x, x_1\}$, $\vec{e}_0 = (2, 1, 1, 0)$ and $\chi = [i := i + 1, a := a + i, j := j + 1, b := b + j]$.

Note that the divergence pattern contains an invariant $x_1 = x - 1$. However, this invariant does not contain any divergence variable and therefore will not be included by Heuristic 4.2.10. To fix this issue, we will replace the divergence shape $\ell \approx r$ by a constrained divergence shape $\ell \approx r \ [\varphi]$, where φ is used to store invariants that only contain stable variables. The detection of such invariants only involves syntactical checks to identify conjuncts $\psi \in T(\Sigma_{\text{theory}}, \mathcal{V}_{\text{stb}})$ of type bool that occur in every φ_i .

Definition 4.2.15 (Constrained divergence shape). Let $(\ell_i \rightarrow r_i [\varphi_i])_{i \in \mathbb{N}}$ be a simple divergence pattern.

$$C = \{\psi \in T(\Sigma_{theory}, \mathcal{V}_{stb}) \mid \psi :: \text{bool} \text{ and } \varphi_i \text{ contains a conjunct } \psi \text{ for all } i \in \mathbb{N}\}$$

We define the *constrained divergence shape* as the constrained equation

$$\ell \approx r \quad \left[\bigwedge_{\psi \in C} \psi \right]$$

Heuristic 4.2.16. Let $(\rho_i)_{i \in \mathbb{N}}$ be a simple divergence pattern with constrained divergence shape $\ell \approx r [\varphi]$, divergence variables $\mathcal{V}_{div} = \{x_1, \dots, x_n\}$ and affine invariant matrix $\mathbf{A}$ such that $\ker(\mathbf{A})$ is finitely generated by $\mathcal{G} \subseteq \mathbb{Z}[\mathcal{V}_{stb}]^n$. Then we propose the following lemma candidate

$$\ell \approx r \quad \left[\varphi \wedge \bigwedge_{\vec{a} \in \mathcal{G}} \left(a_0 + \sum_{j=1}^n a_j x_j = 0 \right) \right]$$

With Heuristic 4.2.16, we generate the candidate

$$u \ x \ i \ a \approx x + (u \ x_1 \ j \ b) \quad [x_1 = x - 1 \wedge i = j + 1 \wedge a + 1 = b + i]$$

Unfortunately, this is not a lemma: we need an additional inequality invariant $x \geq j$.

4.3 Inequality invariants

Although matrix invariants are effective for discovering equality invariants, they are often insufficient for generating lemmas. A key limitation is that they do not cover inequality invariants. For example, we saw that Heuristic 4.2.16 alone is insufficient to prove $\text{sumlt } x \approx \text{sumRec } x$, because the proof requires an additional invariant $x \geq j$. On the other hand, the InGen method *is* capable of generating such invariants.

Here, we aim to combine the best of both worlds, using matrix invariants to generate equalities and an InGen-like strategy to generate inequalities. Rather than incorporating InGen as a whole, we analyze how it actually generates inequalities and isolate the underlying mechanism, which we can then formulate as a self-contained inequality generation method for simple divergence patterns.

This section is structured as follows:

- In Section 4.3.1 we define the *divergence constraint* φ_{div} of a divergence shape, using the divergence rule ρ_{div} .
- In Section 4.3.2 we show how to use φ_{div} to obtain InGen-like inequalities.
- In Section 4.3.3 we explain how to use an InGen-like inequality to derive additional inequality invariants. In practice, these *derived invariants* may describe the necessary remaining inequalities for obtaining a lemma.

4.3.1 Divergence constraint

To understand how InGen generates inequalities we should first take a closer look at the specific role that inequalities play in a simple divergence pattern, which we already partially discussed in Section 4.2.4:

The divergence pattern is typically generated from a single rewrite rule in $\mathcal{R}$, which we defined as the divergence rule ρ_{div} . Inequalities in the divergence pattern are imposed during the (Expansion) steps to enforce the constraint of ρ_{div} . For instance, consider the first two divergence rules in the divergence pattern of $\text{sumIt } x \approx \text{sumRec } x$

$$\begin{aligned}\rho_0 & \quad u \ x \ 2 \ 1 \rightarrow x + (u \ x_1 \ 1 \ 0) \quad [x_1 = x - 1 \wedge x \geq 1] \\ \rho_1 & \quad u \ x \ 3 \ 3 \rightarrow x + (u \ x_1 \ 2 \ 1) \quad [x_1 = x - 1 \wedge x \geq 2]\end{aligned}$$

We see that the inequality $x \geq 2$ in the constraint of ρ_1 was imposed by (Expansion) on

$$(\mathbf{E}_0) \quad u \ x \ 2 \ 1 \approx x + (u \ x_1 \ 1 \ 0) \quad [x_1 = x - 1 \wedge x \geq 1]$$

to enforce the constraint of $\rho_{\text{div}} = u \ x \ i \ a \rightarrow u \ x \ (i+1) \ (a+i) \ [i \leq x]$. We say that $x \geq 2$ is the *divergence constraint* for $(\mathbf{E}_0)$. More generally, we say that $i \leq x$ is the divergence constraint for the divergence shape $u \ x \ i \ a \approx x + (u \ x_1 \ j \ b)$.

Definition 4.3.1 (Divergence constraint). Let $(\ell_i \rightarrow r_i \ [\varphi_i])_{i \in \mathbb{N}}$ be a simple divergence pattern with divergence shape $\ell \approx r$. Assume $\ell = \ell[f \ l_1 \cdots l_m]_p$ for some position p and corresponding divergence rule $f \ s_1 \cdots s_k \rightarrow t \ [\psi] \in \mathcal{R}$. We define the *divergence constraint* φ_{div} as the constraint $\psi\delta$, where $\delta = \text{mgu}(f \ s_1 \cdots s_k, f \ l_1 \cdots l_k)$.

Example 4.3.2. We consider the divergence pattern for $\text{sumIt } x \approx \text{sumRec } x$ with divergence shape $u \ x \ i \ a \approx x + (u \ x_1 \ j \ b)$ and divergence rule $u \ x \ i \ a \rightarrow u \ x \ (i+1) \ (a+i) \ [i \leq x]$ corresponding to position $p = \epsilon$ in $\ell = u \ x \ i \ a$. In particular, δ is the identity substitution and therefore $\varphi_{\text{div}} = (i \leq x)$.

4.3.2 InGen-like inequalities

Since InGen only generates inequalities via (Expansion) steps, we observe that these are necessarily instances of the divergence constraint.

Example 4.3.3. Let us apply InGen to $\text{sumIt } x \approx \text{sumRec } x$. First, we create the equivalent system where the initializations $i_{\text{I}} = 1$ and $a_{\text{I}} = 0$ are moved into the constraint:

$$\begin{aligned}\text{sumIt } x & \rightarrow u \ x \ i_{\text{I}} \ a_{\text{I}} & [i_{\text{I}} = 1 \wedge a_{\text{I}} = 0] \\ u \ x \ i \ a & \rightarrow u \ x \ (i+1) \ (a+i) & [i \leq x] \quad \text{sumRec } x \rightarrow x + (\text{sumRec } (x-1)) \quad [x > 0] \\ u \ x \ i \ a & \rightarrow a & [i > x] \quad \text{sumRec } x \rightarrow 0 \quad [x \leq 0]\end{aligned}$$

This yields the following simple divergence, where we abbreviate $\psi_I = (i_I = 1 \wedge a_I = 0)$

$$\begin{aligned} & \text{u } x \ i_I \ a_I \leftarrow \text{sumRec } x \quad [\psi_I] \\ \rho_0 \quad & \text{u } x \ i_0 \ a_0 \rightarrow x + (\text{u } x_1 \ i_I \ a_I) \quad [x \geq i_I \wedge (i_0 = i_I + 1 \wedge a_0 = a_I + i_I \wedge x_1 = x - 1) \wedge \psi_I] \\ \rho_1 \quad & \text{u } x \ i_1 \ a_1 \rightarrow x + (\text{u } x_1 \ i_0 \ a_0) \quad [x \geq i_0 \wedge \dots] \\ \rho_2 \quad & \text{u } x \ i_2 \ a_2 \rightarrow x + (\text{u } x_1 \ i_1 \ a_1) \quad [x \geq i_1 \wedge \dots] \end{aligned}$$

We note that every inequality $x \geq i_n$ is an instance of the divergence constraint $\varphi_{\text{div}} = (i \leq x)$, but in a delayed manner: each ρ_n only contains all $\varphi_{\text{div}}[i := i_k]$ for all $k < n$. This makes sense, because if it also contained $\varphi_{\text{div}}[i := i_n]$ then we could apply the divergence rule $\rho_{\text{div}} = \text{u } x \ i \ a \rightarrow \text{u } x \ (i + 1) \ (a + i) \ [i \leq x]$ with (Simplify), hence there was no need to perform an (Expansion).

InGen generates a lemma candidate by removing all initializations, which is usually performed at the end of an (Expansion) cycle. Therefore, after removing ψ_I from the constraint, we can think of each ρ_n as an oriented lemma candidate. For example, at the moment just before the ρ_0 -(Expansion) we have the equation

$$(E_0) \quad \text{u } x \ i_0 \ a_0 \approx x + (\text{u } x_1 \ i_I \ a_I) \quad [x \geq i_I \wedge (i_0 = i_I + 1 \wedge a_0 = a_I + i_I \wedge x_1 = x - 1) \wedge \psi_I]$$

By removing the initializations ψ_I we obtain a candidate

$$(C) \quad \text{u } x \ i_0 \ a_0 \approx x + (\text{u } x_1 \ i_I \ a_I) \quad [x \geq i_I \wedge (i_0 = i_I + 1 \wedge a_0 = a_I + i_I \wedge x_1 = x - 1)]$$

For HORI this is not actually a lemma because the corresponding induction rule is not terminating (we will see in Chapter 5 that for Bounded RI it *does* suffice as a lemma). In Section 4.3.3 we will explain how to transform this into a lemma with a terminating induction rule, but first we focus on how to reproduce this inequality $x \geq i_I$.

InGen-like inequalities We will show, along Example 4.3.3, that we can reproduce the inequalities generated by InGen in a systematic manner. We assume we are provided with the output from Heuristic 4.2.16. In the setting of Example 4.3.3 this yields

$$(C_0) \quad \text{u } x \ i \ a \approx x + (\text{u } x_1 \ j \ b) \quad [x_1 = x - 1 \wedge i = j + 1 \wedge a + 1 = b + i]$$

Now, we follow an InGen-like strategy, meaning that we will introduce $\varphi_{\text{div}} = (i \leq x)$ in a delayed manner. This can be achieved in two ways: (1) let the divergence shape be one step ahead of φ_{div} , or (2) let φ_{div} be one step behind on the divergence shape. Both approaches rely on the manipulation of terms using the divergence substitution

$$\chi = [i := i + 1, a := a + i, j := j + 1, b := b + j]$$

- (1). To make sure that the divergence is one step ahead of φ_{div} , we apply χ to the divergence shape in (C_0) . This yields

$$(C_0)\chi \quad \text{u } x(i+1)(a+i) \approx x + (\text{u } x_1(j+1)(b+j)) \quad [x_1 = x-1 \wedge i = j+1 \wedge a+1 = b+i]$$

We can now add φ_{div} to the constraint (and also move the non-variable theory terms to the constraint) to obtain a candidate

$$(C_1) \quad \text{u } x \ i' \ a' \approx x + (\text{u } x_1 \ j' \ b') \\ [i \leq x \wedge i' = i+1 \wedge a' = a+i \wedge j' = j+1 \wedge b' = b+j \wedge \\ x_1 = x-1 \wedge i = j+1 \wedge a+1 = b+i]$$

- (2). To make sure that φ_{div} is one step behind on the divergence shape, we look for a term ψ such that $\psi\chi = \varphi_{\text{div}}$. In case χ defines an injective function $\chi : \mathbb{Z}^n \rightarrow \mathbb{Z}^n$ we can compute $\psi = \varphi_{\text{div}}\chi^{-1}$, with χ^{-1} the inverse of χ . In this example, $\chi : \mathbb{Z}^4 \rightarrow \mathbb{Z}^4$ defined by $\chi(i, a, j, b) = (i+1, a+i, j+1, b+j)$ is injective, and we compute

$$\chi^{-1} = [i := i-1, a := a-(i-1), j := j-1, b := b-(j-1)]$$

and $\psi = \underbrace{(i \leq x)}_{\varphi_{\text{div}}} \chi^{-1} = (i-1 \leq x)$. We add ψ to (C_0) to obtain a candidate

$$(C_2) \quad \text{u } x \ i \ a \approx x + (\text{u } x_1 \ j \ b) \quad [i-1 \leq x \wedge x_1 = x-1 \wedge i = j+1 \wedge a+1 = b+i]$$

Note: we do not always need the full χ to be injective. It may be sufficient that there exists some $\text{Var}(\varphi_{\text{div}}) \cap \mathcal{V}_{\text{div}} \subseteq V \subseteq \mathcal{V}_{\text{div}}$ with $\text{Var}(\chi(x)) \subseteq V$ for all $x \in V$ and such that $\chi|_V : \mathbb{Z}^{|V|} \rightarrow \mathbb{Z}^{|V|}$ (i.e. restricting the domain of χ to V) is injective. For example, let $\varphi_{\text{div}} = (x \geq a)$ and consider the non-injective substitution $\chi = [i := i+1, a := a+i, j := j^2+1, b := b+j]$. Note that the variables j and b are irrelevant for computing the pre-image of φ_{div} . We choose $V = \mathcal{V}_{\text{div}} \setminus \{j, b\} = \{i, a\}$. Then $\text{Var}(\chi(x)) \subseteq V$ for all $x \in V$, therefore $\chi|_V$ is well-defined. Moreover, $\chi|_V = [i := i+1, a := a+i]$ is injective with inverse $\chi|_V^{-1} = [i := i-1, a := a-(i-1)]$. Hence $\varphi_{\text{div}}\chi|_V^{-1} = (x \geq a-i+1)$.

Although (C_1) and (C_2) are equivalent, (C_1) has a more elaborate presentation. In general, strategy (1) will produce more elaborate lemmas than (2). On the other hand, strategy (1) is always applicable while (2) is only applicable when χ is (to some extent) injective. In practice, we therefore always start with (2) and only use (1) when (2) fails.

4.3.3 Derived invariants

Even when combining Heuristic 4.2.16 with divergence invariants, it may happen that this is not enough for obtaining a lemma. At this stage we have accumulated sufficiently many invariants such that their combination is very suitable for deriving additional invariants. With these derived invariants we can transform (C₂) from Section 4.3.2 into a lemma with a terminating rewrite rule (as required by HORI). For illustrative purposes, however, we will first consider another example:

Example 4.3.4. Consider $\text{sumRec } x \approx \text{sumlt2 } x$ where $\mathcal{R}$ is given by

$$\begin{array}{ll} \text{sumlt2 } x \rightarrow \text{d } x \ 0 & \\ \text{d } x \ a \rightarrow \text{d } (x - 1) \ (a + x) & [x > 0] \\ \text{d } x \ a \rightarrow a & [x \leq 0] \end{array} \quad \begin{array}{ll} \text{sumRec } x \rightarrow x + (\text{sumRec } (x - 1)) & [x > 0] \\ \text{sumRec } x \rightarrow 0 & [x \leq 0] \end{array}$$

This yields the following divergence

$$\begin{array}{ll} \text{sumRec } x \rightarrow \text{d } x \ 0 & \\ \text{d } y_0 \ a_0 \rightarrow x + (\text{d } y_0 \ b_0) & [x > 0 \wedge y_0 = x - 1 \wedge a_0 = x \quad \wedge b_0 = 0] \\ \text{d } y_1 \ a_1 \rightarrow x + (\text{d } y_1 \ b_1) & [x > 1 \wedge y_1 = x - 2 \wedge a_1 = 2x - 1 \quad \wedge b_1 = x - 1] \\ \text{d } y_2 \ a_2 \rightarrow x + (\text{d } y_2 \ b_2) & [x > 2 \wedge y_2 = x - 3 \wedge a_2 = 3x - 3 \quad \wedge b_2 = 2x - 3] \end{array}$$

with divergence shape $\text{d } y \ a \approx x + (\text{d } y \ b)$. We have $\mathcal{V}_{\text{div}} = \{y, a, b\}$, $\mathcal{V}_{\text{stb}} = \{x\}$ and $\chi = [y := y - 1, a := a + y, b := b + y]$. With Heuristic 4.2.10 we generate

$$\text{d } y \ a \approx x + (\text{d } y \ b) \quad [a = x + b]$$

which in HORI is not a lemma because both $\text{d } y \ a \rightarrow x + (\text{d } y \ b) \ [a = x + b]$ and $x + (\text{d } y \ b) \rightarrow \text{d } y \ a \ [a = x + b]$ are not terminating. We have $\chi^{-1} = [y := y + 1, a := a - (y + 1), b := b - (y + 1)]$ and $\varphi_{\text{div}} = (y > 0)$. We compute an InGen-like inequality invariant $\psi = \varphi_{\text{div}} \chi^{-1} = (y + 1 > 0)$, but this is still not sufficient to guarantee termination.

We demonstrate how to combine ψ with $\vec{e}_0$ and χ to compute additional invariants to obtain a terminating induction rule:

- (i). *Initialization invariant:* Combining ψ with $\vec{e}_0 = (y_0, a_0, b_0) = (x - 1, x, 0)$, we obtain $(y + 1 > 0)[y := x - 1] = (x - 1) + 1 > 0$, or equivalently $x > 0$. To verify that $x > 0$ is preserved along the whole divergence pattern, we should check validity of the implication $\psi \chi \rightarrow \psi$, i.e. $y > 0 \rightarrow y + 1 > 0$, which is indeed valid.
- (ii). *Monotonicity invariants:* $\psi = (y + 1 > 0)$ is equivalent to $y \geq 0$. Therefore, from $\chi(b) = b + y$ it follows that $\chi(b) \geq b$ for all b . So b is monotonically increasing and thus bounded below by $b_0 = 0$. This yields $b \geq 0$.

Including $x > 0$, $b \geq 0$ and ψ we obtain a lemma with a terminating induction rule:

$$\mathsf{d} \ y \ a \approx x + (\mathsf{d} \ y \ b) \quad [x > 0 \wedge b \geq 0 \wedge y + 1 > 0 \wedge a = x + b]$$

The above approach is generalized in Heuristic 4.3.5 where, for the verification of the required implications, we additionally exploit the available information from φ (which equals the conjunction of all matrix invariants). We note that the heuristic is only applicable in settings where strategy (2) from Section 4.3.2 can be applied.

Heuristic 4.3.5 (Derived invariants). Let $(\ell_i \rightarrow r_i [\varphi_i])_{i \in \mathbb{N}}$ be a simple divergence pattern with $\mathcal{V}_{div} = \{x_1, \dots, x_n\}$, $\mathcal{D}iv = \{\vec{e}_i \mid i \in \mathbb{N}\}$, and lemma candidate $\ell \approx r [\varphi]$ obtained from Heuristic 4.2.16. Assume we have divergence constraint φ_{div} and we computed an InGen-like inequality $\psi = \varphi_{div}\chi^{-1}$. We define two types of derived invariants:

- (i). *Initialization invariant*: If $\text{Var}(\psi[\vec{x} := \vec{e}_0]) \neq \emptyset$ and the implication $\psi\chi \wedge \varphi \rightarrow \psi$ is valid then we introduce the initialization invariant $\psi[\vec{x} := \vec{e}_0]$.
- (ii). *Monotonicity invariants*: We define a set of monotonicity invariants

$$\bigcup_{j=1}^n \{x_j \geq e_{j,0} \mid \varphi \wedge \psi \rightarrow \chi(x_j) \geq x_j\} \cup \bigcup_{j=1}^n \{x_j \leq e_{j,0} \mid \varphi \wedge \psi \rightarrow \chi(x_j) \leq x_j\}$$

Note: this yields at most n monotonicity invariants, as by Definition 4.2.1 we have $\chi(x_j) \neq x_j$ for all $1 \leq j \leq n$.

With this heuristic, we indeed compute the invariants from Example 4.3.4. Now, reconsider **(C₂)** from Section 4.3.2.

Example 4.3.6. In Section 4.3.2 we considered the equivalence $\text{sumIt } x \approx \text{sumRec } x$ in

$$\begin{array}{ll} \text{sumIt } x \rightarrow \mathsf{u} \ x \ 1 \ 0 & \text{sumRec } x \rightarrow x + (\text{sumRec } (x - 1)) \quad [x > 0] \\ \mathsf{u} \ x \ i \ a \rightarrow \mathsf{u} \ x \ (i + 1) \ (a + i) \quad [i \leq x] & \text{sumRec } x \rightarrow 0 \quad [x \leq 0] \\ \mathsf{u} \ x \ i \ a \rightarrow a & [i > x] \end{array}$$

and computed a lemma candidate

$$\mathbf{(C_2)} \quad \mathsf{u} \ x \ i \ a \approx x + (\mathsf{u} \ x_1 \ j \ b) \quad [i - 1 \leq x \wedge x_1 = x - 1 \wedge i = j + 1 \wedge a + 1 = b + i]$$

via the InGen-like inequality $\psi = \varphi_{div}\chi^{-1} = (i - 1 \leq x)$. We explained that **(C₂)** is not a lemma in HORI, because the corresponding induction rule is not terminating. We recall that $\chi = [i := i + 1, a := a + i, j := j + 1, b := b + j]$ and $\vec{e}_0 = (i_0, a_0, j_0, b_0) = (2, 1, 1, 0)$, so we compute an initialization invariant $(i - 1 \leq x)[i := 2] = 2 - 1 \leq x$, or equivalently $x \geq 1$. We check validity of $\underbrace{i \leq x}_{\psi\chi} \rightarrow \underbrace{i - 1 \leq x}_{\psi}$, so $x \geq 1$ is preserved along the divergence pattern. This is already sufficient to obtain a lemma with a terminating induction rule:

$$\mathbf{(C_2)'} \quad \mathsf{u} \ x \ i \ a \approx x + (\mathsf{u} \ x_1 \ j \ b) \quad [x \geq 1 \wedge i - 1 \leq x \wedge x_1 = x - 1 \wedge i = j + 1 \wedge a + 1 = b + i]$$

4.4 Non-affine matrix Invariants

After studying inequality invariants we will now finish the discussion on matrix invariants. In Section 4.2 we explained that any non-zero polynomial $f \in \mathbb{Z}[\mathcal{V}_{stb}][\mathcal{V}_{div}]$ that satisfies $f(\vec{e}_i) = 0$ for all i can serve as an invariant, but until now we restricted to $\mathcal{V}_{div}$ -affine templates. In practice this is insufficient, as we easily encounter situations where affine templates do not capture all required invariants. We will now consider such an example.

Example 4.4.1. Consider the following LCSTRSs

factlter $x \rightarrow u \ x \ 1 \ 1$

$$\begin{array}{ll} u \ x \ i \ a \rightarrow u \ x \ (i + 1) \ (i * a) & [i \leq x] \quad \text{factRec } x \rightarrow x * (\text{factRec } (x - 1)) \quad [x > 1] \\ u \ x \ i \ a \rightarrow a & [i > x] \quad \text{factRec } x \rightarrow 1 \quad [x \leq 1] \end{array}$$

The proof of $\text{factlter } x \approx \text{factRec}$ shows the following divergence

$$\begin{array}{l} u \ x \ 1 \ 1 \leftarrow \text{factRec } x \\ u \ x \ 3 \ 2 \rightarrow x * (u \ x_1 \ 2 \ 1) \quad [2 \leq x \wedge x_1 = x - 1] \\ u \ x \ 4 \ 6 \rightarrow x * (u \ x_1 \ 3 \ 2) \quad [3 \leq x \wedge x_1 = x - 1] \\ u \ x \ 5 \ 24 \rightarrow x * (u \ x_1 \ 4 \ 6) \quad [4 \leq x \wedge x_1 = x - 1] \end{array}$$

with constrained divergence shape $u \ x \ i \ a \approx x * (u \ x_1 \ j \ b) \ [x_1 = x - 1]$, $\mathcal{V}_{stb} = \emptyset$, $\mathcal{V}_{div} = \{i, a, j, b\}$, $\chi = [i := i + 1, a := i * a, j := j + 1, b := j * b]$ and $\vec{e}_0 = (i_0, a_0, j_0, b_0) = (3, 2, 2, 1)$. If we apply Heuristic 4.2.16 then we find the following candidate

$$u \ x \ i \ a \approx x * (u \ x_1 \ j \ b) \quad [x_1 = x - 1 \wedge i = j + 1]$$

which is not a lemma: the crucial non-affine invariant $a = j * b$ is missing. (Also, we need to include the InGen-like invariant $x \geq j$. To satisfy the termination requirement for (Expansion) in HORI, we additionally have to include the derived invariant $j \geq 2$.)

Non-affine matrix invariants The matrix invariant method applies to any polynomial template. To illustrate this, we note that in Example 4.4.1 the invariant $a = j * b$ can be generated by the inclusion of the degree 2 monomial $j * b$ in the affine template (the monomial a is already included in the affine template). So we take

$$f = c_0 + c_i i + c_a a + c_j j + c_b b + c_{(j,b)} j b \text{ with } c_0, c_i, c_a, c_j, c_b, c_{(j,b)} \in \mathbb{Z} \text{ (we have } \mathcal{V}_{stb} = \emptyset)$$

The template has 6 unknown coefficients, so we compute $f(\vec{e}_i) = 0$ for all $0 \leq i \leq 5$. This yields the matrix equation

$$\underbrace{\begin{pmatrix} 1 & 3 & 2 & 2 & 1 & 2 \cdot 1 \\ 1 & 4 & 6 & 3 & 2 & 3 \cdot 2 \\ 1 & 5 & 24 & 4 & 6 & 4 \cdot 6 \\ 1 & 6 & 120 & 5 & 24 & 5 \cdot 24 \\ 1 & 7 & 720 & 6 & 120 & 6 \cdot 120 \\ 1 & 8 & 5040 & 7 & 720 & 7 \cdot 720 \end{pmatrix}}_{\mathbf{D}} \underbrace{\begin{pmatrix} c_0 \\ c_i \\ c_a \\ c_j \\ c_b \\ c_{(j,b)} \end{pmatrix}}_{\vec{c}} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

So for each row in $\mathbf{D}$, the value in the last jb -column is precisely the product of the value in the j column with the value in the b column. With a computer algebra system we compute that $\ker(\mathbf{D})$ is generated by two vectors

$$(1, -1, 0, 1, 0, 0) \quad \text{and} \quad (0, 0, -1, 0, 0, 1)$$

corresponding to the invariants

$$1 - i + j = 0 \quad \text{and} \quad -a + jb = 0$$

So this yields exactly all desired invariants of degree $0 \leq d \leq 2$

Computing non-affine matrix invariants In the previous discussion, we used our knowledge of the degree 2 invariant $a = j * b$ to extend the affine template with an additional monomial jb . In general, of course, this is not known beforehand. Therefore, when computing degree 2 invariants, we will include all degree 2 monomials (in the next paragraph we discuss how to make more clever choices). Since the affine template contains all monomials of degree $0 \leq d < 2$, the corresponding template can be written as $f = h_0 + h_1 + h_2$ where h_i contains all monomials of degree i for $0 \leq i \leq 2$. That is

$$\begin{aligned} h_0 &= c_0 \\ h_1 &= c_i i + c_a a + c_j j + c_b b \\ h_2 &= c_{(i,i)} i^2 + c_{(i,a)} i a + c_{(a,a)} a^2 + c_{(i,j)} i j + c_{(a,j)} a j + c_{(j,j)} j^2 + \\ &\quad c_{(i,b)} i b + c_{(a,b)} a b + c_{(j,b)} j b + c_{(b,b)} b^2 \end{aligned}$$

The polynomial f has 15 unknown coefficients. We therefore choose to restrict the divergence criterion $f(\vec{e}_i) = 0$ for $0 \leq i \leq 14$. Again, this yields a matrix equation $\mathbf{D}\vec{c} = \vec{0}$, where $\mathbf{D}$ is a 15×15 matrix and $\vec{c} = (c_0, c_i, \dots, c_b, c_{(i,i)}, \dots, c_{(b,b)})$ is the vector of unknown coefficients. We cannot display the matrix $\mathbf{D}$ here, because it is too big

(containing huge numbers). However, with a computer algebra system we easily compute $\ker(\mathbf{D})$ and a corresponding set of 6 invariants:

$$\begin{aligned} (1). \quad & -i + j + 1 = 0 & (2). \quad & -i^2 + ij + i = 0 & (3). \quad & -ia + aj + a = 0 \\ (4). \quad & -i^2 + 2i + j^2 - 1 = 0 \\ (5). \quad & ib - a - b = 0 & (6). \quad & -a + jb = 0 \end{aligned}$$

We observe that we only need to preserve invariants **(1)** and **(6)**, namely

- **(2)** and **(3)** are easily seen to be consequences of **(1)**. Also **(4)** is a consequence of **(1)**, which becomes clear after squaring the identity $j = i - 1$.
- **(5)** can be written as $-a + (i - 1)b = 0$, which is equivalent to **(6)** by $j = i - 1$.

More efficient template strategy The previous computation suggests a more optimized strategy, in which we first compute lower degree invariants before choosing which higher-degree monomials to include. For example, if we first compute the affine invariants then we already obtain $i = j + 1$. From this, we can conclude that for computing degree 2 invariants we can choose $c_{(i,j)} = 0$ and either

- (1). delete every coefficient in which i occurs, i.e. $c_i = c_{(i,a)} = \dots = 0$

or

- (2). delete every coefficient in which j occurs

4.5 Mathematical context

In this section we aim to achieve two goals:

- (1). Providing a mathematical justification of the claims made in Section 4.2.3:
 - (i). No matter what polynomial template we assume (i.e. not necessarily affine), the resulting matrix $\mathbf{M}$ has a finitely generated kernel.
 - (ii). Except in the special case that $\mathcal{V}_{stb} = \emptyset$, the specific choice of constructing a square matrix is merely a heuristic.
 - (iii). Computation of matrix invariants is achieved via Gröbner basis computations, relying on the Buchberger algorithm [20].
- (2). Comment on how an alternative popular invariant generation method can be applied in our setting. This alternative method is based on algebraic geometry and has the benefit that we do not need to fix a polynomial template beforehand, thereby eliminating the problem of choosing which monomials to include.

Remark. The material presented in this section requires mathematical prerequisites beyond what is typically assumed in term rewriting and formal methods. While this section provides the underlying mathematical justification for the matrix invariant method, it is not required for understanding its practical application in program equivalence, already discussed in previous sections. Therefore, readers interested only in the practical application of the method should feel free to skip this section.

Claim (i): finitely generated kernels

This claim follows from the following more general statement:

Theorem 4.5.1. Let $\mathbf{M}$ be an $m \times n$ matrix with entries from a Noetherian ring R . Then $\ker(\mathbf{M})$ is finitely generated.

Proof. Any $m \times n$ matrix $\mathbf{M}$ with entries from a ring R defines an R -module homomorphism $\mathbf{M} : R^n \rightarrow R^m$, by $\vec{x} \mapsto \mathbf{M}\vec{x}$. Furthermore, $\ker(\mathbf{M})$ is a submodule of R^n [12, Chapter 2]. Now, since R is a Noetherian ring we can say more. We cite two well-known results from commutative algebra:

- (1). If R is a Noetherian ring then R^n is a Noetherian R -module, for every $n \geq 1$ [12, Proposition 6.5].
- (2). If M is a Noetherian R -module then every submodule $N \subseteq M$ is finitely generated [12, Proposition 6.2].

Combining (1) and (2) yields that any submodule of R^n is finitely generated, for all $n \geq 1$. In particular, $\ker(\mathbf{M})$ is a submodule of R^n , so finitely generated. $\square$

By Hilbert Basis Theorem [12, Theorem 7.5]: if R is Noetherian then $R[x]$ is Noetherian as well. Since $\mathbb{Z}$ is a Noetherian ring, we can repeatedly apply this theorem to obtain that $\mathbb{Z}[\mathcal{V}]$ is Noetherian for any finite set $\mathcal{V}$ of variables. In particular, $\mathbb{Z}[\mathcal{V}_{stb}]$ is Noetherian. Therefore, Theorem 4.5.1 implies that $\ker(\mathbf{M})$ is finitely generated.

Claim (ii): square matrices is just a heuristic

Computing matrix invariants requires us to fix a polynomial template with unknown coefficients assembled in a vector $\vec{c}$. Evaluating $f(\vec{e}_i) = 0$ for all $0 \leq i < t = |\vec{c}|$ yields a $t \times t$ matrix equation $\mathbf{M}$ corresponding to the equation $\mathbf{M}\vec{c} = \vec{0}$.

We now argue that the specific choice of constructing a square matrix $\mathbf{M}$, beyond the setting of $\mathcal{V}_{stb} = \emptyset$, is merely a heuristic and does not constitute a theoretical justification.

The choice is inspired by the classical linear algebra view, where matrices are interpreted as linear transformations of vector spaces. In this setting, a system in t

unknowns can involve at most t linearly independent equations in a t -dimensional vector space. This motivates using the condition $f(\vec{e}_0) = 0, \dots, f(\vec{e}_{t-1}) = 0$, at least if we would obtain t linearly independent vectors over a *field*. In the special case $\mathcal{V}_{stb} = \emptyset$, we indeed obtain entries in $\mathbb{Z}[\mathcal{V}_{stb}] = \mathbb{Z}$, which may be interpreted as elements in $\mathbb{Q}$, where this reasoning makes sense. However, in the general case of $\mathcal{V}_{stb} \neq \emptyset$ this is invalid:

As we just mentioned, $\vec{x} \mapsto \mathbf{M}\vec{x}$ defines a linear transformation of $\mathbb{Z}[\mathcal{V}_{stb}]$ -modules, where linear algebra theory does not automatically apply. In the module setting, adding additional rows can still be seen to only shrink (or leave unchanged) the kernel. This suggests further investigation, aiming to achieve stabilization within a theoretically bounded amount of steps. However, this requires a more sophisticated analysis beyond the scope of this thesis. For now, the choice for generating square matrices remains an arbitrary heuristic.

Claim (iii): matrix invariants via Gröbner basis computations

To justify this claim, it is convenient to adopt a different mathematical perspective: we will explain that kernels of matrices can be interpreted as *syzygies*, which can be computed via Gröbner basis algorithms using the Buchberger algorithm. The Buchberger algorithm can be seen as a special instance of the Knuth-Bendix completion algorithm, instantiated to term rewriting systems on polynomial rings [13].

Definition 4.5.2 (Syzygies). Let R be a ring and M be an R -module. A syzygy of $m_1, \dots, m_n \in M$ is a tuple $(r_1, \dots, r_n) \in R^n$ such that $r_1 m_1 + \dots + r_n m_n = 0$ [20]. The set of all syzygies is denoted $\text{Syz}(m_1, \dots, m_n)$.

It is easily verified that $\text{Syz}(m_1, \dots, m_n)$ is a submodule of R^n .

Lemma 4.5.3. Let $\mathbf{M}$ be an $m \times n$ matrix with entries from a ring R and let $\vec{M}_i \in R^m$ denote the i^{th} column of $\mathbf{M}$ for $1 \leq i \leq n$. Then $\ker(\mathbf{M}) = \text{Syz}(\vec{M}_1, \dots, \vec{M}_n)$.

This lemma follows directly from the definition of matrix multiplication, since $\mathbf{M}\vec{a} = \vec{0}$ is equivalently written as $a_1 \vec{M}_1 + \dots + a_n \vec{M}_n = \vec{0}$.

Computing matrix invariants via syzygies With syzygies we obtain a concrete procedure for computing matrix invariants: we fix a polynomial template with unknown coefficients assembled in a vector $\vec{c}$. Evaluating $f(\vec{e}_i) = 0$ for all $0 \leq i < t = |\vec{c}|$ yields a $t \times t$ matrix $\mathbf{M}$, corresponding to the equation $\mathbf{M}\vec{c} = \vec{0}$. By Lemma 4.5.3 we should compute $\text{Syz}(\vec{M}_1, \dots, \vec{M}_t)$, where $\vec{M}_i \in R^m$ denotes the i^{th} column of $\mathbf{M}$.

However, much of the literature for computing syzygies, e.g. [20], assumes a base ring which is a *field*, while in our setting we have a base ring $\mathbb{Z}$, which is not a field. To apply these methods in our situation, we would have to interpret integer computations not in

$\mathbb{Z}$ but in $\mathbb{Q}$. In this way, we can identify $\mathcal{D}iv \subseteq \mathbb{Q}[\mathcal{V}_{stb}]^n$ (rather than $\mathcal{D}iv \subseteq \mathbb{Z}[\mathcal{V}_{stb}]^n$), and interpret the entries from $\mathbf{M}$ as elements in $\mathbb{Q}[\mathcal{V}_{stb}]$. However, this leads us to compute $\ker_{\mathbb{Q}[\mathcal{V}_{stb}]}(\mathbf{M})$, which is in general not equal to $\ker_{\mathbb{Z}[\mathcal{V}_{stb}]}(\mathbf{M})$. We note that

- (a). This does not affect correctness: essentially because any zero-dependency over $\mathbb{Q}$ can be transformed into a zero-dependency over $\mathbb{Z}$. For example, if we have $\mathcal{V}_{div} = \{x, y\}$ and compute an invariant $\frac{3}{2}x + \frac{1}{3}y = 0$ then we can multiply-out denominators to obtain an equivalent zero-dependency over $\mathbb{Z}$, namely $9x + 2y = 0$.
- (b). This *does* affect completeness: by the strategy explained in (a) we transform a generating set of $\ker_{\mathbb{Q}[\mathcal{V}_{stb}]}(\mathbf{M})$ into a set $X \subseteq \ker_{\mathbb{Z}[\mathcal{V}_{stb}]}(\mathbf{M})$. However, we cannot generally conclude that X generates all of $\ker_{\mathbb{Z}[\mathcal{V}_{stb}]}(\mathbf{M})$.

It is worth noting that in [1, Chapter 4] it is explained that syzygy computations can be generalized to rings $R[x_1, \dots, x_m]$ with R a Noetherian ring. Since this includes $\mathbb{Z}[\mathcal{V}_{stb}]$, this would imply that we *can* directly compute $\ker_{\mathbb{Z}[\mathcal{V}_{stb}]}(\mathbf{M})$. However, a detailed investigation of the corresponding advantages and disadvantages compared to computations over $\mathbb{Q}$ lies outside the scope of this thesis.

An alternative: the algebraic geometry approach

The matrix invariant method requires us to first choose a particular polynomial template before we are able to construct a matrix $\mathbf{M}$ of which we should compute $\ker(\mathbf{M})$. Another approach, relating to algebraic geometry, allows us to directly compute a generating set representing *all* polynomial invariants. That is, there is no need to restrict to a specific template shape, or to put a bound on the polynomial degree of the computed invariants. However, we currently have no experience with this method, as its implementation requires more sophisticated knowledge about computer algebra systems for computing ideals in rings. Nevertheless, to facilitate future implementation of this alternative method, we will explain how our specific situation can be instantiated to this approach.

Algebraic geometry approach We recall that by Definition 4.2.5 an invariant is a non-zero function $f : \mathbb{Z}[\mathcal{V}_{stb}]^n \rightarrow \mathbb{Z}[\mathcal{V}_{stb}]$ such that $f(\vec{e}) = 0$ for all $\vec{e} \in \mathcal{D}iv$ (with $n = |\mathcal{V}_{div}|$ and $\mathcal{D}iv \subseteq \mathbb{Z}[\mathcal{V}_{stb}]^n$). As we discussed, if we restrict to polynomial invariants then every such function can be represented by a polynomial $f \in \mathbb{Z}[\mathcal{V}_{stb}][\mathcal{V}_{div}]$, where the $\mathcal{V}_{div}$ -variables are used to insert input arguments from $\mathbb{Z}[\mathcal{V}_{stb}]^n$. This description directly connects to a ring-theoretic setting:

We denote $R = \mathbb{Z}[\mathcal{V}_{stb}]$ so that an invariant corresponds to an element in $f \in R[\mathcal{V}_{div}]$. Now, in terminology of ring theory, the set

$$I(\mathcal{D}iv) = \{f \in R[\mathcal{V}_{div}] \mid f(\vec{v}) = 0 \text{ for all } \vec{v} \in \mathcal{D}iv\}$$

is an *ideal* of $R[\mathcal{V}_{div}]$. (So $I(\mathcal{D}iv)$ contains all polynomial invariants, but also $f = 0$.) In a Noetherian ring every ideal is finitely generated [12, Chapter 7], and in the previous paragraph we explained that $R[\mathcal{V}_{div}]$ is Noetherian. So $I(\mathcal{D}iv)$ is finitely generated.

Applying this approach to our setting We can apply this approach in our setting by implementing the following steps:

- (a). heuristically choose a finite subset $V \subseteq \mathcal{D}iv \subseteq \mathbb{Z}[\mathcal{V}_{stb}]^n$
- (b). define $I(V) = \{f \in \mathbb{Z}[\mathcal{V}_{stb}][\mathcal{V}_{div}] \mid f(\vec{v}) = 0 \text{ for all } \vec{v} \in V\}$, being an ideal of $\mathbb{Z}[\mathcal{V}_{stb}][\mathcal{V}_{div}] = \mathbb{Z}[\mathcal{V}_{stb}, \mathcal{V}_{div}]$
- (c). call a method for computing a finite set of generators of $I(V)$

In much of the literature on invariant generation in integer programs, e.g. [3], the computational methods require integers to be interpreted in $\mathbb{Q}$ (or even in the algebraic closure of $\mathbb{Q}$). Consequently, in order to apply this approach to our setting, we should actually redefine steps (a) and (b) above by interpreting $V \subseteq \mathcal{D}iv \subseteq \mathbb{Q}[\mathcal{V}_{stb}]^n$ and correspondingly $I(V) \subseteq \mathbb{Q}[\mathcal{V}_{stb}, \mathcal{V}_{div}]$. In this way, we can apply standard computational methods from algebraic geometry, in which Gröbner basis techniques provide a standard tool for computing ideals of polynomial rings over fields [20].

Note that this transformation is very similar to what we discussed for syzygy computations, where we also changed the base-ring from $\mathbb{Z}$ to $\mathbb{Q}$. Therefore, a natural question is whether there also exist more sophisticated methods that can compute $I(V)$ directly in $\mathbb{Z}[\mathcal{V}_{stb}, \mathcal{V}_{div}]$, instead of $\mathbb{Q}[\mathcal{V}_{stb}, \mathcal{V}_{div}]$, and if not: to what extent does this matter for invariant generation on integer programs? A systematic investigation of this is beyond the scope of this thesis.

4.6 Implementation

The computation of matrix invariants, as well as the computation of InGen-like inequalities (and corresponding derived invariants) is currently not automated in Cora or any other rewriting tool. For the InGen-like invariants and corresponding derived inequalities it is reasonably doable to compute them manually. For matrix invariants it is more complicated to derive them by hand, as this involves a sophisticated algebraic computation to obtain a set of generators for the kernel of a symbolic matrix.

Therefore, to make matrix invariants practically reachable (and to show that the methodology is not merely a theoretically infeasible idea) we automated the computation of matrix invariants as a stand-alone script.

The script is implemented in both GNU Octave and Python (SymPy). Both are available on <https://github.com/kasperhagens/polynomial-invariants.git>. A permanent

(frozen) archive of the website content is available at <https://doi.org/10.5281/zenodo.22343027>. We note that GNU Octave is an open-source alternative to Matlab, which (like Matlab) is provided with a graphical user interface, and which for the computation of matrix invariants internally relies on SymPy (a specific library in Python for symbolic computation). We therefore also included a direct implementation in Python.

The implementation involves a function `invariants`, which computes all matrix invariants of degree d , including *all* degree i monomials for $0 \leq i \leq d$ (i.e. it follows the strategy of Section 4.4 but without the optimization discussed in the last paragraph.)

Invariant function The function `invariants` expects the following input

n : the number of divergence variables

d : the maximum degree of polynomial invariants

χ : the divergence substitution

$\vec{e}_0$: the initial divergence vector

The initial vector $\vec{e}_0$ contains all the variables from $\mathcal{V}_{stb}$ required to compute the divergence space $\mathcal{Div}$. Therefore, $\mathcal{V}_{stb}$ is not required as part of the input. The output is a set of polynomial invariants of $\mathcal{V}_{div}$ -degree at most d .

4.7 Conclusion and future work

In this chapter, we explained that lemma generation in RI for LCSTRSs requires invariant generation, for which the methods from equational reasoning do not suffice. We also showed that the existing InGen method provides a useful starting point in which equality invariants are derived via $\mathcal{R}_{calc}$ -reductions and in which inequality invariants are obtained via (Expansion) steps. However, in practice this method often yields invariants that are insufficiently strong. In addition, InGen may produce non-invariants.

We proposed to interpret integer theory terms as polynomials over $\mathbb{Z}$. This allows us to express invariant generation on simple divergence patterns as a purely algebraic problem, of which the analysis can be delegated to a computer algebra system. Although matrix invariants themselves are often insufficient for lemma generation, they constitute an essential ingredient upon which several heuristics can be built. A typical limitation of matrix invariants is that they cannot generate inequalities. We proposed several strategies for generating inequalities, based on InGen.

Compared to InGen, the simple-divergence based heuristics are less generally applicable. However, the heuristics are specialized to integer programs and therefore more powerful in this specific setting.

Limitations of invariant generation Similar to InGen, the simple-divergence-based heuristics are not suitable for comparing two non-tail-recursive programs, as this results in $\mathcal{V}_{div} = \emptyset$. In such a situation we need to resort to more high-level generalization strategies, based on structural generalization, rather than semantic generalization via invariant generation. In Chapter 6 we introduce such a strategy, based on template transformations.

More high-level strategies may also be necessary to deal with situations where the required invariants are non-polynomial. This, for instance, is the case for the equivalence $\text{factTU } x \approx \text{factTD } x$, comparing two factorial implementations in the following system:

$$\begin{array}{ll}
 \text{factTU } x \rightarrow u \ x \ 1 \ 1 & \text{factTD } x \rightarrow d \ x \ 1 \\
 u \ x \ i \ a \rightarrow a & [i > x] \quad d \ x \ a \rightarrow a \quad [x \leq 0] \\
 u \ x \ i \ a \rightarrow u \ x \ (i + 1) \ (i * a) & [i \leq x] \quad d \ x \ a \rightarrow d \ (x - 1) \ (a * x) \quad [x > 0]
 \end{array}$$

This yields the following simple divergence pattern:

$$\begin{array}{ll}
 u \ x \ i_0 \ a_0 \approx d \ j_0 \ b_0 & [i_0 = 2 \wedge a_0 = 1 \wedge j_0 = x - 1 \wedge b_0 = x \quad \wedge x \geq 1] \\
 u \ x \ i_1 \ a_1 \approx d \ j_1 \ b_1 & [i_1 = 3 \wedge a_1 = 2 \wedge j_1 = x - 2 \wedge b_1 = x * (x - 1) \quad \wedge x \geq 2] \\
 u \ x \ i_2 \ a_2 \approx d \ j_2 \ b_2 & [i_2 = 4 \wedge a_2 = 6 \wedge j_2 = x - 3 \wedge b_2 = x * (x - 1) * (x - 2) \quad \wedge x \geq 3]
 \end{array}$$

However, we are not able to generate a lemma of the form $u \ x \ i \ a \approx d \ j \ b$ because the generalization on the accumulator arguments requires invariants $a = \prod_{k=1}^{i-1} k$ and

$b = \prod_{k=0}^{i-2} (x - k)$, which are not polynomial and therefore cannot be generated with matrix invariants. But even if we could derive these invariants, this would not be useful because the application of lemmas in automated RI systems for LCSTRSs is facilitated by SMT, which do not typically support non-polynomial arithmetic.

Future work Matrix invariants and InGen-like inequalities can only be derived if the divergence pattern is unfolded by the repeated application of a single divergence rule ρ_{div} . In practice, however, it may happen that there are multiple divergence rules:

- We might for example have multiple parallel diverging branches, each of which might describe a simple divergence pattern. Existing research [41] may provide insights for how to deal with such situations.
- Nested loops: this is more complicated, as this likely produces divergence patterns with diverging sub divergence patterns. It is not clear if, and to what extent, matrix invariants and inequality invariants can be adapted to such a situation. Existing research [15] may provide useful insights for how to deal with this.

Chapter 5

Bounded Rewriting Induction

In Chapter 3 we introduced HORI and Global RI as higher-order extensions of the first-order RI system [22]. Both systems follow the same induction strategy as their first-order counterpart, via the (Expansion) rule: at stage i of a deduction sequence we may save an equation $s \approx t [\varphi] \in \mathcal{E}_i$ as an induction rule in $\mathcal{H}_{i+1} = \mathcal{H}_i \cup \{s \rightarrow t [\varphi]\}$ provided that $\mathcal{R} \cup \mathcal{H}_{i+1}$ is terminating. The application of the (Expansion) rule as such is said to *orient* the equation $s \approx t [\varphi]$, which is often written $s > t [\varphi]$, since this process implicitly constructs a well-founded ordering $> = \rightarrow_{\mathcal{R} \cup \mathcal{H}}^+$ where $\mathcal{H} = \bigcup_i \mathcal{H}_i$ denotes the set of all induction rules.

Although, from a methodological point of view, the (Expansion) strategy is very convenient to guarantee correct induction proofs, it has an undesirable practical side effect: even if $\mathcal{R}$ is known to be terminating, it may not be easy or even possible to prove the same for $\mathcal{R} \cup \mathcal{H}$. For instance, in the terminating system $\mathcal{R} = \{\text{add } x \ 0 \rightarrow x, \text{ add } x \ (\text{s } y) \rightarrow \text{s } (\text{add } x \ y)\}$, the equation $\text{add } x \ y \approx \text{add } y \ x$ is not orientable in either direction, hence cannot be used as induction hypothesis.

As already observed by Reddy [43], we do not necessarily need every induction hypothesis to be oriented: it suffices to require that an induction rule $s \rightarrow t [\varphi]$ can only be applied to terms $>$ -smaller than s . Importantly, it is not required to choose the well-founded ordering $> = \rightarrow_{\mathcal{R} \cup \mathcal{H}}^+$. Reddy proposed using modulo rewriting to construct a well-founded order $>$ which does not necessarily contain all induction rules. This approach was investigated by Aoto, who introduced several extensions of RI for first-order unconstrained rewriting [4–6]. In this chapter, we pursue the same goal: minimizing termination requirements. We do so by employing an induction procedure in which the RI process collects the requirements needed to construct a well-founded relation $>$ more carefully than in the HORI methodology, where induction hypotheses need not be oriented.

Chapter setup

- ▶ In Section 5.2 we introduce Bounded RI. This system builds on HORI, but provides a more semantic approach and strictly contains it. Contributions include:
 - We introduce the notion of *bounding pair* $(>, \geq)$, providing the fundamental ingredient for expressing the ordering requirements for induction proofs.
 - We replace equations by *equation contexts*, containing the extra information of two *bounding terms* which define an upper bound for applying an induction hypothesis. More specifically, the bounding terms provide us with a way to keep track of terms to be in used in the ordering requirements (along the equation we are proving) – instead of orienting the induction hypotheses themselves – yielding milder termination requirements.
 - In contrast to the HORI system from Chapter 3, we also include derivation rules to deduce non-equivalence (Section 5.2.3) for ground confluent LCSTRSs. This is thus far only achieved for first-order RI [22].
- ▶ In Section 5.3, we prove soundness and completeness of Bounded RI. Apart from its necessity for the results of Section 5.2 to be meaningful, there are two additional contributions:
 - We show that Bounded RI can be used to prove that equations are *bounded ground convertible*, which is strictly stronger than the property of ground convertibility [27], and which explicitly relates to a bounding pair $(>, \geq)$.
 - We structure the soundness proof in a very modular way, by focusing on two properties that all but one of the deduction rules satisfy. This makes it possible to easily add new derivation rules to the existing system without having to redo the full proof.
- ▶ In Section 5.4 we discuss two strategies to construct a well-founded ordering. We show that we can define the ordering as a reduction relation $\rightarrow_{\mathcal{R} \cup Q}^+$ and briefly discuss strategies to choose the set Q . We also show that we can use a version of the dependency pair framework directly to obtain more liberal requirements.
- ▶ In Section 5.5 we illustrate how the two soundness properties – *Preserving Bounds* (Definition 5.3.1) and *Base Soundness Property* (Definition 5.3.9) – can be exploited to obtain new deduction rules, by extending Bounded RI with some practically useful deduction rules.
- ▶ In Section 5.6 we show that Bounded RI can be used to prove ground confluence of an LCSTRS. This is a known result, but only for first-order TRSs [8].

- We recap the definition of first-order critical pairs for LCTRSs [44], extend it to LCSTRSs and prove the Critical Peak Lemma. This is a first critical peak / pair definition for higher-order constrained systems.
- We introduce the Ground Critical Peak Theorem, enabling the use of RI for proving ground confluence. We follow the first-order approach [8], but make important adaptations to be compatible with Bounded RI.
- In Section 5.7 we briefly discuss the implementation of Bounded RI in Cora, and how it can be used for semi-automatic, human-guided equivalence proofs with fully automatic termination and SMT implication checks.

We relate our contributions to existing work in Section 5.8, and conclude in Section 5.9.

5.1 Preliminaries

We introduce the required assumptions which are necessary to guarantee correctness of Bounded RI, which largely coincide with the assumptions we made to define HORI. The only difference is that we will additionally define infinite rule arity for constructors. This yields an easier definition of the semi constructor terms than in Definition 3.1.6 and allows us to write the deduction rule (Semi-constructor) more conveniently.

Rule arity We again assume that $\mathcal{R}$ satisfies the following properties:

- for all $\ell \rightarrow r [\varphi] \in \mathcal{R}$: ℓ is not a theory term (such rules are contained in $\mathcal{R}_{calc}$)
- for all $f \ell_1 \cdots \ell_k \rightarrow r [\varphi]$, $g \ell'_1 \cdots \ell'_n \rightarrow r' [\psi] \in \mathcal{R} \cup \mathcal{R}_{calc}$: if $f = g$ then $k = n$

As remarked in Chapter 3, the latter restriction blocks us for instance from having both a rule $\text{append nil} \rightarrow \text{id}$ and a rule $\text{append (cons } x \ y) \ z \rightarrow \text{cons } x \ (\text{append } y \ z)$. (The solution here is to replace the first rule by $\text{append nil } x \rightarrow \text{id } x$.) As before, rule-arity is imposed in order to obtain quasi-reductivity.

For every symbol $f :: \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota$ in $\mathcal{D} \cup \Sigma_{calc}$ with $\iota \in \mathcal{S}$, we let $ar(f)$ be the unique number $0 \leq k \leq m$ such that for every rule of the form $f \ell_1 \cdots \ell_k \rightarrow r [\varphi]$ in $\mathcal{R} \cup \mathcal{R}_{calc}$ we have $ar(f) = k$. For all constructors $f \in \mathcal{C}$, we define $ar(f) = \infty$.

Semi-constructor terms Let $\mathcal{L} = (\Sigma, \mathcal{R})$ be some LCSTRS. The semi-constructor terms over $\mathcal{L}$, notation $SCT_{\mathcal{L}}$, are defined by

- (i). $\mathcal{V} \subseteq SCT_{\mathcal{L}}$
- (ii). if $f \in \Sigma$ with $f :: \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota$, $\iota \in \mathcal{S}$ and $s_1 :: \sigma_1, \dots, s_n :: \sigma_n \in SCT_{\mathcal{L}}$ with $n \leq m$ and $n < ar(f)$, then $f s_1 \cdots s_n \in SCT_{\mathcal{L}}$.

Semi-constructor terms are always normal forms. Furthermore, as $ar(f) = \infty$ for every constructor f , the constructor terms $T(C, \mathcal{V})$ are contained in $SCT_{\mathcal{L}}$. However, also terms with partial applications, such as `append (cons  $x$   $y$ )`, are included. The set $SCT_{\mathcal{L}}^0$ refers to *ground* semi-constructor terms, built without (i). A *ground semi-constructor substitution* (abbreviated to *gsc substitution*) is a substitution such that $im(\gamma) \subseteq SCT_{\mathcal{L}}^0$.

Quasi-reductivity We recall: an LCSTRS $\mathcal{L} = (\Sigma, \mathcal{R})$ is quasi-reductive if for every $t \in T(\Sigma, \emptyset)$ we either have $t \in SCT_{\mathcal{L}}^0$ or t reduces with $\rightarrow_{\mathcal{R}}$.

Put differently, the only irreducible ground terms are semi-constructor terms. Weak normalization and quasi-reductivity together ensure that every ground term reduces to a semi-constructor term. Note that, if $s_1, \dots, s_n$ are ground normal forms and $f \in \Sigma$, then $f s_1 \dots s_n$ is a ground normal form if and only if $n < ar(f)$.

Inextensible theory sorts We say that an LCSTRS has *inextensible theory sorts* if the only constructors with a type $\sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow \iota$ with $\iota \in \mathcal{S}_{theory}$ are values (in which case $n = 0$). Hence, it is not for instance allowed to have a constructor `error :: int`. While we do not explicitly limit interest to LCSTRSs with inextensible theory sorts, it is satisfied in all our examples, because in practice it is very difficult to achieve quasi-reductivity without this property.

5.2 Bounded Rewriting Induction

Since the introduction of RI in 1990 by Reddy [43], many variations of the system have appeared [4–6, 21, 22], including HORI and Global RI. All are based on well-founded induction, using some well-founded relation $>$. Depending on how $>$ is being constructed these versions of RI can be categorized into two approaches:

- (1). The first approach [21, 22, 27, 43] employs a fixed strategy to construct a terminating rewrite system $A \supseteq \mathcal{R}$ and then chooses $> = \rightarrow_A^+$. As we explained, this is also the approach being used by (Expansion) in HORI.
- (2). The second approach [4–6] employs a well-founded relation $>$ that satisfies certain requirements (like monotonicity and stability, but also ground totality), and constructs the proof in a more targeted way. This relation may either be fixed beforehand (e.g., the lexicographic path ordering), or constructed during or after the proof, as the proof process essentially accumulates termination requirements.

Approach (1) leads to more severe termination requirements because it forces A to include all (oriented) induction hypotheses (represented by rewrite rules), while (2) is designed to keep termination requirements as mild as possible; for example by orienting

some requirements using a relation $\geq$ rather than $>$. However, approach (2) imposes more bureaucracy, since derivation rules rely on several steps being done at once – for example, by reasoning *modulo* the set of induction hypotheses. This makes it quite hard to use, especially when $>$ is not fixed beforehand but rather constructed on the fly.

With Bounded RI, we aim to combine the best of both worlds. We reduce termination requirements by using a pair $(>, \geq)$, which may either be fixed in advance, or constructed as part of the proof process. In particular, we do not orient induction hypotheses themselves: we only require that a particular instance of the induction hypothesis is allowed to be applied whenever it is strictly dominated by some efficiently chosen *bounding term*, being associated to the particular equation under consideration.

Importantly, we do not impose the ground totality requirement (which would be extremely restrictive in higher-order rewriting), and thus allow for $>$ to for instance be a relation $(\rightarrow_A \cup \triangleright)^+$, or built by a construction based on dependency pairs (see Section 5.4).

First, we specify the desired properties of the orderings $>$ and $\geq$.

Definition 5.2.1 (Ordering Pair). An *ordering pair* on a set $\mathcal{A}$ is a tuple $(>, \geq)$ with

- (i). $>$ a *well-founded partial ordering* (i.e. a transitive, anti-symmetric, irreflexive and well-founded relation) on $\mathcal{A}$
- (ii). $\geq$ a *quasi-order* (i.e. a transitive and reflexive relation) on $\mathcal{A}$

such that $> \subseteq \geq$, and both $> \cdot \geq \subseteq >$ and $\geq \cdot > \subseteq >$ (that is, $a > b \geq c$ and $a \geq b > c$ imply $a > c$, for all $a, b, c \in \mathcal{A}$).

Definition 5.2.2 (Bounding Pair). A *bounding pair* on an LCSTRS $\mathcal{L} = (\Sigma, \mathcal{R})$ is an ordering pair $(>, \geq)$ on $T(\Sigma, \emptyset)$ such that $s \geq t$ whenever $s \rightarrow_{\mathcal{R}} t$ or $s > t$.

We extend an bounding pair $(>, \geq)$ to non-ground terms with a constraint as follows: a substitution γ respects s, t, ψ if $\text{Var}(s, t) \subseteq \text{dom}(\gamma)$ and γ respects ψ . We define

$$\begin{aligned} s > t [\psi] &\iff s\gamma > t\gamma \text{ for all ground substitutions } \gamma \text{ that respect } s, t, \psi \\ s \geq t [\psi] &\iff s\gamma \geq t\gamma \text{ for all ground substitutions } \gamma \text{ that respect } s, t, \psi \end{aligned}$$

5.2.1 Bounded Rewriting Induction

One of the most notable changes in Bounded RI compared to HORI are the proof states. In HORI, every proof state is of the shape $(\mathcal{E}, \mathcal{H})$ with $\mathcal{E}$ a set of equations and $\mathcal{H}$ a set of rewrite rules. In Bounded RI, proof states are still tuples $(\mathcal{E}, \mathcal{H})$, but now $\mathcal{H}$ is a set of equations (i.e. not oriented) and $\mathcal{E}$ is a set of *equation contexts*. An equation context is defined as an expression $(\varsigma ; s \approx t ; \tau) [\psi]$, which can be seen as the combination of an equation $s \approx t [\psi]$ with two *bounding terms* ς, τ . The intuition is that bounding

terms really dictate a bound: we are only allowed to apply induction hypotheses in $\mathcal{H}$ to $s \approx t [\psi]$ if this equation is below the bound determined by ς and τ .

Definition 5.2.3 (Equation context). Let a bounding pair $(>, \geq)$ be given. Let $\bullet$ be a fresh symbol, and define $\bullet > s$ and $\bullet \geq s$ for all $s \in T(\Sigma, \mathcal{V})$, and also $\bullet \geq \bullet$. An *equation context* $(\varsigma ; s \approx t ; \tau) [\psi]$ is a tuple of two elements $\varsigma, \tau \in T(\Sigma, \mathcal{V}) \cup \{\bullet\}$, two terms s, t and a constraint ψ . We write $(\varsigma ; s \approx t ; \tau) [\psi]$ (so with $\approx$ instead of $\approx$) to denote either an equation context $(\varsigma ; s \approx t ; \tau) [\psi]$ or an equation context $(\tau ; t \approx s ; \varsigma) [\psi]$. A substitution γ *respects* an equation context $(\varsigma ; s \approx t ; \tau) [\psi]$ if γ respects ψ and $\text{Var}(\varsigma, s, t, \tau) \subseteq \text{dom}(\gamma)$.

An equation context couples an equation with a bound on the induction: we implicitly use the induction hypothesis: “all ground instances of an equation in $\mathcal{H}$ that are strictly smaller than the current instance of $\varsigma \approx \tau [\psi]$ are convertible”. For example, we might prove the equivalence of two particular implementations of the factorial function, and obtain an induction hypothesis $\text{fact}_1 m \approx \text{fact}_2 m [m \geq 0]$, which we wish to apply to $(\text{fact}_1 n ; \text{fact}_1 k \approx \text{fact}_2 k ; \text{fact}_2 n) [n > 0 \wedge n = k + 1]$. For an appropriately chosen $>$ we have both $\text{fact}_1 n > \text{fact}_1 k [n > 0 \wedge n = k + 1]$ and $\text{fact}_2 n > \text{fact}_2 k [n > 0 \wedge n = k + 1]$. Furthermore, $n > 0 \wedge n = k + 1$ implies $k \geq 0$. So, we can apply the induction hypothesis.

Definition 5.2.4 (Proof state). A proof state is a tuple $(\mathcal{E}, \mathcal{H})$ with $\mathcal{E}$ a set of equation contexts and $\mathcal{H}$ a set of equations.

From Definition 5.2.3 we can see that $\bullet$ behaves as an infinity-term with respect to $>$ and $\geq$. As expressed below in Theorem 5.2.5: when using bounded RI to prove a set of equations, we put them into a set $\mathcal{E}$ of equation contexts using infinite bounds $\varsigma = \tau = \bullet$. This is not problematic, because we always start with the proof state $(\mathcal{E}, \emptyset)$, so there are no induction hypotheses available yet. Once we add an induction hypothesis to the proof state, the bounds will be correctly lowered, as dictated by Figure 5.1 (Induct).

Theorem 5.2.5 (Soundness of Bounded RI). Let $\mathcal{L} = (\Sigma, \mathcal{R})$ be a weakly normalizing, quasi-reductive LCSTRS; $\mathcal{A}$ a set of equations; and let $\mathcal{E}$ be the set of equation contexts $\{(\bullet ; s \approx t ; \bullet) [\psi] \mid s \approx t [\psi] \in \mathcal{A}\}$. Let $(>, \geq)$ be some bounding pair, such that $(\mathcal{E}, \emptyset) \vdash^* (\emptyset, \mathcal{H})$, for some $\mathcal{H}$ using the derivation rules in Figure 5.1. Then every equation in $\mathcal{A}$ is an inductive theorem.

Again, soundness is interpreted via $\mathcal{R}$ -ground convertibility, rather than equality in every possible model of $\mathcal{R}$ (see the discussion in Remark 3.1.2). The proof of Theorem 5.2.5 is provided in Section 5.3.

The deduction rules for Bounded RI are provided in Figure 5.1, and will be explained via a running example in Section 5.2.2. Although it assumes a fixed bounding pair as given, in practice we can leave this pair undecided, use the proof system to collect the necessary requirements, and then select a suitable bounding pair at the end.

(Simplify)

$$\frac{(\mathcal{E} \uplus \{(\zeta ; C[\ell\delta] \simeq t ; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \{(\zeta ; C[r\delta] \approx t ; \tau) [\psi]\}, \mathcal{H})} \quad \ell \rightarrow r [\varphi] \in \mathcal{R} \cup \mathcal{R}_{calc} \text{ and } \psi \models^\delta \varphi$$

(Case)

$$\frac{(\mathcal{E} \uplus \{(\zeta ; s \approx t ; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \{(\zeta\delta ; s\delta \approx t\delta ; \tau\delta) [\psi\delta \wedge \varphi] \mid (\delta, \varphi) \in C\}, \mathcal{H})} \quad \begin{array}{l} C \text{ a cover set of } s \approx t [\psi] \\ \text{(see Definition 5.2.10)} \end{array}$$

(Delete)

$$\frac{(\mathcal{E} \uplus \{(\zeta ; s \approx t ; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E}, \mathcal{H})} \quad \psi \text{ unsatisfiable, or } s = t$$

(Semi-constructor)

$$\frac{(\mathcal{E} \uplus \{(\zeta ; f s_1 \cdots s_n \approx f t_1 \cdots t_n ; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \{(\zeta ; s_i \approx t_i ; \tau) [\psi] \mid 1 \leq i \leq n\}, \mathcal{H})} \quad n > 0 \text{ and } (f \in \mathcal{V} \text{ or } n < ar(f))$$

(Induct)

$$\frac{(\mathcal{E} \uplus \{(\zeta ; s \approx t ; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \{(s ; s \approx t ; t) [\psi]\}, \mathcal{H} \cup \{s \approx t [\psi]\})}$$

(Hypothesis)

$$\frac{(\mathcal{E} \uplus \{(\zeta ; C[\ell\delta] \simeq t ; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \{(\zeta ; C[r\delta] \approx t ; \tau) [\psi]\}, \mathcal{H})} \quad \begin{array}{l} \ell \simeq r [\varphi] \in \mathcal{H} \text{ and } \psi \models^\delta \varphi \text{ and} \\ \zeta > \ell\delta [\psi] \text{ and } \zeta > r\delta [\psi] \text{ and } \zeta \geq C[r\delta] [\psi] \end{array}$$

(H-Delete)

$$\frac{(\mathcal{E} \uplus \{(\zeta ; C[\ell\delta] \simeq C[r\delta] ; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E}, \mathcal{H})} \quad \begin{array}{l} \ell \simeq r [\varphi] \in \mathcal{H} \text{ and } \psi \models^\delta \varphi \text{ and} \\ \zeta > \ell\delta [\psi] \text{ or } \tau > r\delta [\psi] \end{array}$$

(Generalize)/(Alter)

$$\frac{(\mathcal{E} \uplus \{(\zeta ; s \approx t ; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \{(\zeta' ; s' \approx t' ; \tau') [\psi']\}, \mathcal{H})} \quad \begin{array}{l} (\zeta' ; s' \approx t' ; \tau') [\psi'] \text{ generalizes/alters } (\zeta ; s \approx t ; \tau) [\psi] \\ \text{(see Definition 5.2.11), and } \zeta' \geq s' [\psi'] \text{ and } \tau' \geq t' [\psi'] \end{array}$$

(Postulate)

$$\frac{(\mathcal{E}, \mathcal{H})}{(\mathcal{E} \cup \{(\bullet ; s \approx t ; \bullet) [\psi]\}, \mathcal{H})}$$

Fig. 5.1 Derivation rules for Bounded Rewriting Induction, given a bounding pair $(>, \geq)$.

First, we introduce notation that compactly expresses the constraint implications appearing in some deduction rules of Figure 5.1.

Definition 5.2.6 ($\models^\delta$). Let δ be a substitution and φ, ψ be constraints. We write $\psi \models^\delta \varphi$ if $\delta(\text{Var}(\varphi)) \subseteq \text{Val} \cup \text{Var}(\psi)$, and $\psi \implies \varphi\delta$ is a valid constraint.

Hence, if $\psi \models^\delta \varphi$ and γ is a substitution that respects ψ , then γ also respects $\varphi\delta$. This is for example used in the deduction rule (Simplify) to ensure that every ground instance of an equation $C[\ell\delta] \approx t[\psi]$ can be reduced with a rewrite rule $\ell \rightarrow r[\varphi]$.

Example 5.2.7. Consider the equation RD $f\ i' i\ a \approx t\ [i' = i + 1]$ and rewrite rule RD $f\ n\ i\ a \rightarrow a\ [i < n]$, and let $\delta = [n := i']$. We have $(i < n)\delta = i < i'$ and $(i' = i + 1) \models^\delta (i < n)$ holds, because

- ▷ $\delta(\text{Var}(i < n)) = \delta(\{i, n\}) = \{i, i'\} \subseteq \text{Var}(i' = i + 1) = \{i, i'\}$
- ▷ $(i' = i + 1) \implies (i < n)\delta$ is valid

In Section 5.3.1 we explain that the proof system in Figure 5.1 satisfies a property *Preserving Bounds*, meaning that it only produces *bounded* equation contexts:

Definition 5.2.8 (Bounded equation context). A *bounded equation context* is an equation context $(\varsigma ; s \approx t ; \tau) [\psi]$ such that both $\varsigma \geq s[\psi]$ and $\tau \geq t[\psi]$.

Bounded equation contexts play an important role in the soundness proof, since it allows us to avoid ordering requirements that would otherwise need to be added to Figure 5.1.

In many practical cases, we will also consider an even stronger restriction:

Definition 5.2.9 (Strongly bounded equation context). An equation context $(\varsigma ; s \approx t ; \tau) [\psi]$ is *strongly bounded* if either $\varsigma = s$ or $\varsigma > s[\psi]$, and also either $\tau = t$ or $\tau > t[\psi]$.

Clearly, all equation contexts at the start of a RI deduction are strongly bounded, as all initial equation contexts have $\varsigma = \tau = \bullet$. At first, it may not be clear why *strongly bounded* is a strictly stronger property than *bounded*. To illustrate this, consider the following example with a well-founded ordering $>$ whose restriction to integers is defined as

$$n > m \iff n > m \wedge n \geq 0$$

Then $n \geq m [n \geq m \wedge n \geq 0]$ holds, but we have neither $n = m$ nor $n > m [n \geq m \wedge n \geq 0]$.

In practice, we highly benefit from strong boundedness because many of the required inequalities in Figure 5.1 can be verified by a straightforward syntactic check: whenever $\varsigma \neq s$, we immediately conclude $\varsigma > s[\psi]$. While strong boundedness is *not* necessarily preserved by the rules of Figure 5.1, we can maintain it by a suitable choice of reduction pair and careful application of the (Hypothesis), (Generalize), and (Alter) rules.

5.2.2 Explanation by example

We will now elaborate on the deduction rules of Figure 5.1, and illustrate their use by two running examples. Along the way, we will also introduce the definitions of *cover set* and *generalizing / altering*, as referred to by Figure 5.1.

We consider the LCSTRS defined by

$$\mathcal{R} = \left\{ \begin{array}{ll} \text{(R1)} & \text{RD } f \ n \ i \ a \rightarrow a \quad [i < n] \\ \text{(R2)} & \text{RD } f \ n \ i \ a \rightarrow f \ i \ (\text{RD } f \ n \ (i-1) \ a) \quad [i \geq n] \\ \text{(R3)} & \text{TU } f \ i \ m \ a \rightarrow a \quad [i > m] \\ \text{(R4)} & \text{TU } f \ i \ m \ a \rightarrow \text{TU } f \ (i+1) \ m \ (f \ i \ a) \quad [i \leq m] \end{array} \right\}$$

and use Bounded RI to prove $\text{RD } f \ n \ i \ a \approx \text{TU } f \ n \ i \ a$. Following Theorem 5.2.5, we aim to show that there is a set $\mathcal{H}$ such that

$$(\mathcal{E}_1, \emptyset) \vdash^* (\emptyset, \mathcal{H}) \text{ with } \mathcal{E}_1 = \{(\bullet; \text{RD } f \ n \ i \ a \approx \text{TU } f \ n \ i \ a; \bullet) [\text{true}]\}$$

We do *not* fully choose the bounding pair $(>, \geq)$ in advance. Rather, we will use the proof process to accumulate requirements on the well-founded ordering $>$ to be used. However, we precommit to a bounding pair where $\geq$ is the reflexive closure of $>$ and $\rightarrow_{\mathcal{R}} \cup \triangleright \subseteq >$. With these assumptions, strongly bounded equation contexts are easily maintained as an invariant throughout a Bounded RI deduction sequence.

(Induct) In HORI, the (Expansion) rule is used us to do a case analysis and create an induction hypothesis at the same time. In Bounded RI, we will consider these as two separate actions. To start an induction proof, we use the following rule

$$\frac{(\mathcal{E} \uplus \{(\varsigma; s \approx t; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \{(s; s \approx t; t) [\psi]\}, \mathcal{H} \cup \{s \approx t [\psi]\})}$$

Here, two things happen. First, the current equation is added to the set $\mathcal{H}$ of induction hypotheses, making it available for later application of (Hypothesis) or ($\mathcal{H}$ -Delete). Second, the bounding terms ς, τ are replaced by s, t . This ensures that, when an equation in $\mathcal{H}$ is applied, it is only on equations that are strictly smaller than $s \approx t [\psi]$.

In the running example, we use (Induct) to obtain $(\mathcal{E}_1, \emptyset) \vdash (\mathcal{E}_2, \mathcal{H}_2)$, where

$$\begin{aligned} \mathcal{E}_2 &= \{(\varsigma_2; \text{RD } f \ n \ i \ a \approx \text{TU } f \ n \ i \ a; \tau_2) [\text{true}]\} \\ \mathcal{H}_2 &= \{\text{RD } f \ n \ i \ a \approx \text{TU } f \ n \ i \ a\} \end{aligned}$$

We will retain $\varsigma_2 = \text{RD } f \ n \ i \ a$ and $\tau_2 = \text{TU } f \ n \ i \ a$ for later reference.

(Case) To continue with the equation in $\mathcal{E}_2$, we should decompose it into the cases $i < n$ and $i \geq n$, e.g. by using the two RD-rules. In HORI this was performed automatically

by (Expansion), but here this is treated as a separate operation. Bounded RI does not require to create a decomposition based on the rules in $\mathcal{R}$: any case distinction is allowed, provided that all cases together cover the original equation.

Definition 5.2.10 (Cover set). A cover set of an equation $s \approx t [\psi]$ is a set C of pairs (δ, φ) , with δ a substitution and φ a constraint, such that for every gsc substitution γ respecting $s \approx t [\psi]$, there exists $(\delta, \varphi) \in C$ and a substitution σ such that $\gamma(x) = \delta(x)\sigma$ for all $x \in \text{dom}(\gamma)$, and σ respects $\psi\delta \wedge \varphi$. (Hence, $s\gamma \approx t\gamma [\psi\gamma]$ is an instance of $s\delta \approx t\delta [\psi\delta \wedge \varphi]$.)

Now, the deduction rule (Case) reads as follows:

$$\frac{(\mathcal{E} \uplus \{(\varsigma; s \approx t; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \{(\varsigma\delta; s\delta \approx t\delta; \tau\delta) [\psi\delta \wedge \varphi] \mid (\delta, \varphi) \in C\}, \mathcal{H})} \quad \begin{array}{l} C \text{ a cover set of } s \approx t [\psi] \\ \text{(see Definition 5.2.10)} \end{array}$$

Continuing the running example, we observe the only gsc terms of type `int` are values (since we have inextensible theory sorts). Hence, $C = \{([], i < n), ([], i \geq n)\}$ is a cover set of $\text{RD } f \ n \ i \ a \approx \text{TU } f \ n \ i \ a$. Using (Case), we obtain $(\mathcal{E}_2, \mathcal{H}_2) \vdash (\mathcal{E}_3, \mathcal{H}_2)$ with

$$\mathcal{E}_3 = \left\{ \begin{array}{l} (\varsigma_2; \text{RD } f \ n \ i \ a \approx \text{TU } f \ n \ i \ a; \tau_2) [i < n] \\ (\varsigma_2; \text{RD } f \ n \ i \ a \approx \text{TU } f \ n \ i \ a; \tau_2) [i \geq n] \end{array} \right\}$$

Note that the bounding terms ς_2, τ_2 are unchanged because the substitutions in the cover set were both empty. This is, however, not true in general (as we will see later in this section). Strong boundedness is still satisfied, by $\varsigma = s$ and $\tau = t$.

(Simplify) Due to the notation $\models^\delta$ we can now, compared to HORI, present the (Simplify) rule more compactly:

$$\frac{(\mathcal{E} \uplus \{(\varsigma; C[\ell\delta] \simeq t; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \{(\varsigma; C[r\delta] \approx t; \tau) [\psi]\}, \mathcal{H})} \quad \ell \rightarrow r [\varphi] \in \mathcal{R} \cup \mathcal{R}_{\text{calc}} \text{ and } \psi \models^\delta \varphi$$

So, the requirement $\psi \models^\delta \varphi$ guarantees that the δ -instance of $\ell \rightarrow r [\varphi]$ is actually applicable. The bounding terms are not affected by the reduction.

Note that, in contrast to HORI, (Simplify) cannot be used to apply rewrite steps with $\mathcal{H}$. For this, we now have the deduction rules ($\mathcal{H}$ -Delete) and (Hypothesis).

Continuing the running example, the first equation in $\mathcal{E}_3$ has constraint $i < n$, so we apply (Simplify) on both sides, using **(R1)** and **(R3)**. For the second equation, we also apply (Simplify) to both sides, using **(R2)** and **(R4)**. We obtain $(\mathcal{E}_3, \mathcal{H}_2) \vdash^* (\mathcal{E}_4, \mathcal{H}_2)$ with

$$\mathcal{E}_4 = \left\{ \begin{array}{l} (\varsigma_2; a \approx a; \tau_2) \quad [i < n] \\ (\varsigma_2; f \ i \ (\text{RD } f \ n \ (i-1) \ a) \approx \text{TU } f \ (n+1) \ i \ (f \ n \ a); \tau_2) \quad [i \geq n] \end{array} \right\}$$

Note that for both equations $(\varsigma_2 ; s \approx t ; \tau_2) [\psi]$ we now have the property: $\varsigma_2 > s [\psi]$ and $\tau_2 > t [\psi]$, since $\rightarrow_{\mathcal{R}}$ is included in $>$. Thus, strong boundedness is preserved.

(Delete) The rule for deleting an equation remains unchanged:

$$\frac{(\mathcal{E} \uplus \{(\varsigma ; s \approx t ; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E}, \mathcal{H})} \quad \psi \text{ unsatisfiable, or } s = t$$

In the running example, we use (Delete) we obtain $(\mathcal{E}_4, \mathcal{H}_2) \vdash (\mathcal{E}_5, \mathcal{H}_2)$ with

$$\mathcal{E}_5 = \{(\varsigma_2 ; f \ i \ (\text{RD } f \ n \ (i - 1) \ a) \approx \text{TU } f \ (n + 1) \ i \ (f \ n \ a) ; \tau_2) [i \geq n]\}$$

(Alter) The (Alter) definition from HORI is extended to the setting of equation contexts, where we now additionally have to guarantee that no bounding terms is increased.

Definition 5.2.11. We say that an equation context $(\varsigma' ; s' \approx t' ; \tau') [\psi']$

► *generalizes* $(\varsigma ; s \approx t ; \tau) [\psi]$ if for every gsc substitution γ that respects $(\varsigma ; s \approx t ; \tau) [\psi]$ there is a substitution δ that respects $(\varsigma' ; s' \approx t' ; \tau') [\psi']$ such that $s\gamma = s'\delta$ and $t\gamma = t'\delta$, and $\varsigma\gamma \geq \varsigma'\delta$ and $\tau\gamma \geq \tau'\delta$.

► *alters* $(\varsigma ; s \approx t ; \tau) [\psi]$ if both

- $(\varsigma' ; s' \approx t' ; \tau') [\psi']$ generalizes $(\varsigma ; s \approx t ; \tau) [\psi]$, and
- $(\varsigma ; s \approx t ; \tau) [\psi]$ generalizes $(\varsigma' ; s' \approx t' ; \tau') [\psi']$.

Now, (Alter) is defined as follows

$$\frac{(\mathcal{E} \uplus \{(\varsigma ; s \approx t ; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \{(\varsigma' ; s' \approx t' ; \tau') [\psi']\}, \mathcal{H})} \quad \begin{array}{l} (\varsigma' ; s' \approx t' ; \tau') [\psi'] \text{ alters } (\varsigma ; s \approx t ; \tau) [\psi] \\ \text{and } \varsigma' \geq s' [\psi] \text{ and } \tau' \geq t' [\psi] \end{array}$$

There are many ways to use this deduction rule, but following the discussion in Section 3.2.1, we will particularly consider two ways:

- (I) Replacing a constraint by an equivalent one: that is, altering $(\varsigma ; s \approx t ; \tau) [\psi]$ into $(\varsigma ; s \approx t ; \tau) [\psi']$ if $(\exists \vec{x}. \psi) \iff (\exists \vec{y}. \psi')$ is logically valid, where $\{\vec{x}\} = \text{Var}(\psi) \setminus \text{Var}(\varsigma, s, t, \tau)$ and $\{\vec{y}\} = \text{Var}(\psi') \setminus \text{Var}(\varsigma, s, t, \tau)$.

This assumes that the system has inextensible theory sorts; if not, we must also require that $\text{Var}(\psi) \setminus \{\vec{x}\} = \text{Var}(\psi') \setminus \{\vec{y}\}$.

A particular example of this case is to replace $(\varsigma ; s \approx t ; \tau) [\psi]$ by $(\varsigma ; s \approx t ; \tau) [\psi \wedge x_1 = u_1 \wedge \dots \wedge x_n = u_n]$, where all x_i are fresh variables, with $x_i \notin \text{Var}(u_j)$ for $j \leq i$.

- (II) Replacing variables by equivalent variables or values: that is, altering $(\varsigma ; s \approx t ; \tau) [\psi]$ into $(\varsigma\gamma ; s\gamma \approx t\gamma ; \tau\gamma) [\psi]$ if $\gamma = [x_1 := u_1, \dots, x_n := u_n]$ and

$\psi \implies x_1 = u_1 \wedge \dots \wedge x_n = u_n$ is valid, where $x_1, \dots, x_n$ are variables and $u_1, \dots, u_n$ are each variables or values.

Back to the example: by case (I), we use (Alter) to obtain $(\mathcal{E}_5, \mathcal{H}_2) \vdash (\mathcal{E}_6, \mathcal{H}_2)$, with

$$\mathcal{E}_6 = \left\{ \begin{array}{l} (\varsigma_2 ; f \ i \ (\text{RD } f \ n \ (i-1) \ a) \approx \text{TU } f \ (n+1) \ i \ (f \ n \ a) ; \tau_2) \\ [i' = i-1 \wedge n' = n+1 \wedge i \geq n] \end{array} \right\}$$

To allow this rule to be applied, we must have $\varsigma_2 \geq f \ i \ (\text{RD } f \ n \ (i-1) \ a) [\varphi]$ and $\tau_2 \geq \text{TU } f \ (n+1) \ i \ (f \ n \ a) [\varphi]$ where φ is the constraint $i' = i-1 \wedge n' = n+1 \wedge i \geq n$. But this follows immediately from the fact that the previous equation context was strongly bounded: if $\varsigma > s [i \geq n]$ then also $\varsigma > s [i' = i-1 \wedge n' = n+1 \wedge i \geq n]$, and similar for $\tau > t [i \geq n]$. Moreover, strong boundedness is clearly still satisfied.

Remark 5.2.12. The preservation of strong boundedness in the running example is not a coincidence: we will see in Lemma 5.3.3 that most of the deduction rules of Figure 5.1 preserve this property automatically if $>$ includes $\rightarrow_{\mathcal{R}}$ and $\triangleright$. In (Alter) and (Generalize) this is not necessarily the case, but in Lemma 5.3.4 we prove that it holds for (Alter) when used as in (I) or (II).

We continue the example by two successive (Simplify) steps, using calculation rules $i-1 \rightarrow i' [i' = i-1]$ and $n+1 \rightarrow n' [n' = n+1]$ to obtain $(\mathcal{E}_7, \mathcal{H}_2)$, with

$$\mathcal{E}_7 = \left\{ \begin{array}{l} (\varsigma_2 ; f \ i \ (\text{RD } f \ n \ i' \ a) \approx \text{TU } f \ n' \ i \ (f \ n \ a) ; \tau_2) \\ [i' = i-1 \wedge n' = n+1 \wedge i \geq n] \end{array} \right\}$$

Note that these steps were only possible because of the (Alter) step that preceded them.

(Again, strong boundedness is preserved because $>$ includes $\rightarrow_{\mathcal{R}}$.)

(Hypothesis) Similar to (Simplify), we can use an induction hypothesis to reduce either side of an equation. Here, finally, the bounding terms ς, τ come into play, as we need to ensure a decrease to facilitate the well-founded induction.

$$\frac{(\mathcal{E} \uplus \{(\varsigma ; C[\ell\delta] \simeq t ; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \{(\varsigma ; C[r\delta] \approx t ; \tau) [\psi]\}, \mathcal{H})} \quad \begin{array}{l} \ell \simeq r [\varphi] \in \mathcal{H} \text{ and } \psi \models^{\delta} \varphi \text{ and} \\ \varsigma > \ell\delta [\psi] \text{ and } \varsigma > r\delta [\psi] \text{ and } \varsigma \geq C[r\delta] [\psi] \end{array}$$

We use (Hypothesis) to reduce the left-hand side of the equation in $\mathcal{E}_7$ with the only induction hypothesis from $\mathcal{H}_2$ in the direction $\text{RD } f \ n \ i \ a \rightarrow \text{TU } f \ n \ i \ a$, with substitution $[i := i']$. This allows us to deduce $(\mathcal{E}_7, \mathcal{H}_2) \vdash (\mathcal{E}_8, \mathcal{H}_2)$ with

$$\mathcal{E}_8 = \left\{ \begin{array}{l} (\varsigma_2 ; f \ i \ (\text{TU } f \ n \ i' \ a) \approx \text{TU } f \ n' \ i \ (f \ n \ a) ; \tau_2) \\ [i' = i-1 \wedge n' = n+1 \wedge i \geq n] \end{array} \right\}$$

However, to be allowed to apply this deduction rule, we must show that the $>$ requirements are satisfied; that is, that we have

$$\begin{aligned} \text{RD } f \ n \ i \ a &> \text{RD } f \ n \ i' \ a & [i' = i - 1 \wedge n' = n + 1 \wedge i \geq n] \\ \text{RD } f \ n \ i \ a &> \text{TU } f \ n \ i' \ a & [i' = i - 1 \wedge n' = n + 1 \wedge i \geq n] \\ \text{RD } f \ n \ i \ a &\geq f \ i \ (\text{TU } f \ n \ i' \ a) & [i' = i - 1 \wedge n' = n + 1 \wedge i \geq n] \end{aligned}$$

The first is satisfied by the strong boundedness property. The second is an immediate consequence of the third, since $f \ i \ (\text{TU } f \ n \ i' \ a) \triangleright \text{TU } f \ n \ i' \ a$ and we have committed to let $\triangleright$ be included in $>$. For the third, we remember the following requirement:

$$\text{(REQ1)} \quad \text{RD } f \ n \ i \ a > f \ i \ (\text{TU } f \ n \ i' \ a) \quad [i' = i - 1 \wedge n' = n + 1 \wedge i \geq n]$$

Here, we have replaced the $\geq$ by a $>$ to ensure that strong boundedness is preserved. Since $\geq$ is the reflexive closure of $>$, this is not actually a stronger requirement.

Let $\varsigma_9 = f \ i \ (\text{TU } f \ n \ i' \ a)$, $\tau_9 = \text{TU } f \ n' \ i \ (f \ n \ a)$, and apply (Induct) to $(\mathcal{E}_8, \mathcal{H}_2)$. We then obtain the proof state $(\mathcal{E}_9, \mathcal{H}_9)$, defined as follows:

$$\begin{aligned} \mathcal{E}_9 &= \left\{ \begin{array}{l} (\varsigma_9 ; f \ i \ (\text{TU } f \ n \ i' \ a) \approx \text{TU } f \ n' \ i \ (f \ n \ a) ; \tau_9) \\ [i' = i - 1 \wedge n' = n + 1 \wedge i \geq n] \end{array} \right\} \\ \mathcal{H}_9 &= \left\{ \begin{array}{l} \text{RD } f \ n \ i \ a \approx \text{TU } f \ n \ i \ a \\ f \ i \ (\text{TU } f \ n \ i' \ a) \approx \text{TU } f \ n' \ i \ (f \ n \ a) \quad [i' = i - 1 \wedge n' = n + 1 \wedge i \geq n] \end{array} \right\} \end{aligned}$$

We now use (Case) again to split the constraint in $\mathcal{E}_9$ into $i = n$ and $i > n$ to obtain:

$$\mathcal{E}_{10} = \left\{ \begin{array}{l} (\varsigma_9 ; f \ i \ (\text{TU } f \ n \ i' \ a) \approx \text{TU } f \ n' \ i \ (f \ n \ a) ; \tau_9) \\ [i' = i - 1 \wedge n' = n + 1 \wedge i = n] \\ (\varsigma_9 ; f \ i \ (\text{TU } f \ n \ i' \ a) \approx \text{TU } f \ n' \ i \ (f \ n \ a) ; \tau_9) \\ [i' = i - 1 \wedge n' = n + 1 \wedge i > n] \end{array} \right\}$$

Observing that $i' = i - 1 \wedge n' = n + 1 \wedge i = n$ implies both $n > i'$ and $n' > i$, and that $i' = i - 1 \wedge n' = n + 1 \wedge i > n$ implies both $n \leq i'$ and $n' \leq i$, we use (Simplify) on both sides of the first equation with (R3) and on both sides of the second equation with (R4) respectively, to deduce $(\mathcal{E}_{10}, \mathcal{H}_9) \vdash^* (\mathcal{E}_{11}, \mathcal{H}_9)$, with

$$\mathcal{E}_{11} = \left\{ \begin{array}{l} (\varsigma_9 ; f \ i \ a \approx f \ n \ a ; \tau_9) \quad [i' = i - 1 \wedge n' = n + 1 \wedge i = n] \\ (\varsigma_9 ; f \ i \ (\text{TU } f \ (n + 1) \ i' \ (f \ n \ a)) \approx \text{TU } f \ (n' + 1) \ i \ (f \ n' \ (f \ n \ a)) ; \tau_9) \\ [i' = i - 1 \wedge n' = n + 1 \wedge i > n] \end{array} \right\}$$

Note that the first equation above does not yet satisfy the requirements for (Delete), even though the $i = n$ part of the constraint makes it look very delete-worthy. We resolve

this by using (Alter) (case (II)), replacing the first equation context by $(\varsigma_9 ; f \ n \ a \approx f \ n \ a ; \tau_9) [i' = i - 1 \wedge n' = n + 1 \wedge i = n]$, after which it can immediately be deleted. Also using (Alter) (now case (I)) on the second equation, and then using (Simplify) with calculation rules as we did before, we are left with $(\mathcal{E}_{12}, \mathcal{H}_9)$, where

$$\mathcal{E}_{12} = \left\{ \begin{array}{l} (\varsigma_9 ; f \ i \ (\text{TU } f \ n' \ i' \ (f \ n \ a)) \approx \text{TU } f \ n'' \ i \ (f \ n' \ (f \ n \ a)) ; \tau_9) \\ [i' = i - 1 \wedge n' = n + 1 \wedge n'' = n' + 1 \wedge i > n] \end{array} \right\}$$

($\mathcal{H}$ -Delete). With this deduction rule we may delete any equation that has a subequation which is an instance of an equation in $\mathcal{H}$. The rule looks very similar to (Hypothesis), but with lighter requirements on $>$.

$$\frac{(\mathcal{E} \uplus \{(\varsigma ; C[\ell\delta] \simeq C[r\delta] ; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E}, \mathcal{H})} \quad \begin{array}{l} \ell \simeq r [\varphi] \in \mathcal{H} \text{ and } \psi \models^\delta \varphi \text{ and} \\ \varsigma > \ell\delta [\psi] \text{ or } \tau > r\delta [\psi] \end{array}$$

Considering $\mathcal{H}_9$, and renaming the variables to avoid confusion, the second hypothesis reads: $g \ x \ (\text{TU } g \ y \ x' \ z) \approx \text{TU } g \ y' \ x \ (g \ y \ z) [x' = x - 1 \wedge y' = y + 1 \wedge x \geq y]$. Let δ be the substitution $[g := f, x := i, x' := i', y := n', y' := n'', z := f \ n \ a]$. We can now deduce $(\mathcal{E}_{12}, \mathcal{H}_9) \vdash (\emptyset, \mathcal{H}_9)$, provided one of the following requirements is satisfied:

$$\begin{array}{ll} \varsigma_9 = & f \ i \ (\text{TU } f \ n \ i' \ a) > f \ i \ (\text{TU } f \ n' \ i' \ (f \ n \ a)) \quad [i' = i - 1 \wedge n' = n + 1 \wedge i \geq n] \\ \tau_9 = & \text{TU } f \ n' \ i \ (f \ n \ a) > \text{TU } f \ n'' \ i \ (f \ n' \ (f \ n \ a)) \quad [i' = i - 1 \wedge n' = n + 1 \wedge i \geq n] \end{array}$$

Both are satisfied by the strong boundedness property so there is no check required.

So we have shown $(\mathcal{E}_1, \emptyset) \vdash^* (\emptyset, \mathcal{H}_9)$, hence by Theorem 5.2.5 the equation $\text{RD } f \ n \ i \ a \approx \text{TU } f \ n \ i \ a$ is an inductive theorem, provided we indeed have a suitable bounding pair that satisfies **(REQ1)**. But this is easily achieved: we let $>$ be $(\rightarrow_{\mathcal{R} \cup \mathcal{Q}} \cup \triangleright)^+$ where $\mathcal{Q} = \{\text{RD } f \ n \ i \ a \rightarrow f \ i \ (\text{TU } f \ n \ i' \ a) [i' = i - 1 \wedge n' = n + 1 \wedge i \geq n]\}$. This is a bounding pair because $\rightarrow_{\mathcal{R} \cup \mathcal{Q}}$ is terminating (which can for instance be proved using static dependency pairs [24]).

Remark 5.2.13. The choice to take $> = (\rightarrow_{\mathcal{R} \cup \mathcal{Q}} \cup \triangleright)^+$ is quite natural: in many traditional RI systems [21, 22, 27, 43] this is the only choice for $(>, \geq)$, with $\mathcal{Q}$ always being a directed version of the final $\mathcal{H}$ (so in the case of this example, $\mathcal{H}_9$). However, while such a choice is natural in strategies for rewriting induction, we leave it open in the definition to allow for alternative orderings, as we will discuss in Section 5.4.

An example with structural induction We continue with another example, both to introduce the remaining deduction rules and to illustrate that the method can be used not only for induction on integers, but also for structural induction on terms. Consider

$\Sigma_{terms} = \{\text{nil} :: \text{list}, \text{cons} :: \text{int} \rightarrow \text{list} \rightarrow \text{list}, \text{app}, \text{rev} :: \text{list} \rightarrow \text{list} \rightarrow \text{list}\}$ with rules

$$\begin{array}{ll} \text{(R1)} \quad \text{app nil } ys \rightarrow ys & \text{(R2)} \quad \text{app (cons } x \text{ } xs) \text{ } ys \rightarrow \text{cons } x \text{ (app } xs \text{ } ys) \\ \text{(R3)} \quad \text{rev nil } ys \rightarrow ys & \text{(R4)} \quad \text{rev (cons } x \text{ } xs) \text{ } ys \rightarrow \text{rev } xs \text{ (cons } x \text{ } ys) \end{array}$$

Suppose we wish to show that $\text{rev (app } xs \text{ } ys) \text{ nil} \approx \text{app (rev } ys \text{ nil) (rev } xs \text{ nil)}$ is an inductive theorem. We start with $\mathcal{E}_1$ containing this equation coupled with bullets as bounding terms, and after an (Induct) step we obtain $(\mathcal{E}_2, \mathcal{H}_2)$, where

$$\begin{array}{ll} \mathcal{E}_2 &= \{(\varsigma_2 ; \text{rev (app } xs \text{ } ys) \text{ nil} \approx \text{app (rev } ys \text{ nil) (rev } xs \text{ nil)} ; \tau_2)\} \\ \mathcal{H}_2 &= \{\text{rev (app } xs \text{ } ys) \text{ nil} \approx \text{app (rev } ys \text{ nil) (rev } xs \text{ nil)}\} \\ \varsigma_2 &= \text{rev (app } xs \text{ } ys) \text{ nil} \\ \tau_2 &= \text{app (rev } ys \text{ nil) (rev } xs \text{ nil)} \end{array}$$

(We will omit the constraint from equation contexts when it is just $[\text{true}]$.)

Now we apply (Case), using the cover set $([xs := \text{cons } a \text{ } as], \text{true}), ([xs := \text{nil}], \text{true})$. This is indeed a cover set because every ground semiconstructor instance of xs must be either nil or headed by the list constructor cons . This yields $(\mathcal{E}_2, \mathcal{H}_2) \vdash (\mathcal{E}_3, \mathcal{H}_3)$, with

$$\mathcal{E}_3 = \left\{ \begin{array}{l} (\varsigma_3 ; \text{rev (app (cons } a \text{ } as) \text{ } ys) \text{ nil} \approx \text{app (rev } ys \text{ nil) (rev (cons } a \text{ } as) \text{ nil)} ; \tau_3) \\ (\varsigma_4 ; \text{rev (app nil } ys) \text{ nil} \approx \text{app (rev } ys \text{ nil) (rev nil nil)} ; \tau_4) \end{array} \right\}$$

where

$$\begin{array}{ll} \varsigma_3 &= \text{rev (app (cons } a \text{ } as) \text{ } ys) \text{ nil} & \varsigma_4 &= \text{rev (app nil } ys) \text{ nil} \\ \tau_3 &= \text{app (app (cons } a \text{ } as) \text{ } ys) \text{ } zs & \tau_4 &= \text{app (app nil } ys) \text{ } zs \end{array}$$

Note that here the bounding terms are substituted along with the equation. This is the only deduction rule that does so.

After a few (Simplify) steps on both equations, we deduce $(\mathcal{E}_3, \mathcal{H}_2) \vdash^* (\mathcal{E}_4, \mathcal{H}_2)$:

$$\mathcal{E}_4 = \left\{ \begin{array}{l} (\varsigma_3 ; \text{rev (app } as \text{ } ys) \text{ (cons } a \text{ nil)} \approx \text{app (rev } ys \text{ nil) (rev } as \text{ (cons } a \text{ nil))} ; \tau_3) \\ (\varsigma_4 ; \text{rev } ys \text{ nil} \approx \text{app (rev } ys \text{ nil) nil} ; \tau_4) \end{array} \right\}$$

(Generalize) To continue on $\mathcal{E}_4$ we should apply a generalization, for which the corresponding induction rule is defined similar as in HORI:

$$\frac{(\mathcal{E} \uplus \{(\varsigma ; s \approx t ; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \{(\varsigma' ; s' \approx t' ; \tau') [\varphi]\}, \mathcal{H})} \quad \begin{array}{l} (\varsigma' ; s' \approx t' ; \tau') [\varphi] \text{ generalizes } (\varsigma ; s \approx t ; \tau) [\psi] \\ \text{and } \varsigma' \geq s' [\psi] \text{ and } \tau' \geq t' [\psi] \end{array}$$

As mentioned in Chapter 3, (Generalize) is quite similar to (Alter) – and every step that can be done by (Alter) can also be done by (Generalize) – but they are used quite

differently: (Alter) is designed to set up an equation for the use of simplification or deletion, while (Generalize) is a form of lemma generation.

In the second equation of $\mathcal{E}_4$, we use (Generalize) to abstract the term $\text{rev } ys \text{ nil}$ into a fresh variable zs , deriving $(\mathcal{E}_4, \mathcal{H}_2) \vdash (\mathcal{E}_5, \mathcal{H}_2)$, with

$$\mathcal{E}_5 = \left\{ \begin{array}{l} (\zeta_3 ; \text{rev } (\text{app } as \ ys) \ (\text{cons } a \ \text{nil}) \approx \text{app } (\text{rev } ys \ \text{nil}) \ (\text{rev } as \ (\text{cons } a \ \text{nil})) ; \tau_3) \\ (zs ; zs \approx \text{app } zs \ \text{nil} ; \text{app } zs \ \text{nil}) \end{array} \right\}$$

This is a generalization because for every gsc substitution γ that respects **true** we can choose the substitution $\delta = [zs := \text{rev } \gamma(ys) \ \text{nil}] \cup [\gamma(x) \mid x \in \mathcal{V} \setminus \{zs\}]$ and have both $(\text{rev } ys \ \text{nil})\gamma = zs\delta$ and $(\text{app } (\text{rev } ys \ \text{nil}) \ \text{nil})\gamma = (\text{app } zs \ \text{nil})\delta$, as well as $\zeta_4\gamma \geq zs\delta = (\text{rev } ys \ \text{nil})\gamma$ and $\tau_4\gamma \geq (\text{app } zs \ \text{nil})\delta$ because $(\zeta_4 ; \text{rev } ys \ \text{nil} \approx \text{app } (\text{rev } ys \ \text{nil}) \ \text{nil} ; \tau_4)$ is a bounded equation context. Moreover, the ordering requirements of the derivation rule, $zs \geq zs \ [\text{true}]$ and $\text{app } zs \ \text{nil} \geq \text{app } zs \ \text{nil} \ [\text{true}]$, clearly hold.

Next, we use (Induct) to deduce $(\mathcal{E}_5, \mathcal{H}_2) \vdash (\mathcal{E}_5, \mathcal{H}_5)$, with

$$\mathcal{H}_5 = \left\{ \begin{array}{l} \text{rev } (\text{app } xs \ ys) \ \text{nil} \approx \text{app } (\text{rev } ys \ \text{nil}) \ (\text{rev } xs \ \text{nil}) \\ zs \approx \text{app } zs \ \text{nil} \end{array} \right\}$$

We use (Case) on the second equation, to split it up into separate cases for **nil** and **cons**:

$$\mathcal{E}_5 = \left\{ \begin{array}{l} (\zeta_3 ; \text{rev } (\text{app } as \ ys) \ (\text{cons } a \ \text{nil}) \approx \text{app } (\text{rev } ys \ \text{nil}) \ (\text{rev } as \ (\text{cons } a \ \text{nil})) ; \tau_3) \\ (\text{nil} ; \text{nil} \approx \text{app } \text{nil} \ \text{nil} ; \text{app } \text{nil} \ \text{nil}) \\ (\text{cons } x \ xs ; \text{cons } x \ xs \approx \text{app } (\text{cons } x \ xs) \ \text{nil} ; \text{app } (\text{cons } x \ xs) \ \text{nil}) \end{array} \right\}$$

The second of these equations is quickly dispatched through applications of (Simplify), ending in a (Delete) step. For the third equation, we use (Simplify) to obtain $(\mathcal{E}_6, \mathcal{H}_5)$:

$$\mathcal{E}_6 = \left\{ \begin{array}{l} (\zeta_3 ; \text{rev } (\text{app } as \ ys) \ (\text{cons } a \ \text{nil}) \approx \text{app } (\text{rev } ys \ \text{nil}) \ (\text{rev } as \ (\text{cons } a \ \text{nil})) ; \tau_3) \\ (\text{cons } x \ xs ; \text{cons } x \ xs \approx \text{cons } x \ (\text{app } xs \ \text{nil}) ; \text{app } (\text{cons } x \ xs) \ \text{nil}) \end{array} \right\}$$

(Semi-constructor) Due to extending rule arity to constructors, we can now present the (Semi-constructor) rule more conveniently, compared to HORI:

$$\frac{(\mathcal{E} \uplus \{(\zeta ; f \ s_1 \cdots s_n \approx f \ t_1 \cdots t_n ; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \{(\zeta ; s_i \approx t_i ; \tau) [\psi] \mid 1 \leq i \leq n\}, \mathcal{H})} \quad n > 0 \text{ and } (f \in \mathcal{V} \text{ or } n < ar(f))$$

If f is a constructor then $ar(f) = \infty$ when, so certainly $n < ar(f)$ holds.

We apply (Semi-constructor) on the second equation in $\mathcal{E}_6$ to obtain $(\mathcal{E}_7, \mathcal{H}_5)$, with

$$\mathcal{E}_7 = \left\{ \begin{array}{l} (\zeta_3 ; \text{rev } (\text{app } as \ ys) \ (\text{cons } a \ \text{nil}) \approx \text{app } (\text{rev } ys \ \text{nil}) \ (\text{rev } as \ (\text{cons } a \ \text{nil})) ; \tau_3) \\ (\text{cons } x \ xs ; x \approx x ; \text{app } (\text{cons } x \ xs) \ \text{nil}) \\ (\text{cons } x \ xs ; xs \approx \text{app } xs \ \text{nil} ; \text{app } (\text{cons } x \ xs) \ \text{nil}) \end{array} \right\}$$

The second equation context is immediately deleted. We furthermore note that the third is an instance of the induction hypothesis $zs \approx \text{app } zs \text{ nil} \in \mathcal{H}$, so we can dispatch it using ($\mathcal{H}$ -Delete), provided the ordering requirements are satisfied. If we choose a bounding pair with $\triangleright \subseteq \triangleright$, we certainly have $(\text{cons } x \text{ } xs)\gamma \triangleright xs\gamma$ for all γ , which suffices.

So we then obtain

$$\mathcal{E}_8 = \{(\varsigma_3 ; \text{rev } (\text{app } as \text{ } ys) (\text{cons } a \text{ nil}) \approx \text{app } (\text{rev } ys \text{ nil}) (\text{rev } as (\text{cons } a \text{ nil})) ; \tau_3)\}$$

(Postulate) Rather than generalizing a given equation in $\mathcal{E}$, it might be preferable to introduce a lemma from scratch. For $\mathcal{E}_8$ it is useful to introduce lemmas to express the interaction between rev and app . For this purpose, we introduce the (Postulate) rule, which was not present in HORI.

$$\frac{(\mathcal{E}, \mathcal{H})}{(\mathcal{E} \cup \{(\bullet ; s \approx t ; \bullet) [\psi]\}, \mathcal{H})}$$

The combination of (Postulate) with (Induct) provides a way to introduce induction hypotheses in $\mathcal{H}$, which are not necessarily generalizations of existing equations in $\mathcal{E}$.

In $\mathcal{E}_8$, we use two successive applications of (Postulate) to introduce equations $\text{rev } xs \text{ } ys \approx \text{app } (\text{rev } xs \text{ nil}) \text{ } ys$ and $\text{app } (\text{app } xs \text{ } ys) \text{ } zs \approx \text{app } xs (\text{app } ys \text{ } zs)$. This yields a proof state $(\mathcal{E}_9, \mathcal{H}_5)$, with $\mathcal{E}_9 =$

$$\left\{ \begin{array}{l} (\mathbf{E}_0) \quad (\varsigma_3 ; \text{rev } (\text{app } as \text{ } ys) (\text{cons } a \text{ nil}) \approx \text{app } (\text{rev } ys \text{ nil}) (\text{rev } as (\text{cons } a \text{ nil})) ; \tau_3) \\ (\mathbf{E}_1) \quad (\bullet ; \text{rev } xs \text{ } ys \approx \text{app } (\text{rev } xs \text{ nil}) \text{ } ys ; \bullet) \\ (\mathbf{E}_2) \quad (\bullet ; \text{app } (\text{app } xs \text{ } ys) \text{ } zs \approx \text{app } xs (\text{app } ys \text{ } zs) ; \bullet) \end{array} \right\}$$

We use (Induct) to obtain $(\mathcal{E}_9, \mathcal{H}_6)$, where $\mathcal{H}_6$ contains $(\mathbf{E}_2)$ as induction hypothesis:

$$\mathcal{H}_6 = \mathcal{H}_5 \cup \{\text{app } (\text{app } xs \text{ } ys) \text{ } zs \approx \text{app } xs (\text{app } ys \text{ } zs)\}$$

After this, the equation context $(\mathbf{E}_2)$ is easily removed from $\mathcal{E}_9$, using (Case) to deal with the two separate instances of xs , as follows

- nil case: can be removed with (Delete) after some applications of (Simplify) with $\mathcal{R}$.
- cons case: can be removed with followed by ($\mathcal{H}$ -Delete) after some applications of (Simplify) with $\mathcal{R}$ and application of (Semi-constructor).

This does not create new ordering requirements since we had already set $\triangleright \subseteq \triangleright$. We are left with a proof state $(\{(\mathbf{E}_0), (\mathbf{E}_1)\}, \mathcal{H}_6)$.

We then continue with (Induct), to add the equation $(\mathbf{E}_1)$ to $\mathcal{H}_6$, after which we remove the equation from $\{(\mathbf{E}_0), (\mathbf{E}_1)\}$. This is less easy as it was to remove $(\mathbf{E}_2)$ and

requires the use of (Hypothesis) from both (E_1) and (E_2) before we can eventually apply $(\mathcal{H}\text{-Delete})$ with (E_1) . We refer to the Cora proof file for the specific details.

We are then finally ready to use the hypotheses from (E_1) and (E_2) to eliminate (E_0) . We only informally sketch the remainder of the proof, starting with the equation in (E_0)

$$(G_0) \quad \text{rev}(\text{app } as \ ys) (\text{cons } a \ \text{nil}) \approx \text{app}(\text{rev } ys \ \text{nil}) (\text{rev } as (\text{cons } a \ \text{nil}))$$

- ▷ Applying the (E_1) induction hypothesis $\text{rev } xs \ ys \approx \text{app}(\text{rev } xs \ \text{nil}) \ ys$ with (Hypothesis), we can rewrite the left-hand side of (G_0) to obtain

$$(G_1) \quad \text{app}(\text{rev}(\text{app } as \ ys) \ \text{nil}) (\text{cons } a \ \text{nil}) \approx \text{app}(\text{rev } ys \ \text{nil}) (\text{rev } as (\text{cons } a \ \text{nil}))$$

- ▷ Using the induction hypothesis $\text{rev}(\text{app } xs \ ys) \ \text{nil} \approx \text{app}(\text{rev } ys \ \text{nil}) (\text{rev } xs \ \text{nil})$ with (Hypothesis), we can rewrite the left-hand side of (G_1) to obtain

$$(G_2) \quad \text{app}(\text{app}(\text{rev } ys \ \text{nil}) (\text{rev } as \ \text{nil})) (\text{cons } a \ \text{nil}) \approx \text{app}(\text{rev } ys \ \text{nil}) (\text{rev } as (\text{cons } a \ \text{nil}))$$

- ▷ Using the (E_2) induction hypothesis $\text{app}(\text{app } xs \ ys) \ zs \approx \text{app } xs (\text{app } ys \ zs)$ with (Hypothesis), we can rewrite the left-hand side of (G_2) to obtain

$$(G_3) \quad \text{app}(\text{rev } ys \ \text{nil}) (\text{app}(\text{rev } as \ \text{nil}) (\text{cons } a \ \text{nil})) \approx \text{app}(\text{rev } ys \ \text{nil}) (\text{rev } as (\text{cons } a \ \text{nil}))$$

- ▷ Note that (G_3) is equivalently written as

$$C[\text{app}(\text{rev } as \ \text{nil}) (\text{cons } a \ \text{nil})] \approx C[\text{rev } as (\text{cons } a \ \text{nil})] \text{ with } C[\square] = \text{app}(\text{rev } ys \ \text{nil}) \ \square$$

Therefore, we can apply $(\mathcal{H}\text{-Delete})$ with the (E_1) induction hypothesis $\text{rev } xs \ ys \approx \text{app}(\text{rev } xs \ \text{nil}) \ ys$ to delete (G_3) .

The proof requires termination requirements which can be satisfied, e.g., by choosing $>$ to be the lexicographic path ordering with $\text{rev} > \text{app} > \text{cons} > \text{nil}$.

5.2.3 Completeness

Thus far, we have used rewriting induction to prove inductive theorems. However, we can also use it to derive that some equations are *not* inductive theorems.

Example 5.2.14. Consider $\Sigma_{\text{terms}} = \{G :: (\text{int} \rightarrow \text{int}) \rightarrow \text{int} \rightarrow \text{int} \rightarrow \text{int}, H :: (\text{int} \rightarrow \text{int}) \rightarrow \text{int} \rightarrow \text{int} \rightarrow \text{int} \rightarrow \text{int}\}$ with rules

$$\begin{array}{ll} G \ f \ n \ x \rightarrow G \ f \ (n-1) \ (f \ x) & [n > 0] \\ G \ f \ n \ x \rightarrow x & [n \leq 0] \\ H \ f \ n \ m \ x \rightarrow H \ f \ (n-1) \ m \ (f \ x) & [n > 0] \\ H \ f \ n \ m \ x \rightarrow H \ f \ (m-1) \ n \ (f \ x) & [m > 0] \\ H \ f \ n \ m \ x \rightarrow x & [n \leq 0 \wedge m \leq 0] \end{array}$$

Intuitively, G computes the function $(f, n, x) \mapsto f^n(x)$ for all $n \geq 0$, and H computes the function $(f, n, m, x) \mapsto f^{(n+m)}(x)$ for all $n, m \geq 0$. In particular, we have an inductive theorem $G \ f \ k \ x \approx H \ f \ n \ m \ x \ [k = n + m \wedge n \geq 0 \wedge m \geq 0]$. The condition $n, m \geq 0$ is really necessary: the equation $G \ f \ k \ x \approx H \ f \ n \ m \ x \ [k = n + m]$ is *not* an inductive theorem. For example, there does not even exist a single ground substitution γ such that $(G \ ([+] \ 1) \ (-1) \ x)\gamma \leftrightarrow_{\mathcal{R}}^* (H \ ([+] \ 1) \ 1 \ (-2) \ x)\gamma$.

To prove non-equivalences, we will extend RI with a new deduction rule (Disprove). However, we must take some care: it is entirely possible, in the course of a RI proof, to end up with unsound equations even if the original equations are all inductive theorems. For example, $RD \ [+ \ n \ 0 \ a \approx RD \ [+ \ n \ (-1) \ a]$ is an inductive theorem, but with (Generalize) we can obtain $RD \ [+ \ n \ i \ a \approx RD \ [+ \ n \ i' \ a \ [i' = i - 1]$ from which we derive a contradiction. Hence, we must carefully consider the derivation path.

Definition 5.2.15 (Completeness Property). We say a deduction rule has the Completeness Property if, whenever $\mathcal{R}$ is a weakly normalizing, quasi-reductive, ground confluent LCSTRS, if we can deduce $(\mathcal{E}, \mathcal{H}) \vdash (\mathcal{E}', \mathcal{H}')$ by this rule, then

$$\leftrightarrow_{\mathcal{E}} \cup \leftrightarrow_{\mathcal{H}} \subseteq \leftrightarrow_{\mathcal{R}}^* \text{ on ground terms} \quad \implies \quad \leftrightarrow_{\mathcal{E}'} \cup \leftrightarrow_{\mathcal{H}'} \subseteq \leftrightarrow_{\mathcal{R}}^* \text{ on ground terms}$$

Here, $\leftrightarrow_{\mathcal{E}}$ is the relation $\leftrightarrow_{\mathcal{A}}$ where $\mathcal{A} = \{s \approx t \ [\psi] \mid (\varsigma ; s \approx t ; \tau) \ [\psi] \in \mathcal{E}\}$.

Lemma 5.2.16. The derivation rules (Simplify), (Case), (Delete), (Induct), (Hypothesis), (Alter) and ($\mathcal{H}$ -Delete) all have the Completeness Property, as does the limitation of (Semi-constructor) to cases with $f \in \Sigma$.

Hence, all deduction rules of Figure 5.1 other than (Generalize), (Postulate) and the general case of (Semi-constructor), are complete.

Proof. Suppose that $\mathcal{R}$ is weakly normalizing, quasi-reductive and ground confluent, and that $\leftrightarrow_{\mathcal{E}} \cup \leftrightarrow_{\mathcal{H}} \subseteq \leftrightarrow_{\mathcal{R}}^*$ on ground terms. Write $\mathcal{A} = \{s \approx t \ [\psi] \mid (\varsigma ; s \approx t ; \tau) \ [\psi] \in \mathcal{E}\}$.

For any $s \approx t \ [\psi] \in \mathcal{H}'$ we have either $s \approx t \ [\psi] \in \mathcal{H}$ or, if (Induct) was used, $s \approx t \ [\psi] \in \mathcal{A}$. Either way, a step by this equation can also be done by $\leftrightarrow_{\mathcal{H}} \cup \leftrightarrow_{\mathcal{E}}$, so indeed $\leftrightarrow_{\mathcal{H}'} \subseteq \leftrightarrow_{\mathcal{R}}^*$ on ground terms. To show that $\leftrightarrow_{\mathcal{E}'} \subseteq \leftrightarrow_{\mathcal{R}}^*$ on ground terms, let γ be a ground substitution respecting some equation context in $\mathcal{E}'$. Note that we only need to consider equation contexts in $\mathcal{E}' \setminus \mathcal{E}$ because for an equation context $(\varsigma ; s \approx t ; \tau) \ [\psi] \in \mathcal{E}$ we can immediately conclude that $s\gamma \leftrightarrow_{\mathcal{R}}^* t\gamma$ by assumption. We consider every deduction rule in the lemma.

(*Simplify*), (*Hypothesis*). Then $\mathcal{E}' \setminus \mathcal{E} = \{(\varsigma ; C[r\delta] \approx t ; \tau) \ [\psi]\}$ while $\mathcal{A}$ contains $C[\ell\delta] \approx t \ [\psi]$ for either some $\ell \rightarrow r \ [\varphi] \in \mathcal{R} \cup \mathcal{R}_{calc}$ or some $\ell \simeq r \ [\varphi] \in \mathcal{H}$, and δ such that $\psi \models^{\delta} \varphi$. Since γ respects ψ we have $C[\ell\delta]\gamma \leftrightarrow_{\mathcal{R}}^* t\gamma$ by assumption on

$\leftrightarrow_{\mathcal{A}}$. Furthermore, in either case we have $C[r\delta]\gamma \leftrightarrow_{\mathcal{R}}^* C[\ell\delta]\gamma$, whether because $\leftarrow_{\mathcal{R}}$ is included in $\leftrightarrow_{\mathcal{R}}^*$ or by assumption on $\leftrightarrow_{\mathcal{H}}$.

(Case). Then the only elements of $\mathcal{E}' \setminus \mathcal{E}$ have a form $(\varsigma\delta ; s\delta \approx t\delta ; \tau\delta) [\psi\delta \wedge \varphi]$, where $s \approx t [\psi] \in \mathcal{A}$. As γ respects $\psi\delta \wedge \varphi$, it respects $\psi\delta$. Therefore, $\sigma = \delta\gamma$ respects ψ , and is a ground substitution; $\text{dom}(\sigma) \supseteq \text{Var}(s, t)$. Hence, $s\delta\gamma = s\sigma \leftrightarrow_{\mathcal{R}}^* t\sigma = t\delta\gamma$, as desired.

(Delete), ($\mathcal{H}$ -Delete). We have $\mathcal{E}' \setminus \mathcal{E} = \emptyset$ so there is nothing to prove.

(Induct). For all $(\varsigma ; s \approx t ; \tau) [\psi] \in \mathcal{E}'$ we have $s \approx t [\psi] \in \mathcal{A}$.

(Alter). Then $\mathcal{E}' \setminus \mathcal{E} = \{(\varsigma' ; s' \approx t' ; \tau') [\psi']\}$ and there is some $(\varsigma ; s \approx t ; \tau) [\psi] \in \mathcal{E}$ which generalizes $(\varsigma' ; s' \approx t' ; \tau') [\psi']$. Since $\mathcal{R}$ is weakly normalizing and quasi-reductive, $\gamma^\downarrow = [x := \gamma(x)\downarrow_{\mathcal{R}} \mid x \in \text{dom}(\gamma)]$ is a well-defined, gsc substitution that respects $(\varsigma' ; s' \approx t' ; \tau') [\psi']$. By definition of generalization, there is a substitution σ that respects $(\varsigma ; s \approx t ; \tau) [\psi]$ with $s\sigma = s'\gamma^\downarrow$ and $t\sigma = t'\gamma^\downarrow$. Hence, $s'\gamma \rightarrow_{\mathcal{R}}^* s'\gamma^\downarrow = s\sigma \leftrightarrow_{\mathcal{R}}^* t\sigma = t'\gamma^\downarrow \leftarrow_{\mathcal{R}}^* t'\gamma$.

(Semi-constructor) (with $f \in \Sigma$). Then $\mathcal{E}' \setminus \mathcal{E} = \{(\varsigma ; s_i \approx t_i ; \tau) [\psi] \mid 1 \leq i \leq n\}$ for some $(\varsigma ; f s_1 \cdots s_n \approx f t_1 \cdots t_n ; \tau) [\psi] \in \mathcal{E}$, with $n < \text{ar}(f)$. By assumption, $(f s_1 \cdots s_n)\gamma \leftrightarrow_{\mathcal{R}}^* (f t_1 \cdots t_n)\gamma$. Now, note that $\mathcal{R}$ is ground confluent. So in fact there is some u such that both $(f s_1 \cdots s_n)\gamma \rightarrow_{\mathcal{R}}^* u$ and $(f t_1 \cdots t_n)\gamma \rightarrow_{\mathcal{R}}^* u$. Since $n < \text{ar}(f)$, necessarily u has the form $f u_1 \cdots u_n$ with both $s_i\gamma \rightarrow_{\mathcal{R}}^* u_i$ and $t_i\gamma \rightarrow_{\mathcal{R}}^* u_i$ for $1 \leq j \leq n$. $\square$

With the notion of a complete deduction step, we can define the states in a derivation path that are suitable for the new deduction rule that we wish to introduce:

Definition 5.2.17 (Complete proof state). Let $\mathcal{P}_1 \vdash^* \mathcal{P}_n$ be some RI deduction sequence and $1 \leq i \leq n$. We say that $\mathcal{P}_i = (\mathcal{E}_i, \mathcal{H}_i)$ is complete if one of the following holds

- (a). $i = 1$
- (b). $\mathcal{P}_{i-1}$ is complete and $\mathcal{P}_{i-1} \vdash \mathcal{P}_i$ by a deduction rule with the Completeness Property
- (c). there is a complete proof state $\mathcal{P}_j$ with $j < i$ and $\mathcal{E}_i \subseteq \mathcal{E}_j$.

Cases (a) and (b) ensure that deductions remain complete as long as they only use steps with the Completeness Property. With (c) we can restore completeness, once we have removed all equations originating from deduction steps that cause loss of completeness.

Note that “complete proof state” is *only* defined within the setting of a derivation sequence: it is not a standalone property of a proof state. This makes it difficult to formulate a deduction rule for proving non-theorems: so far, the deduction rules only consider proof states, not the history of how they were obtained. This is resolved by defining a *new* proof system, still based on the original one in Figure 5.1, but using an extended form of proof states: tuples $(\mathcal{E}, \mathcal{H}, F)$ where F is either C , or $l(Q)$ for some set

(StepCC)	$\frac{(\mathcal{E}, \mathcal{H}, \mathcal{C})}{(\mathcal{E}', \mathcal{H}', \mathcal{C})} \quad (\mathcal{E}, \mathcal{H}) \vdash (\mathcal{E}', \mathcal{H}') \text{ by a step with the Completeness Property}$
(StepCI)	$\frac{(\mathcal{E}, \mathcal{H}, \mathcal{C})}{(\mathcal{E}', \mathcal{H}', \mathcal{I}(\mathcal{E}))} \quad (\mathcal{E}, \mathcal{H}) \vdash (\mathcal{E}', \mathcal{H}') \text{ by any derivation step}$
(StepII)	$\frac{(\mathcal{E}, \mathcal{H}, \mathcal{I}(\mathcal{Q}))}{(\mathcal{E}', \mathcal{H}', \mathcal{I}(\mathcal{Q}))} \quad (\mathcal{E}, \mathcal{H}) \vdash (\mathcal{E}', \mathcal{H}') \text{ by any derivation step}$
(Complete)	$\frac{(\mathcal{E}, \mathcal{H}, \mathcal{I}(\mathcal{Q}))}{(\mathcal{E}, \mathcal{H}, \mathcal{C})} \quad \mathcal{E} \subseteq \mathcal{Q}$
(Disprove)	$\frac{(\mathcal{E} \cup \{(\varsigma; s \approx t; \tau) [\psi]\}, \mathcal{H}, \mathcal{C})}{\perp} \quad s \approx t [\psi] \text{ is contradictory (see Definition 5.2.18)}$

Fig. 5.2 Derivation rules for Bounded RI with completeness tracking and (Disprove).

of equations $\mathcal{Q}$. Here, $\mathcal{Q}$ should be understood as the last set $\mathcal{E}_i$ that precedes $(\mathcal{E}, \mathcal{H})$ in the derivation sequence for which $(\mathcal{E}_i, \mathcal{H}_i)$ was known to be complete.

The new proof system is given in Figure 5.2. Here, (StepCC) encodes Definition 5.2.17(b): a complete state is reduced to another complete state by a deduction rule with the completeness property. The rule (StepCI) is used to handle the remaining deduction rules, whose application causes the next proof state to become incomplete. This derivation rule is formulated in a general way, allowing it to be applied with any deduction rule, to ensure that we can also make progress when it is unknown whether a given deduction rule has the Completeness Property or not. However, in practice (StepCC) should always take precedence. The rule (StepII) allows progress when the proof state is incomplete: we pass on $\mathcal{Q}$ unmodified, as this encodes the last set of equations for which the proof state was complete, which does not change. The (Complete) rule encodes Definition 5.2.17(c): we can recover completeness of a proof state if its set $\mathcal{E}$ is included in the equations of a prior complete proof state.

Finally, (Disprove) is the reason for defining this new proof system, yielding a way to prove non-theorems. This derivation rule relies on the following definition:

Definition 5.2.18 (Contradictory equation). An equation $s \approx t [\psi]$ is contradictory if there exists a ground substitution δ that respects ψ , such that one of the following holds:

- (1). there exist $f, g \in \Sigma$ with $f \neq g$, as well as $s'_1, \dots, s'_n, t'_1, \dots, t'_m$ such that $s\delta = f s'_1 \cdots s'_n$ and $t\delta = g t'_1 \cdots t'_m$ and $n < ar(f)$ and $m < ar(g)$
- (2). $s, t \in T(\Sigma_{theory}, \mathcal{V})$ and $\llbracket (\psi \wedge s \neq t)\delta \rrbracket = \top$

Note that if s and t only have base-type variables, then the existence of a suitable substitution δ for case (2) corresponds exactly to satisfiability of $\psi \wedge (s \neq t)$. However, when s or t contains higher-order variables, then we first have to choose an instantiation for every such variable before we can check for satisfiability. For example, if we have an equation $x \approx f x [\psi]$ with $x :: \text{int}$ and $f :: \text{int} \rightarrow \text{int}$ then $\psi \wedge (x \neq f x)$ is not a constraint and we first have to instantiate f as a theory term (e.g. $f := [+]\ 1$) before we can establish satisfiability.

We now formulate a deduction rule for proving non-theorems:

(Disprove) A complete proof state with a contradictory equation yields $\perp$.

$$\frac{(\mathcal{E} \cup \{(\zeta; s \approx t; \tau) [\psi]\}, \mathcal{H}, \mathcal{C})}{\perp} \quad s \approx t [\psi] \text{ is contradictory (see Definition 5.2.18)}$$

We can see that if (Disprove) applies then $s \approx t [\psi]$ cannot be an inductive theorem. This is expressed by the following theorem (which we will prove in Section 5.3.5).

Theorem 5.2.19 (Completeness of RI). Let $\mathcal{L}$ be a weakly normalizing, quasi-reductive, ground confluent LCSTRS and let $\mathcal{E}$ be a set of equations. If $(\mathcal{E}, \emptyset, \mathcal{C}) \vdash^* \perp$ then there is an equation in $\mathcal{E}$ which is not an inductive theorem.

We assume weak normalization in Definition 5.2.15 and Theorem 5.2.19 because, without this assumption, (Alter) would not have the Completeness Property. If the derivation $(\mathcal{E}, \emptyset, \mathcal{C}) \vdash^* \perp$ does not use the (Alter) step, then the conclusion of Theorem 5.2.19 can still be obtained without having to require weak normalization.

Example 5.2.20. Consider the LCSTRS from Example 5.2.14. The system terminates and in Example 5.6.12 we prove ground confluence. Let $\mathcal{E}_1 = \{(\bullet; G f k x \approx H f n m x; \bullet) [k = n + m]\}$. We derive $(\mathcal{E}_1, \emptyset, \mathcal{C}) \vdash^* \perp$. First, apply (StepCC) with (Case):

$$\mathcal{E}_2 = \left\{ \begin{array}{ll} \text{(E1)} & (\bullet; G f k x \approx H f n m x; \bullet) \quad [k < 0 \wedge n > 0 \wedge k = n + m] \\ \text{(E2)} & (\bullet; G f k x \approx H f n m x; \bullet) \quad [k \geq 0 \wedge k = n + m] \\ \text{(E3)} & (\bullet; G f k x \approx H f n m x; \bullet) \quad [n \leq 0 \wedge k = n + m] \end{array} \right\}$$

From (E1) we will derive $\perp$, so we can forget about (E2) and (E3). After applying some (Simplify) steps, together with (Alter) – all applied through (StepCC) – we get

$$\mathcal{E}_3 \supseteq \{(\bullet; x \approx H f n' m (f x); \bullet) [n' = n - 1 \wedge k < 0 \wedge n > 0 \wedge k = n + m]\}$$

A new application of (Case), followed by a number of (Simplify)-steps with $\mathcal{R}$ yields $(\mathcal{E}_4, \emptyset, \mathcal{C})$ with $\mathcal{E}_4 \supseteq \{(\mathbf{E1})', (\mathbf{E2})'\}$, where

$$(\mathbf{E1})' \quad (\bullet; x \approx f \ x; \bullet) \quad [n' \leq 0 \wedge n' = n - 1 \wedge k < 0 \wedge n > 0 \wedge k = n + m]$$

$$(\mathbf{E2})' \quad (\bullet; x \approx H \ f \ n' \ m \ (f \ x); \bullet) \quad [n' > 0 \wedge n' = n - 1 \wedge k < 0 \wedge n > 0 \wedge k = n + m]$$

Now, $(\mathbf{E1})'$ is contradictory by Definition 5.2.18.(2), so with (Disprove) we obtain $\perp$. Therefore, $G \ f \ k \ x \approx H \ f \ n \ m \ x \ [k = n + m]$ is not an inductive theorem.

Although we now technically have two different proof systems (one with completeness flag and one without), in practice we can use both at the same time:

Lemma 5.2.21. For all $F: (\mathcal{E}, \mathcal{H}) \vdash^* (\mathcal{E}, \mathcal{H}) \Leftrightarrow$ there exists F' with $(\mathcal{E}, \mathcal{H}, F) \vdash^* (\mathcal{E}, \mathcal{H}, F')$.

Proof. $\Leftarrow$: by simply erasing the flags from each step (and omitting (Complete) steps).

$\Rightarrow$: by adding flags using (StepCI) and (StepII). $\square$

Hence, once we prove Theorem 5.2.5 we immediately have the parallel result that every equation in $\mathcal{A}$ is an inductive theorem if $(\mathcal{E}, \emptyset, \mathcal{C}) \vdash^* (\emptyset, \mathcal{H}', F)$ for some F .

5.3 Proofs

In this section, we prove soundness and completeness theorems of Section 5.2, in a form that allows for easily adding new deduction rules. In addition, we supply the results mentioned in Remark 5.2.12, which allow us to limit interest to bounded equation contexts when using Bounded Rewriting Induction. This section is organized as follows

- In Section 5.3.1 we show that the rules from Figure 5.1 restrict to a proof system on bounded equation contexts. This property is called *Preserving Bounds*.
- In Section 5.3.2 we introduce some necessary prerequisites on multisets.
- In Section 5.3.3 we introduce $\mathcal{R}$ -bounded ground convertibility; a restriction of $\mathcal{R}$ -ground convertibility which limits intermediate terms in the conversion using the bounding pair $(>, \geq)$. We use this notion to express the *Base Soundness property*, and prove that every deduction rule except (Induct) satisfies this property. This allows us to obtain a more transparent and modular proof procedure towards proving Theorem 5.2.5.
- In Section 5.3.4 we accumulate the results of the preceding subsections to finally conclude Theorem 5.2.5. In particular, Theorem 5.3.18 shows that every deduction rule that satisfies Preserving Bounds and the Base Soundness property is sound.
- In Section 5.3.5 we prove Theorem 5.2.19 by induction on the length of a derivation sequence that ends in $\perp$. The base case is covered by Proposition 5.3.22, after which the full proof is obtained using Lemma 5.2.16.

5.3.1 Bounded equation contexts

In the following, we assume given a fixed bounding pair $(>, \geq)$ on $T(\Sigma, \mathcal{V})$. In this subsection we will show that Bounded Rewriting Induction can always be limited to bounded equation contexts.

Definition 5.3.1 (Preserving Bounds). We say that a deduction rule $(\mathcal{E}, \mathcal{H}) \vdash (\mathcal{E}', \mathcal{H}')$

- ▷ *preserves bounds* if, whenever all equation contexts in $\mathcal{E}$ are bounded, then also all equation contexts in $\mathcal{E}'$ are bounded.
- ▷ *preserves strong bounds* if, whenever all equation contexts in $\mathcal{E}$ are *strongly* bounded, then also all equation contexts in $\mathcal{E}'$ are *strongly* bounded.

Lemma 5.3.2. All derivation rules in Figure 5.1 preserve bounds.

Proof. All elements of $\mathcal{E}' \cap \mathcal{E}$ are bounded equation contexts by assumption; as for the rest, consider the rule by which $(\mathcal{E}, \mathcal{H}) \vdash (\mathcal{E}', \mathcal{H}')$:

(Simplify) $\mathcal{E}' \setminus \mathcal{E} = \{(\varsigma ; C[r\delta] \approx t ; \tau) [\psi]\}$ with $(\varsigma ; C[\ell\delta] \approx t ; \tau) [\psi] \in \mathcal{E}$. Since the original equation context is bounded, we have $\tau \geq t [\psi]$ and $\varsigma \geq C[\ell\delta] [\psi]$. Since $\rightarrow_{\mathcal{R}} \subseteq \geq$, we have $C[\ell\delta] \geq C[r\delta] [\psi]$. Then $\varsigma \geq C[r\delta] [\psi]$ by transitivity of $\geq$.

(Case) $\mathcal{E}$ contains a bounded equation context $(\varsigma ; s \approx t ; \tau) [\psi]$, and $\mathcal{E}' \setminus \mathcal{E}$ contains equation contexts $(\varsigma\delta ; s\delta \approx t\delta ; \tau\delta) [\psi\delta \wedge \varphi]$. To see that these are bounded as well, consider a ground substitution σ that respects $\psi\delta \wedge \varphi$. This substitution certainly respects $\psi\delta$, so the composition $\delta\sigma$ respects ψ . Since by assumption $\varsigma \geq s [\psi]$ and $\tau \geq t [\psi]$ hold, we have $\varsigma(\delta\sigma) \geq s(\delta\sigma)$ and $\tau(\delta\sigma) \geq t(\delta\sigma)$.

(Delete), (H-Delete) $\mathcal{E}' \setminus \mathcal{E} = \emptyset$

(Semi-constructor) $\mathcal{E}' \setminus \mathcal{E}$ contains only equation contexts $(\varsigma ; s_i \approx t_i ; \tau) [\psi]$ where $\mathcal{E}$ contains a bounded equation context $(\varsigma ; f s_1 \cdots s_n \approx f t_1 \cdots t_n ; \tau) [\psi]$; since $\varsigma\gamma \geq (f s_1 \cdots s_n)\gamma$ implies $\varsigma\gamma \geq s_i\gamma$ (as $\geq$ is transitive and includes $\triangleright$), and we similarly have $\tau\gamma \geq t_i\gamma$, all elements of $\mathcal{E}' \setminus \mathcal{E}$ are bounded.

(Induct) $\mathcal{E}' \setminus \mathcal{E} = \{(s ; s \approx t ; t) [\psi]\}$ and $s \geq s [\psi]$ and $t \geq t [\psi]$ hold by reflexivity of $\geq$.

(Hypothesis) $\mathcal{E}' \setminus \mathcal{E} = \{(\varsigma ; C[r] \approx t ; \tau) [\psi]\}$ where $\varsigma \geq C[r] [\psi]$ by definition of the deduction rule, and $\tau \geq t [\psi]$ because the input equation is bounded.

(Generalize)/(Alter) Boundedness of the result is a requirement of the deduction rule.

(Postulate) The only fresh equation context is $(\bullet ; s \approx t ; \bullet) [\psi]$, and $\bullet > s\gamma, t\gamma$ holds. $\square$

By almost similar reasoning we can see that the derivation rules – when applied in certain ways and with a suitably restricted bounding pair – also preserve *strong* bounds.

Lemma 5.3.3. Suppose $s > t$ whenever $s \rightarrow_{\mathcal{R}} t$ or $s \triangleright t$. Let $s \geq^! t [\psi]$ denote that either $s = t$ or $s > t [\psi]$. Consider the variation of the proof system of Figure 5.1 where any use of $\geq$ is replaced by $\geq^!$. All derivation rules in this variant preserve strong bounds.

Note that $\geq$ is only used in the definition of (Hypothesis), (Generalize) and (Alter), so all other derivation rules preserve strong bounds without modification.

Proof. The proof is a straightforward adaptation of the proof of Lemma 5.3.2, observing that if $a \geq^! b [\varphi]$ and $b \geq^! c [\varphi]$ then $a \geq^! c [\varphi]$ immediately follows.

- For (Simplify), we use that certainly $C[\ell\delta] > C[r\delta] [\psi]$ because $\rightarrow_{\mathcal{R}} \subseteq >$.
- For (Case), we observe that $a \geq^! b [\varphi]$ clearly implies $a\delta \geq^! b\delta [\varphi\delta]$ as well.
- For (Semi-constructor), we observe that $f s_1 \cdots s_n \geq^! s_i [\psi]$ for any ψ because $\triangleright$ is included in $>$ (as $\geq$ is the reflexive closure of $>$ and $a \triangleright b$ implies $a \neq b$).
- All other cases are immediately obvious. □

We also observe that, as stated in Remark 5.2.12, the typical use cases of (Alter) automatically preserve both boundedness and strong boundedness:

Lemma 5.3.4. Suppose $(\varsigma ; s \approx t ; \tau) [\psi]$ is a bounded equation context, and $(\varsigma' ; s' \approx t' ; \tau') [\psi']$ alters it by either method (I) or (II). Then also $(\varsigma' ; s' \approx t' ; \tau') [\psi']$ is a bounded equation context.

Furthermore, if $(\varsigma ; s \approx t ; \tau) [\psi]$ is strongly bounded, then so is $(\varsigma' ; s' \approx t' ; \tau') [\psi']$.

Proof. First consider case (I): in this case $(\varsigma' ; s' \approx t' ; \tau') [\psi']$ is $(\varsigma ; s \approx t ; \tau) [\varphi]$, where:

- $\{x_1, \dots, x_n\} = \text{Var}(\psi) \setminus \text{Var}(\varsigma, s, t, \tau)$ and $\{y_1, \dots, y_m\} = \text{Var}(\varphi) \setminus \text{Var}(\varsigma, s, t, \tau)$;
- $\{z_1, \dots, z_k\} = \text{Var}(\varphi, \psi) \cap \text{Var}(\varsigma, s, t, \tau)$;
- $(\exists x_1 \dots x_n. \psi) \Leftrightarrow (\exists y_1 \dots y_m. \varphi)$ is logically valid; that is, for all appropriately typed values $v_1, \dots, v_k$,

$$\exists u_1, \dots, u_n \in \text{Val} \llbracket \psi[z_1 := v_1, \dots, z_k := v_k, x_1 := u_1, \dots, x_n := u_n] \rrbracket = \top$$

$$\iff$$

$$\exists w_1, \dots, w_m \in \text{Val} \llbracket \varphi[z_1 := v_1, \dots, z_k := v_k, y_1 := w_1, \dots, y_m := w_m] \rrbracket = \top$$

Now, let γ be a ground substitution that respects φ . From “respects”, we know that $\llbracket \varphi\gamma \rrbracket = \top$, so for the fixed values $v_1 = \gamma(z_1), \dots, v_k = \gamma(z_k)$, if we choose $w_1 = \gamma(y_1), \dots, w_m = \gamma(y_m)$ then the second part of the “if and only if” above is satisfied, so we can find values $u_1, \dots, u_n$ that satisfy the first part. Now, let δ be the substitution that maps each x_i to u_i and all other variables a to $\gamma(a)$. Then indeed

$\llbracket \psi \delta \rrbracket = \llbracket \psi[z_1 := v_1, \dots, z_k := v_k, x_1 := u_1, \dots, x_n := u_n] \rrbracket = \top$, so δ respects ψ , and since δ is identical to γ on all variables in ς, s, t, τ , we have $\varsigma\gamma = \varsigma\delta \geq s\delta = s\gamma$ and $\tau\gamma = \tau\delta \geq t\delta = t\gamma$ as required.

As for strong boundedness: if $\varsigma = s$ for the original, then clearly the same holds for the altered context. Otherwise $\varsigma > s$ $[\psi]$ so, as we saw above, for all γ we find δ such that $\varsigma\gamma = \varsigma\delta > s\delta = s\gamma$. The same holds for τ and t .

Now consider case (II): then $(\varsigma'; s' \approx t'; \tau') [\psi']$ has the form $(\varsigma\gamma; s\gamma \approx t\gamma; \tau\gamma) [\psi]$ where $\gamma = [x_1 := v_1, \dots, x_n := v_n]$ such that $\psi \implies x_1 = v_1 \wedge x_n = v_n$ is valid; that is, for every substitution δ that respects ψ we have $\delta(x_i) = v_i\delta$. Hence, $\varsigma\gamma\delta = \varsigma\delta \geq s\delta = s\gamma\delta$ and $\tau\gamma\delta = \tau\delta \geq t\delta = t\gamma\delta$ as required, and similar for strong boundedness. $\square$

5.3.2 Multiset orderings

The proofs in the following subsections make substantial use of the multiset extension of an ordering pair $(>, \geq)$ on $\mathcal{A} = T(\Sigma, \mathcal{V})$. We particularly consider multisets containing exactly 2 elements, which allows for a much simpler representation than the common definitions in the literature (where multisets may have infinite size). Here, we present our version of the multiset extension of an ordering pair, as follows:

For an ordering pair $(>, \geq)$ on $\mathcal{A}$, a *size-2 multiset* over $\mathcal{A}$ is an unordered pair $\{\{a, b\}\}$ with $a, b \in \mathcal{A}$ (it is allowed to have $a = b$). We say that:

- $\{\{a_1, a_2\}\} = \{\{b_1, b_2\}\}$ if either $a_1 = b_1$ and $a_2 = b_2$, or $a_1 = b_2$ and $a_2 = b_1$
- $\{\{a_1, a_2\}\} \geq_{mul} \{\{b_1, b_2\}\}$ if one of the following holds:
 - there is some $i \in \{1, 2\}$ such that both $a_i > b_1$ and $a_i > b_2$
 - either $a_1 \geq b_1$ and $a_2 \geq b_2$, or $a_1 \geq b_2$ and $a_2 \geq b_1$
- $\{\{a_1, a_2\}\} >_{mul} \{\{b_1, b_2\}\}$ if one of the following holds:
 - there is some $i \in \{1, 2\}$ such that both $a_i > b_1$ and $a_i > b_2$
 - there exist $i, j \in \{1, 2\}$ such that $a_i > b_j$ and $a_{3-i} \geq b_{3-j}$

Note that $\{\{a_1, a_2\}\} >_{mul} \{\{b_1, b_2\}\}$ implies $\{\{a_1, a_2\}\} \geq_{mul} \{\{b_1, b_2\}\}$, because $a_i > b_j$ implies $a_i \geq b_j$. Hence, we easily see that the multiset extension $(>_{mul}, \geq_{mul})$ is itself an ordering pair on size-2 multisets over $\mathcal{A}$.

5.3.3 Bounded ground convertibility

In principle, to prove Theorem 5.2.5, it suffices to show $\mathcal{R}$ -ground convertibility for every equation in $\mathcal{E}$. However, as we will see, bounded rewriting induction actually proves a stronger property, called *bounded* ground convertibility.

Definition 5.3.5 ($\mathcal{R}/\mathcal{H}/\mathcal{E}$ -Bounded convertibility). Let $\mathcal{R}$ be a set of rewrite rules, $(>, \geq)$ a bounding pair, $\mathcal{H}$ a set of equations, and $\mathcal{E}$ a set of equation contexts. For $a, b \in T(\Sigma, \mathcal{V}) \cup \{\bullet\}$ and $u, v \in T(\Sigma, \mathcal{V})$ we define $u \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}]{\{a, b\}} v$ if one of the following holds:

- (A) $u = C[\ell\sigma]$ and $v = C[r\sigma]$ for some $\ell \rightarrow r[\varphi]$ or $r \rightarrow \ell[\varphi]$ in $\mathcal{R}$, such that:
 - ▷ σ is a ground substitution that respects the rule
 - ▷ $a \geq \ell\sigma$ or $b \geq \ell\sigma$
 - ▷ $a \geq r\sigma$ or $b \geq r\sigma$
- (B) $u = C[\ell\sigma]$ and $v = C[r\sigma]$ for some $\ell \approx r[\varphi]$ or $r \approx \ell[\varphi]$ in $\mathcal{H}$ such that:
 - ▷ σ is a ground substitution that respects the equation, and
 - ▷ $\{a, b\} >_{\text{mul}} \{\ell\sigma, r\sigma\}$
- (C) $u = C[\ell\sigma]$ and $v = C[r\sigma]$ for some $(d; \ell \approx r; e)[\varphi] \in \mathcal{E}$ such that:
 - ▷ σ is a ground substitution that respects the equation context, and
 - ▷ $\{a, b\} \geq_{\text{mul}} \{d\sigma, e\sigma\}$

We write $\xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}]{\{a, b\}^*}$ for the reflexive, transitive closure of $\xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}]{\{a, b\}}$.

Definition 5.3.6 ($\mathcal{R}/\mathcal{H}/\mathcal{E}$ -Bounded ground convertibility). Assume given a fixed LCSTRS with rules $\mathcal{R}$ and a bounding pair $(>, \geq)$. Let $\mathcal{H}$ be a set of equations and $\mathcal{E}$ be a set of equation contexts. An equation context $(\varsigma; s \approx t; \tau)[\psi]$ is $\mathcal{R}/\mathcal{H}/\mathcal{E}$ -bounded ground convertible if $s\gamma \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}]{\{\varsigma\gamma, \tau\gamma\}^*} t\gamma$ for every ground substitution γ that respects ψ .

We will prove (in Theorem 5.3.21) that, if $(\mathcal{E}, \emptyset) \vdash^* (\emptyset, \mathcal{H})$, then every equation context in $\mathcal{E}$ is $\mathcal{R}/\mathcal{H}/\mathcal{E}$ -bounded ground convertible. This result not only implies Theorem 5.2.5, but will also be invaluable to use rewriting induction for ground confluence (Corollary 5.6.11).

To start with the proof, we observe that both relations $\xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}]{\{a, b\}}$ and $\xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}]{\{a, b\}^*}$ are symmetric. We will often use this property without explicitly stating it.

We also immediately see that they are preserved under contexts:

Lemma 5.3.7. Let D be a context.

- (1) If $u \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}]{\{a, b\}} v$, then $D[u] \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}]{\{a, b\}} D[v]$.
- (2) If $u \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}]{\{a, b\}^*} v$, then $D[u] \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}]{\{a, b\}^*} D[v]$.

Proof. Obvious by definition of the relation: in each place where we use a context $C[]$ we can also use the context $D[C[]]$ instead, without affecting the ordering requirements. $\square$

Similarly, the relations are preserved under an increase of the bounding terms a, b :

Lemma 5.3.8. Suppose $\{c, d\} \geq_{mul} \{a, b\}$.

- (1) If $u \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}]{\{a, b\}} v$, then $u \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}]{\{c, d\}} v$. (2) If $u \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}]{\{a, b\}^*} v$, then $u \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}]{\{c, d\}^*} v$.

Proof. The second statement follows immediately from the first. For the first, recall that $(\succ_{mul}, \geq_{mul})$ is an ordering pair; in particular, $X \geq_{mul} Y \geq_{mul} Z$ implies $X \geq_{mul} Z$, and $X \geq_{mul} Y \succ_{mul} Z$ implies $X \succ_{mul} Z$. Hence, both cases (B) and (C) are preserved. If $u \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}]{\{a, b\}} v$ by case (A), then there exists some $e \in \{a, b\}$ such that $e \geq \ell\sigma$. By definition of $\geq_{mul}$, there must be some $f \in \{c, d\}$ such that $f \succ e$ or $f \geq e$; so either way, $f \geq e \geq \ell\sigma$ and $f \geq \ell\sigma$ holds by transitivity. Similarly, some element $g \in \{c, d\}$ exists with $g \geq r\sigma$. $\square$

In the proof of Theorem 5.3.21, we will study derivation sequences $(\mathcal{E}_1, \emptyset) \vdash (\mathcal{E}_2, \mathcal{H}_2) \vdash \dots \vdash (\emptyset, \mathcal{H}_N)$. We show that for every $1 \leq i \leq N$: every equation context in $\mathcal{E}_i$ is $\mathcal{R}/\mathcal{H}_i/\emptyset$ -bounded ground convertible. (Since $\mathcal{H}_1 = \emptyset$ this in particular implies the desired result that $s\gamma \xrightarrow[\mathcal{R}; \emptyset]{\{\varsigma\gamma, \tau\gamma\}^*} t\gamma$ for every equation context $(\varsigma; s \approx t; \tau) [\psi] \in \mathcal{E}_1$ and every ground substitution γ that respects ψ .) The most difficult deduction rule in these proofs is (Induct). To present the proofs in a comprehensible way, we will therefore first prove a property for every deduction rule other than (Induct).

Definition 5.3.9 (Base Soundness Property). We say a deduction rule has the Base Soundness Property if, whenever we can deduce $(\mathcal{E}, \mathcal{H}) \vdash (\mathcal{E}', \mathcal{H}')$ by this rule and all equation contexts in $\mathcal{E}$ are bounded, then $\mathcal{H}' = \mathcal{H}$ and for all $(\varsigma; s \approx t; \tau) [\psi] \in \mathcal{E} \setminus \mathcal{E}'$, and for every ground substitution γ that respects ψ , we have: $s\gamma \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}']{\{\varsigma\gamma, \tau\gamma\}^*} t\gamma$.

Now we can see that all rules other than (Induct) indeed have this property.

Lemma 5.3.10. (Simplify) has the Base Soundness Property.

Proof. We can write $\mathcal{E} = \mathcal{E}_0 \uplus \{(\varsigma; C[\ell\delta] \approx t; \tau) [\psi]\}$ and $\mathcal{E}' = \mathcal{E}_0 \cup \{(\varsigma; C[r\delta] \approx t; \tau) [\psi]\}$ for some $\ell \rightarrow r [\varphi] \in \mathcal{R} \cup \mathcal{R}_{calc}$ such that $\psi \models^\delta \varphi$.

Let γ be a ground substitution that respects ψ . Then we have:

▷ $C[\ell\delta]\gamma \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}']{\{\varsigma\gamma, \tau\gamma\}} C[r\delta]\gamma$ by (A). This holds because:

- From $\psi \models^\delta \varphi$ we know that γ respects $\varphi\delta$. Phrased differently, the composed substitution $\sigma := \delta\gamma$ respects φ . Therefore, $s\gamma = C[\ell\delta]\gamma = C\gamma[\ell\sigma] \rightarrow_{\mathcal{R}} C\gamma[r\sigma] = C[r\delta]\gamma$.
- Since $\varsigma \geq s[\psi]$, and $\geq$ includes ▷, we have $\varsigma\gamma \geq s\gamma \geq \ell\sigma$.
- Since $\rightarrow_{\mathcal{R}}$ is included in $\geq$, it follows that $\varsigma\gamma \geq \ell\sigma \geq r\sigma$ as well.

▷ $C[r\delta]\gamma \xleftrightarrow[\mathcal{R};\mathcal{H};\mathcal{E}']^{\{\varsigma\gamma, \tau\gamma\}} t\gamma$ by (C), because $\{\varsigma\gamma, \tau\gamma\} \succeq_{mul} \{\varsigma\gamma, \tau\gamma\}$ by reflexivity.

Hence, putting these steps together we have $s\gamma \xleftrightarrow[\mathcal{R};\mathcal{H};\mathcal{E}']^{\{\varsigma\gamma, \tau\gamma\} *} t\gamma$. $\square$

Lemma 5.3.11. If $\mathcal{R}$ is weakly normalising and quasi-reductive, then (Case) has the Base Soundness Property.

Proof. We write $\mathcal{E} = \mathcal{E}_0 \uplus \{(\varsigma; s \approx t; \tau) [\psi]\}$ and $\mathcal{E}' = \mathcal{E}_0 \cup \{(\varsigma\delta; s\delta \approx t\delta; \tau\delta) [\psi\delta \wedge \varphi] \mid (\delta, \varphi) \in C\}$ for a cover set C of $s \approx t [\psi]$. Let γ be a ground substitution that respects $(\varsigma; s \approx t; \tau) [\psi]$. Since $\mathcal{R}$ is weakly normalising and quasi-reductive, every term has a normal form, and every ground normal form must be a semi-constructor term. Let $\gamma^\downarrow$ be the substitution that maps each x to a normal form of $\gamma(x)$. If $x \in \text{Var}(\psi)$ then $\gamma(x) \in \mathcal{V}al$, hence $\gamma^\downarrow(x) = \gamma(x)$. So also $\gamma^\downarrow$ respects ψ . Thus we see:

- ▷ $s\gamma \xleftrightarrow[\mathcal{R};\mathcal{H};\mathcal{E}']^{\{\varsigma\gamma, \tau\gamma\} *} s\gamma^\downarrow$ by 0 or more steps using (A), because:
 - By assumption $(\varsigma; s \approx t; \tau) [\psi]$ is a bounded equation context, so $\varsigma\gamma \geq s\gamma$.
 - Writing $s\gamma = u_0 \rightarrow_{\mathcal{R}} u_1 \rightarrow_{\mathcal{R}} \dots \rightarrow_{\mathcal{R}} u_n$, the inclusion of $\rightarrow_{\mathcal{R}}$ in $\geq$ (along with transitivity of $\geq$) ensures that $\varsigma\gamma \geq u_i$ for all i .
 - For $1 \leq i \leq n$, writing $u_{i-1} = C[\ell\delta]$ and $u_i = C[r\delta]$, we observe that $u_{i-1} \geq \ell\delta$ and $u_i \geq r\delta$ because $\triangleright$ is included in $\geq$.
- ▷ $s\gamma^\downarrow \xleftrightarrow[\mathcal{R};\mathcal{H};\mathcal{E}']^{\{\varsigma\gamma, \tau\gamma\}} t\gamma^\downarrow$ by (C):
 - By definition of cover set, there exist a pair $(\delta, \varphi) \in C$ and a substitution ϵ such that ϵ respects $\psi\delta \wedge \varphi$ and $\gamma^\downarrow = \delta\epsilon$ on all variables in $\text{dom}(\gamma^\downarrow)$. In particular, this means that $s\gamma^\downarrow = s\delta\epsilon$ and $t\gamma^\downarrow = t\delta\epsilon$ and $\varsigma\gamma^\downarrow = \varsigma\delta\epsilon$ and $\tau\gamma^\downarrow = \tau\delta\epsilon$.
 - Hence, we can use the step with $(\varsigma\delta; s\delta \approx t\delta; \tau\delta) [\psi\delta \wedge \varphi] \in \mathcal{E}'$.
 - We have $\{\varsigma\gamma, \tau\gamma\} \succeq_{mul} \{\varsigma\gamma^\downarrow, \tau\gamma^\downarrow\} = \{\varsigma\delta\epsilon, \tau\delta\epsilon\}$ because by definition of a bounding pair (in particular $\rightarrow_{\mathcal{R}} \subseteq \geq$) we have $\varsigma\gamma \geq \varsigma\gamma^\downarrow$ and $\tau\gamma \geq \tau\gamma^\downarrow$.
- ▷ $t\gamma^\downarrow \xleftrightarrow[\mathcal{R};\mathcal{H};\mathcal{E}']^{\{\varsigma\gamma, \tau\gamma\} *} t\gamma$ by (A) by the same reasoning as the reduction from $s\gamma$ to $s\gamma^\downarrow$. $\square$

Lemma 5.3.12. (Delete) has the Base Soundness Property.

Proof. We can write $\mathcal{E} = \mathcal{E}' \uplus \{(\varsigma; s \approx t; \tau) [\psi]\}$, where either $s = t$ or ψ is unsatisfiable. Let γ be a ground substitution that respects ψ . The existence of γ implies that ψ is satisfiable, so necessarily $s = t$. But then clearly $s\gamma \xleftrightarrow[\mathcal{R};\mathcal{H};\mathcal{E}']^{\{\varsigma\gamma, \tau\gamma\} *} t\gamma$, using 0 steps. $\square$

Lemma 5.3.13. (Semi-constructor) has the Base Soundness Property.

Proof. We can write $\mathcal{E} = \mathcal{E}_0 \uplus \{(\varsigma ; f s_1 \cdots s_n \approx f t_1 \cdots t_n ; \tau) [\psi]\}$ and $\mathcal{E}' = \mathcal{E}_0 \cup \{(\varsigma ; s_i \approx t_i ; \tau) [\psi] \mid 1 \leq i \leq n\}$ for f a variable or function symbol. Let γ be some ground substitution that respects the equation context. Since $\{\{\varsigma\gamma, \tau\gamma\}\} \succeq_{mul} \{\{\varsigma\gamma, \tau\gamma\}\}$, we use (C) to derive

$$(f s_1 s_2 \cdots s_n)\gamma \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}']{\{\varsigma\gamma, \tau\gamma\}} (f t_1 s_2 \cdots s_n)\gamma \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}']{\{\varsigma\gamma, \tau\gamma\}} \cdots \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}']{\{\varsigma\gamma, \tau\gamma\}} (f t_1 t_2 \cdots t_n)\gamma$$

using a context $C[\Box] = f t_1 \cdots t_{i-1} \Box s_{i+1} \cdots s_n$ for the i^{th} step. Together, this exactly gives $(f s_1 s_2 \cdots s_n)\gamma \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}']{\{\varsigma\gamma, \tau\gamma\} *} (f t_1 t_2 \cdots t_n)\gamma$ as required. $\square$

Lemma 5.3.14. ($\mathcal{H}$ -Delete) has the Base Soundness Property.

Proof. We can write $\mathcal{E} = \mathcal{E}' \uplus \{(\varsigma ; C[\ell\delta] \approx C[r\delta] ; \tau) [\psi]\}$ for some $\ell \simeq r [\varphi]$ in $\mathcal{H}$ such that $\psi \models^\delta \varphi$, and (a) $\varsigma > \ell\delta [\psi]$ or (b) $\tau > r\delta [\psi]$. Let γ be a ground substitution that respects the equation context. Then (a') $\varsigma\gamma > \ell\delta\gamma$ or (b') $\tau\gamma > r\delta\gamma$. Since equation contexts in $\mathcal{E}$ are bounded, and $\triangleright \subseteq \succeq$, we know that (c) $\varsigma\gamma \geq C[\ell\delta]\gamma \geq \ell\delta\gamma$ and (d) $\tau\gamma \geq C[r\delta]\gamma \geq r\delta\gamma$. Using (a') with (d), or (b') with (c), we have $\{\{\varsigma\gamma, \tau\gamma\}\} \succ_{mul} \{\{\ell\delta\gamma, r\delta\gamma\}\}$. So, $C[\ell\delta]\gamma = C\gamma[\ell\delta\gamma] \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}']{\{\varsigma\gamma, \tau\gamma\}} C\gamma[r\delta\gamma] = C[r\delta]\gamma$ by (B). $\square$

Lemma 5.3.15. (Postulate) has the Base Soundness Property.

Proof. Since $\mathcal{H} = \mathcal{H}'$ and $\mathcal{E} \setminus \mathcal{E}' = \emptyset$, there is nothing to prove. $\square$

Lemma 5.3.16. (Hypothesis) has the Base Soundness Property.

Proof. We can write $\mathcal{E} = \mathcal{E}_0 \uplus \{(\varsigma ; C[\ell\delta] \simeq t ; \tau) [\psi]\}$ and $\mathcal{E}' = \mathcal{E}_0 \cup \{(\varsigma ; C[r\delta] \simeq t ; \tau) [\psi]\}$ for some $\ell \simeq r [\varphi]$ in $\mathcal{H}$ such that $\psi \models^\delta \varphi$. Let γ be a ground substitution that respects the equation context. Then from $\varsigma > \ell\delta [\psi]$ and $\varsigma > r\delta [\psi]$, we have $\varsigma\gamma > \ell\delta\gamma$ and $\varsigma\gamma > r\delta\gamma$, which together implies $\{\{\varsigma\gamma, \tau\gamma\}\} \succ_{mul} \{\{\ell\delta\gamma, r\delta\gamma\}\}$. Hence, $C[\ell\delta]\gamma = C\gamma[\ell\delta\gamma] \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}']{\{\varsigma\gamma, \tau\gamma\}} C\gamma[r\delta\gamma] = C[r\delta]\gamma$ by (B). Moreover, we clearly have $C[r\delta]\gamma \xrightarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}']{\{\varsigma\gamma, \tau\gamma\}} t\gamma$ by (C). $\square$

Lemma 5.3.17. If $\mathcal{R}$ is weakly normalising and quasi-reductive, then (Generalize) and (Alter) have the Base Soundness Property.

Proof. In both cases: $\mathcal{E} = \mathcal{E}_0 \uplus \{(\varsigma ; s \approx t ; \tau) [\psi]\}$ and $\mathcal{E}' = \mathcal{E}_0 \cup \{(\varsigma' ; s' \approx t' ; \tau') [\psi']\}$, where the latter equation context generalizes the former. Let γ be a ground substitution that respects $(\varsigma ; s \approx t ; \tau) [\psi]$. As $\mathcal{R}$ is weakly normalizing we can define $\gamma^\downarrow$ as a substitution that maps each x to a normal form of $\gamma(x)$; since $\mathcal{R}$ is quasi-reductive $\gamma^\downarrow$ is a gsc substitution. Hence, by definition of (Generalize), there is a substitution δ that respects ψ' such that (a) $\varsigma\gamma^\downarrow \geq \varsigma'\delta$, (b) $\tau\gamma^\downarrow \geq \tau'\delta$, (c) $s\gamma^\downarrow = s'\delta$ and (d) $t\gamma^\downarrow = t'\delta$.

- ▷ Both $s\gamma \xleftrightarrow[\mathcal{R};\mathcal{H};\mathcal{E}']^{\{\varsigma\gamma, \tau\gamma\} *} s\gamma^\downarrow$ and $t\gamma^\downarrow \xleftrightarrow[\mathcal{R};\mathcal{H};\mathcal{E}']^{\{\varsigma\gamma, \tau\gamma\} *} t\gamma$ hold by (A): the facts that $\varsigma\gamma \geq s\gamma$, $\tau\gamma \geq t\gamma$, and that both $\rightarrow_{\mathcal{R}}$ and $\triangleright$ are included in $\geq$ ensure the ordering requirements.
- ▷ Since $\geq \rightarrow_{\mathcal{R}}$, we have $\varsigma\gamma \geq \varsigma\gamma^\downarrow$ and $\tau\gamma \geq \tau\gamma^\downarrow$, so from (a) and (b) together we obtain $\{\{\varsigma\gamma, \tau\gamma\}\} \geq_{mul} \{\{\varsigma'\delta, \tau'\delta\}\}$. Hence, $s\gamma^\downarrow = s'\delta \xleftrightarrow[\mathcal{R};\mathcal{H};\mathcal{E}']^{\{\varsigma\gamma, \tau\gamma\} *} t'\delta = t\gamma^\downarrow$ by (C). $\square$

5.3.4 Soundness of Bounded Rewriting Induction

Instead of proving Theorem 5.2.5 directly, we will prove a more general property:

Theorem 5.3.18. Let $\mathcal{L}$ be a weakly normalizing, quasi-reductive LCSTRS; let $(>, \geq)$ be a bounding pair, and $\mathcal{E}_1$ a set of bounded equation contexts. Suppose $(\mathcal{E}_1, \emptyset) \vdash (\mathcal{E}_2, \mathcal{H}_2) \vdash \dots \vdash (\mathcal{E}_N, \mathcal{H}_N)$, where each $\vdash$ is derived using either (Induct) or a derivation rule that has both the Preserving Bounds and Base Soundness Properties. Let $1 \leq i \leq N$. Then every equation context in $\mathcal{E}_i$ is $\mathcal{R}/\mathcal{H}_i/\mathcal{E}_N$ -bounded ground convertible (where $\mathcal{H}_1 = \emptyset$).

Note that this result implies Theorem 5.2.5, but with the stronger property of bounded ground convertibility rather than merely ground convertibility. This proof strategy, moreover, allows us to extend the system of Figure 5.1 with additional deduction rules and maintain soundness, so long as the new deduction rules satisfy two properties that are often easy to prove (Preserving Bounds and the Base Soundness Property).

Proving Theorem 5.3.18 roughly proceeds by observing that for $i = N$, every equation context in $\mathcal{E}_i$ is $\mathcal{R}/\mathcal{H}_i/\mathcal{E}_N$ -bounded ground convertible, and then showing that if this property holds for $\mathcal{E}_i$ with $i > 0$, then it also holds for $\mathcal{E}_{i-1}$. We first show the inductive step for the deduction rules where $\mathcal{H}$ is unchanged; that is, all rules other than (Induct).

To avoid excessive whitespace in the following three proofs, we will use $\leftrightarrow_{\mathcal{R};\mathcal{H};\mathcal{E}}^{\{a,b\}}$ as an alternative notation for $\xleftrightarrow[\mathcal{R};\mathcal{H};\mathcal{E}}^{\{a,b\}}$ and $\leftrightarrow_{\mathcal{R};\mathcal{H};\mathcal{E}}^{\{a,b\} *} \text{ for } \xleftrightarrow[\mathcal{R};\mathcal{H};\mathcal{E}}^{\{a,b\} *}.$

Lemma 5.3.19. Consider a deduction step $(\mathcal{E}, \mathcal{H}) \vdash (\mathcal{E}', \mathcal{H})$, i.e. leaving $\mathcal{H}$ unchanged. Suppose that

- (a) for every $(\varsigma ; s \approx t ; \tau) [\psi] \in \mathcal{E} \setminus \mathcal{E}'$ and ground substitution γ that respects it we have $s\gamma \xleftrightarrow[\mathcal{R};\mathcal{H};\mathcal{E}']^{\{\varsigma\gamma, \tau\gamma\} *} t\gamma$; and
- (b) for every $(\varsigma ; s \approx t ; \tau) [\psi] \in \mathcal{E}'$ and ground substitution γ that respects it we have $s\gamma \xleftrightarrow[\mathcal{R};\mathcal{H};\mathcal{E}_N]^{\{\varsigma\gamma, \tau\gamma\} *} t\gamma$.

Then for every $(\varsigma ; s \approx t ; \tau) [\psi] \in \mathcal{E}$ and ground substitution γ that respects this equation context we have $s\gamma \xleftrightarrow[\mathcal{R};\mathcal{H};\mathcal{E}_N]^{\{\varsigma\gamma, \tau\gamma\} *} t\gamma$.

Proof. Let $(\varsigma ; s \approx t ; \tau) [\psi] \in \mathcal{E}$ and γ a ground substitution that respects this equation context ψ . To start, if $(\varsigma ; s \approx t ; \tau) [\psi] \in \mathcal{E}'$ then we are done by assumption (b). Otherwise, $(\varsigma ; s \approx t ; \tau) [\psi] \in \mathcal{E} \setminus \mathcal{E}'$. We will show that $s\gamma \leftrightarrow_{\mathcal{R};\mathcal{H};\mathcal{E}_N}^{\{\varsigma\gamma, \tau\gamma\}^*} t\gamma$.

By assumption (a), we may write $s\gamma = c_0, t\gamma = c_k$, and there are $c_1, \dots, c_{k-1}$ such that $c_i \leftrightarrow_{\mathcal{R};\mathcal{H};\mathcal{E}'}^{\{\varsigma\gamma, \tau\gamma\}} c_{i+1}$ for $0 \leq i < k$. We are done if we can show that in fact $c_i \leftrightarrow_{\mathcal{R};\mathcal{H};\mathcal{E}_N}^{\{\varsigma\gamma, \tau\gamma\}^*} c_{i+1}$ for all such i . To see that this is the case, consider the definition of $\leftrightarrow_{\mathcal{R};\mathcal{H};\mathcal{E}'}^{\{\varsigma\gamma, \tau\gamma\}}$, that is, Definition 5.3.5. If the step from c_i to c_{i+1} is by (A) or (B), then this is also a $\leftrightarrow_{\mathcal{R};\mathcal{H};\mathcal{E}_N}^{\{\varsigma\gamma, \tau\gamma\}}$ step, so we are immediately done. Otherwise, it is by (C), so $c_i = C[s'\sigma]$ and $c_{i+1} = C[t'\sigma]$ for some $(\varsigma' ; s' \approx t' ; \tau') [\varphi] \in \mathcal{E}'$ such that σ respects φ and $\{\{\varsigma\gamma, \tau\gamma\}\} \geq_{mul} \{\{\varsigma'\sigma, \tau'\sigma\}\}$.

By assumption (b), we have that $s'\sigma \leftrightarrow_{\mathcal{R};\mathcal{H};\mathcal{E}_N}^{\{\varsigma'\sigma, \tau'\sigma\}^*} t'\sigma$. By Lemma 5.3.7, this also implies that $c_i = C[s'\sigma] \leftrightarrow_{\mathcal{R};\mathcal{H};\mathcal{E}_N}^{\{\varsigma'\sigma, \tau'\sigma\}^*} C[t'\sigma] = c_{i+1}$. Then, from $\{\{\varsigma\gamma, \tau\gamma\}\} \geq_{mul} \{\{\varsigma'\sigma, \tau'\sigma\}\}$ we obtain $c_i \leftrightarrow_{\mathcal{R};\mathcal{H};\mathcal{E}_N}^{\{\varsigma\gamma, \tau\gamma\}^*} c_{i+1}$ by Lemma 5.3.8, as required. $\square$

Next, we obtain a lemma that will be very useful for when the (Induct) rule is used.

Lemma 5.3.20. Suppose $\varsigma \geq s [\psi]$ and $\tau \geq t [\psi]$, let $\mathcal{H}$ be a set of equations and $\mathcal{H}' = \mathcal{H} \cup \{s \approx t [\psi]\}$. Let $s\gamma \leftrightarrow_{\mathcal{R};\mathcal{H}';\mathcal{E}_N}^{\{s\gamma, t\gamma\}^*} t\gamma$ hold for all ground substitutions γ that respect $(\varsigma ; s \approx t ; \tau) [\psi]$.

Then for all ground substitutions γ that respect $(\varsigma ; s \approx t ; \tau) [\psi]$: $s\gamma \leftrightarrow_{\mathcal{R};\mathcal{H};\mathcal{E}_N}^{\{\varsigma\gamma, \tau\gamma\}^*} t\gamma$.
(So using $\mathcal{H}$ instead of $\mathcal{H}'$, and $\varsigma\gamma, \tau\gamma$ instead of $s\gamma, t\gamma$.)

Proof. Let γ be a ground substitution that respects $(\varsigma ; s \approx t ; \tau) [\psi]$. We prove the lemma by induction on the multiset $\{s\gamma, t\gamma\}$, ordered with $>_{mul}$.

Since $\leftrightarrow_{\mathcal{R};\mathcal{H}';\mathcal{E}_N}^{\{s\gamma, t\gamma\}^*}$ denotes a transitive closure, we have a sequence $c_0, c_1, \dots, c_k$ such that $c_0 = s\gamma$ and $c_k = t\gamma$ and for $0 \leq i < k$: $c_i \leftrightarrow_{\mathcal{R};\mathcal{H}';\mathcal{E}_N}^{\{s\gamma, t\gamma\}} c_{i+1}$. Each step from c_i to c_{i+1} that uses $\leftrightarrow_{\mathcal{R}}$ or $\leftrightarrow_{\mathcal{H}}$ or $\leftrightarrow_{\mathcal{E}_N}$ is also a $\leftrightarrow_{\mathcal{R};\mathcal{H};\mathcal{E}_N}^{\{s\gamma, t\gamma\}}$ step, and therefore a $\leftrightarrow_{\mathcal{R};\mathcal{H};\mathcal{E}_N}^{\{\varsigma\gamma, \tau\gamma\}}$ step by Lemma 5.3.8. So, suppose the step from c_i to c_{i+1} uses $\mathcal{H}' \setminus \mathcal{H}$; that is, there exists a ground substitution δ that respects $s \approx t [\psi]$, such that $\{\{s\gamma, t\gamma\}\} >_{mul} \{\{s\delta, t\delta\}\}$, and such that either $c_i = C[s\delta]$ and $c_{i+1} = C[t\delta]$, or $c_i = C[t\delta]$ and $c_{i+1} = C[s\delta]$. Either way, by the induction hypothesis, $s\delta \leftrightarrow_{\mathcal{R};\mathcal{H};\mathcal{E}_N}^{\{s\delta, t\delta\}^*} t\delta$, and therefore by Lemmas 5.3.7 and 5.3.8, $c_i \leftrightarrow_{\mathcal{R};\mathcal{H};\mathcal{E}_N}^{\{\varsigma\gamma, \tau\gamma\}^*} c_{i+1}$. $\square$

With this, we have all the preparations needed to prove the primary soundness result for Bounded RI. We recall the theorem:

Theorem 5.3.18. Let $\mathcal{L}$ be a weakly normalizing, quasi-reductive LCSTRS; let $(>, \geq)$ be a bounding pair, and $\mathcal{E}_1$ a set of bounded equation contexts. Suppose $(\mathcal{E}_1, \emptyset) \vdash (\mathcal{E}_2, \mathcal{H}_2) \vdash \dots \vdash (\mathcal{E}_N, \mathcal{H}_N)$, where each $\vdash$ is derived using either (Induct) or a derivation rule that has both the Preserving Bounds and Base Soundness Properties. Let $1 \leq i \leq N$. Then every equation context in $\mathcal{E}_i$ is $\mathcal{R}/\mathcal{H}_i/\mathcal{E}_N$ -bounded ground convertible (where $\mathcal{H}_1 = \emptyset$).

Proof. By definition of Preserving Bounds (Definition 5.3.1), and the fact that (Induct) also preserves bounds (Lemma 5.3.2), we obtain by induction on i that (**) each $\mathcal{E}_i$ contains only bounded equation contexts. Now, we will prove the required result by induction on $N - i$. So assume given $1 \leq i \leq N$, let $(\varsigma ; s \approx t ; \tau) [\psi] \in \mathcal{E}_i$, and let γ be a ground substitution that respects ψ . We must show $s\gamma \leftrightarrow_{\mathcal{R};\mathcal{H}_i;\mathcal{E}_N}^{\{\varsigma\gamma,\tau\gamma\}^*} t\gamma$.

If $N - i = 0$ then $s\gamma \leftrightarrow_{\mathcal{R};\mathcal{H}_i;\mathcal{E}_N}^{\{\varsigma\gamma,\tau\gamma\}^*} t\gamma$ by a single step using (C). If $N - i > 0$, so $1 \leq i < N$, consider the deduction step $(\mathcal{E}_i, \mathcal{H}_i) \vdash (\mathcal{E}_{i+1}, \mathcal{H}_{i+1})$. If this step has the Base Soundness Property, then by (**), property (a) of Lemma 5.3.19 is satisfied, while property (b) is satisfied by the induction hypothesis. Thus, $s\gamma \leftrightarrow_{\mathcal{R};\mathcal{H}_i;\mathcal{E}_N}^{\{\varsigma\gamma,\tau\gamma\}^*} t\gamma$ by Lemma 5.3.19.

The only remaining case is that the deduction step was performed by (Induct). That is, $\mathcal{E}_i = \mathcal{E} \uplus \{(a ; u \approx v ; b) [\varphi]\}$ and $\mathcal{E}_{i+1} = \mathcal{E} \cup \{(u ; u \approx v ; v) [\varphi]\}$ and $\mathcal{H}_{i+1} = \mathcal{H}_i \cup \{u \approx v [\varphi]\}$. We distinguish two cases:

- ▷ $(\varsigma ; s \approx t ; \tau) [\psi] \in \mathcal{E}$. Then this equation context is in $\mathcal{E}_{i+1}$, so by induction hypothesis we have $s\gamma \leftrightarrow_{\mathcal{R};\mathcal{H}_{i+1};\mathcal{E}_N}^{\{\varsigma\gamma,\tau\gamma\}^*} t\gamma$. That is, there exist $c_0, \dots, c_k$ with $k \geq 0$ such that $s\gamma = c_0$, $t\gamma = c_k$ and for $0 \leq j < k$ we have $c_j \leftrightarrow_{\mathcal{R};\mathcal{H}_{i+1};\mathcal{E}_N}^{\{\varsigma\gamma,\tau\gamma\}^*} c_{j+1}$. It suffices if each $c_j \leftrightarrow_{\mathcal{R};\mathcal{H}_i;\mathcal{E}_N}^{\{\varsigma\gamma,\tau\gamma\}^*} c_{j+1}$ as well. So fix $j \in \{0, \dots, k-1\}$, and consider how the step from c_j to c_{j+1} is produced. If the step from c_j to c_{j+1} is by (A) (so using $\leftrightarrow_{\mathcal{R}}$) or (C) (so using $\leftrightarrow_{\mathcal{E}_N}$), or if it is by (B) with an equation in $\mathcal{H}_i$, then we immediately have $c_j \leftrightarrow_{\mathcal{R};\mathcal{H}_i;\mathcal{E}_N}^{\{\varsigma\gamma,\tau\gamma\}^*} c_{j+1}$ as well.

The only alternative is a step by (B) using $\mathcal{H}_{i+1} \setminus \mathcal{H}_i = \{u \approx v [\varphi]\}$. That is, there exists a ground substitution σ that respects φ and for which $\{\{\varsigma\gamma, \tau\gamma\}\} \succ_{mul} \{u\sigma, v\sigma\}$, such that either $c_j = C[u\sigma]$ and $c_{j+1} = C[v\sigma]$, or $c_j = C[v\sigma]$ and $c_{j+1} = C[u\sigma]$. Now, observe that for all ground substitutions δ that respect φ we have $u\delta \leftrightarrow_{\mathcal{R};\mathcal{H}_{i+1};\mathcal{E}_N}^{\{u\delta,v\delta\}^*} v\delta$ by the induction hypothesis, because $(u ; u \approx v ; v) [\varphi] \in \mathcal{E}_{i+1}$. Hence we may apply Lemma 5.3.20 (with $u \geq u [\varphi]$ and $v \geq v [\varphi]$) to obtain that $u\delta \leftrightarrow_{\mathcal{R};\mathcal{H}_i;\mathcal{E}_N}^{\{u\delta,v\delta\}^*} v\delta$. By Lemma 5.3.7, this implies $c_j \leftrightarrow_{\mathcal{R};\mathcal{H}_i;\mathcal{E}_N}^{\{u\delta,v\delta\}^*} c_{j+1}$, and by Lemma 5.3.8 we have $c_j \leftrightarrow_{\mathcal{R};\mathcal{H}_i;\mathcal{E}_N}^{\{\varsigma\gamma,\tau\gamma\}^*} c_{j+1}$ as required.

- ▷ $(\varsigma ; s \approx t ; \tau) [\psi] \notin \mathcal{E}$, so $\mathcal{E}_i = \mathcal{E} \uplus \{(\varsigma ; s \approx t ; \tau) [\psi]\}$ and $\mathcal{E}_{i+1} = \mathcal{E} \cup \{(s ; s \approx t ; t) [\psi]\}$. Then the conditions to apply Lemma 5.3.20 are satisfied:

- by (**) and definition of a bounded equation context: $\varsigma \geq s [\psi]$ and $\tau \geq t [\psi]$
- $\mathcal{H}_{i+1} = \mathcal{H}_i \cup \{s \approx t [\psi]\}$
- for all ground substitutions δ that respect ψ we have $s\delta \leftrightarrow_{\mathcal{R};\mathcal{H}_{i+1};\mathcal{E}_N}^{\{s\delta,t\delta\}^*} t\delta$ by the induction hypothesis because $(s ; s \approx t ; t) [\psi] \in \mathcal{E}_{i+1}$

Hence, we immediately conclude that $s\gamma \leftrightarrow_{\mathcal{R};\mathcal{H}_i;\mathcal{E}_N}^{\{\varsigma\gamma,\tau\gamma\}^*} t\gamma$. □

With this, we can easily obtain a core soundness claim of Bounded RI:

Theorem 5.3.21. Let $\mathcal{L}$ be a weakly normalizing, quasi-reductive LCSTRS and let $\mathcal{E}$ be a set of bounded equation contexts. Let $(\succ, \geq)$ be some bounding pair, such that $(\mathcal{E}, \emptyset) \vdash^* (\emptyset, \mathcal{H})$, for some $\mathcal{H}$ using the derivation rules in Figure 5.1.

Then for every equation context $(\varsigma; s \approx t; \tau) [\psi] \in \mathcal{E}$, and every ground substitution γ that respects ψ , we have: $s\gamma \xrightarrow[\mathcal{R}]{\{\varsigma\gamma; \tau\gamma\}}^* t\gamma$, where $u \xrightarrow[\mathcal{R}]{\{\varsigma\gamma; \tau\gamma\}} v$ if $u \xrightarrow[\mathcal{R}; \emptyset]{\{\varsigma\gamma; \tau\gamma\}}$ by (A).

Proof. By Lemmas 5.3.10, 5.3.11, 5.3.12, 5.3.13, 5.3.14, 5.3.15, 5.3.16 and 5.3.17, all derivation rules of Figure 5.1 other than (Induct) have the Base Soundness Property. By Lemma 5.3.2 they also all preserve bounds. Hence, we apply Theorem 5.3.18 and find that, choosing $i = 1$ (with $\mathcal{E}_1 = \mathcal{E}$) we conclude that every equation context in $\mathcal{E}$ is $\mathcal{R}/\emptyset/\emptyset$ -bounded ground convertible. $\square$

We obtain the original soundness result as a corollary:

Theorem 5.2.5. Let $\mathcal{L} = (\Sigma, \mathcal{R})$ be a weakly normalizing, quasi-reductive LCSTRS; $\mathcal{A}$ a set of equations; and let $\mathcal{E}$ be the set of equation contexts $\{(\bullet; s \approx t; \bullet) [\psi] \mid s \approx t [\psi] \in \mathcal{A}\}$. Let $(\succ, \geq)$ be some bounding pair, such that $(\mathcal{E}, \emptyset) \vdash^* (\emptyset, \mathcal{H})$, for some $\mathcal{H}$ using the derivation rules in Figure 5.1. Then every equation in $\mathcal{A}$ is an inductive theorem.

Proof. Clearly, equation contexts of the form $(\bullet; s \approx t; \bullet) [\psi]$ are bounded. Hence, the result immediately follows by Theorem 5.3.21. $\square$

5.3.5 Completeness

To prove Theorem 5.2.19, we start with a proposition.

Proposition 5.3.22. Let $\mathcal{R}$ be a quasi-reductive, ground confluent LCSTRS and let $\mathcal{E}$ be a set of equation contexts. If $(\mathcal{E}, \mathcal{H}, F) \vdash \perp$ then there exist an equation context $(\varsigma; s \approx t; \tau) [\psi] \in \mathcal{E}$ and a ground substitution γ that respects $(\varsigma; s \approx t; \tau) [\psi]$ for which $s\gamma \xleftrightarrow[\mathcal{R}]^* t\gamma$ does not hold.

Proof. By definition of ground confluence, we have $s\gamma \xleftrightarrow[\mathcal{R}]^* t\gamma$ exactly if $s\gamma \rightarrow_{\mathcal{R}}^* \leftarrow t\gamma$ (where $u \rightarrow_{\mathcal{R}}^* \leftarrow v$ means that there exists a term w such that $u \rightarrow_{\mathcal{R}}^* w$ and $v \rightarrow_{\mathcal{R}}^* w$).

The only derivation rule that allows us to deduce $\perp$ is (Disprove), and by definition of this rule, $\mathcal{E}$ contains an equation context $(\varsigma; s \approx t; \tau) [\psi]$ such that $s \approx t [\psi]$ is contradictory. We consider all possibilities of being contradictory.

(1): there exist $f, g \in \Sigma$ with $f \neq g$ and a ground substitution γ that respects ψ such that $s\gamma = f s_1 \dots s_m$ and $t\gamma = g t_1 \dots t_m$ with $n < ar(f)$ and $m < ar(g)$. *Claim:* γ is a substitution for which $s\gamma \rightarrow_{\mathcal{R}}^* \leftarrow t\gamma$ does not hold (and therefore neither does $s\gamma \xleftrightarrow[\mathcal{R}]^* t\gamma$). To see this, assume that there exists u such that both $s\gamma \rightarrow_{\mathcal{R}}^* u$ and $t\gamma \rightarrow_{\mathcal{R}}^* u$. By the arity restriction, $s\gamma$ cannot reduce at the root, and nor can any of its reducts. Therefore, $head(u) = f$. Similarly, we conclude $head(u) = g$. This gives the desired contradiction.

(2): $s, t \in T(\Sigma_{theory}, \mathcal{V})$ and there is a ground substitution γ that respects ψ , with $\llbracket (\psi \wedge s \neq t)\gamma \rrbracket = \top$. Hence, γ maps every variable in $\text{Var}(s) \cup \text{Var}(t)$ to a ground theory term. Using only $\rightarrow_{\mathcal{R}_{calc}}$ we reduce $s\gamma$ and $t\gamma$ to their normal forms (which must be values). We have $\llbracket s\gamma \rrbracket \neq \llbracket t\gamma \rrbracket$, and since $\mathcal{R}_{calc}$ rules preserve interpretation, this implies $\llbracket (s\gamma)\downarrow \rrbracket \neq \llbracket (t\gamma)\downarrow \rrbracket$. But then also $(s\gamma)\downarrow \neq (t\gamma)\downarrow$, as there is a one-to-one correspondence between values and their interpretations. Hence, $s\gamma \rightarrow_{\mathcal{R}}^* t\gamma$ does not hold. $\square$

Proposition 5.3.22 only proves the desired statement for a single-step deduction. To obtain the full statement of Theorem 5.2.19, we will use an inductive reasoning.

Theorem 5.2.19. Let $\mathcal{L}$ be a weakly normalizing, quasi-reductive, ground confluent LCSTRS and let $\mathcal{E}$ be a set of equations. If $(\mathcal{E}, \emptyset, \mathcal{C}) \vdash^* \perp$ then there is an equation in $\mathcal{E}$ which is not an inductive theorem.

Proof. Assume given a deduction

$$(\mathcal{E}_1, \mathcal{H}_1, F_1) \vdash (\mathcal{E}_2, \mathcal{H}_2, F_2) \vdash^* (\mathcal{E}_n, \mathcal{H}_n, F_n) \vdash \perp$$

with $\mathcal{H}_1 = \emptyset$ and $F_1 = \mathcal{C}$. We first observe that (**) for all i , either $F_i = \mathcal{C}$, or $F_i = \mathcal{I}(\mathcal{E}_k)$ for some $k < i$. This follows easily by induction on i .

Now, towards a contradiction, assume that all equations in $\mathcal{E}_1$ are inductive theorems; that is, that $\leftrightarrow_{\mathcal{E}_1} \subseteq \leftrightarrow_{\mathcal{R}}^*$ on ground terms. We show that this assumption implies that $\leftrightarrow_{\mathcal{E}_i} \cup \leftrightarrow_{\mathcal{H}_i} \subseteq \leftrightarrow_{\mathcal{R}}^*$ on ground terms for all $1 \leq i \leq n$, by induction on i (the contradiction then follows by considering $i = n$ and Proposition 5.3.22).

For $i = 1$ this is true by assumption (since $\mathcal{H}_1 = \emptyset$). So let $i > 1$.

If $(\mathcal{E}_{i-1}, \mathcal{H}_{i-1}, F_{i-1}) \vdash (\mathcal{E}_i, \mathcal{H}_i, F_i)$ uses (StepCC), then $(\mathcal{E}_{i-1}, \mathcal{H}_{i-1}) \vdash (\mathcal{E}_i, \mathcal{H}_i)$ by a deduction rule with the Completeness Property, so by definition of this property and the induction hypothesis we immediately obtain $\leftrightarrow_{\mathcal{E}_i} \cup \leftrightarrow_{\mathcal{H}_i} \subseteq \leftrightarrow_{\mathcal{R}}^*$ on ground terms.

If (Complete) is used, then $\mathcal{H}_i = \mathcal{H}_{i-1}$ and by (**) $\mathcal{E}_i \subseteq \mathcal{E}_k$ for some $k < i$. By the induction hypothesis we both have $\leftrightarrow_{\mathcal{H}_{i-1}} \subseteq \leftrightarrow_{\mathcal{R}}^*$ and $\leftrightarrow_{\mathcal{E}_k} \subseteq \leftrightarrow_{\mathcal{R}}^*$, which suffices.

Otherwise, (StepCI) or (StepII) is used. First, observe that $\leftrightarrow_{\mathcal{H}_i} \subseteq \leftrightarrow_{\mathcal{R}}^*$: let $s \approx t [\psi] \in \mathcal{H}_i$ and γ be a ground substitution that respects ψ . If $s \approx t [\psi] \in \mathcal{H}_{i-1}$ then $s\gamma \leftrightarrow_{\mathcal{R}}^* t\gamma$ by the induction hypothesis. Otherwise, the only deduction rule for which $\mathcal{H}_i \setminus \mathcal{H}_{i-1} \neq \emptyset$ is (Induct), in which case there is some $(\varsigma ; s \approx t ; \tau) [\psi] \in \mathcal{E}_{i-1}$, and therefore we also have $s\gamma \leftrightarrow_{\mathcal{R}}^* t\gamma$ by the induction hypothesis.

Next, we consider $\mathcal{E}_i$. Whether (StepCI) or (StepII) is used, by (**) $F_i = \mathcal{I}(\mathcal{E}_k)$ for some $k < i$. For $j > i$, if $F_{j-1} = \mathcal{I}(\mathcal{E}_k)$, there are only two options: $F_j = \mathcal{I}(\mathcal{E}_k)$ as well, or the step $(\mathcal{E}_{j-1}, \mathcal{H}_{j-1}, F_{j-1}) \vdash (\mathcal{E}_j, \mathcal{H}_j, F_j)$ follows by (Complete), and therefore $\mathcal{E}_j \subseteq \mathcal{E}_k$. Since $F_n = \mathcal{C}$, necessarily the latter must at some point occur. Hence, there exists $j > i$ with $\mathcal{E}_j \subseteq \mathcal{E}_k$, so by the induction hypothesis, $\leftrightarrow_{\mathcal{E}_i} \subseteq \leftrightarrow_{\mathcal{R}}^*$.

Let $(\varsigma ; s \approx t ; \tau) [\psi] \in \mathcal{E}_i$ and γ a ground substitution that respects ψ . Then by Theorem 5.3.18 we have $s\gamma \xrightarrow[\mathcal{R}; H_i; \mathcal{E}_i]{\{\varsigma\gamma, \tau\gamma\}^*} t\gamma$, and since we have already seen that both $\leftrightarrow_{\mathcal{H}_i}$ and $\leftrightarrow_{\mathcal{E}_i}$ are included in $\leftrightarrow_{\mathcal{R}}^*$, we conclude $s\gamma \leftrightarrow_{\mathcal{R}}^* t\gamma$ as required. $\square$

Note that this proof only relies on the fact that that derivation rules other than (Induct) and (Disprove) do not change $\mathcal{H}$. Hence, like Theorem 5.3.18, the result is applicable to extensions of Figure 5.1.

5.4 Finding a Bounding Pair

The bounding pair $(>, \geq)$ is an important component of bounded rewriting induction. Yet, how to find it in practice? As we have seen, we do not have to fix the pair in advance, but can use the process of bounded rewriting induction to accumulate requirements. Let us say the process delivers a set REQS of requirements of the form $s > t [\varphi]$ or $s \geq t [\varphi]$ with $s \neq t$ (if $s = t$, then we already know $s \geq t [\varphi]$ is true and $s > t [\varphi]$ is false if φ is satisfiable). We are then still left with the difficulty of proving that a bounding pair for REQS exists. We discuss two different approaches to define such a pair.

5.4.1 Using a reduction ordering

A *reduction ordering* is a monotonic (i.e. $s \sqsupset t$ implies $C[s] \sqsupset C[t]$ for every context $C[\square]$) well-founded partial ordering on the set of terms. Monotonicity does not imply that $s \sqsupset t$ whenever $s \triangleright t$ but we *can* use a reduction ordering that includes $\rightarrow_{\mathcal{R}}$ to construct a bounding pair, as follows:

- ▷ define $a >_1 b$ if $a \sqsupset b$ or $a \triangleright b$; define $a > b$ if $a >_1^+ b$, and $a \geq b$ if $a >_1^* b$;
- ▷ require $\ell\gamma \sqsupset r\gamma$ for each $\ell \rightarrow r [\varphi] \in \mathcal{R}$ and ground substitution γ that respects φ

The monotonicity requirement ensures that well-foundedness of $\sqsupset$ is preserved in $>$: if $s \sqsupset t = C[t']$ and $t' \sqsupset u$, then $s \sqsupset C[t'] \sqsupset C[u]$, so any infinite sequence of $>$ steps can be converted into an infinite sequence of $\sqsupset$ steps.

There are various methods to define a reduction ordering in traditional term rewriting, with recursive path orderings and polynomial interpretations being among the most well-known. Unfortunately, there are not as many methods for higher-order or constrained term rewriting, and even fewer for the combination (although a variant of the recursive path ordering for LCSTRSs exists [25]). That being said, while Bounded RI is defined on LCSTRSs, it can also be used on unconstrained first-order term rewriting systems, where we have many methods at our disposal.

Using a rewrite relation However, it is worth noting that if Q is a set of rules such that $\mathcal{R} \cup Q$ is a terminating rewrite system, then $\rightarrow_{Q \cup \mathcal{R}}^+$ is by definition a reduction ordering. So, if every $s > t [\varphi]$ or $s \geq t [\varphi]$ in REQS gives a valid rewrite rule $s \rightarrow t [\varphi]$, we can simply let Q be this set of rules, and use any method to prove termination. Thus, we could in particular use the dependency pair framework [11, 23], which allows reduction orderings to be used in a more liberal way by requiring *weak* instead of full monotonicity.

A downside of this approach is the possibility to encounter requirements that are not valid rules, and this is especially common in higher-order rewriting since the left-hand side of a rule may not have a variable as the head, and must have the same type as the right-hand side. For example, suppose $\text{REQS} \ni f(F\ x)\ y > y - 1 [x > y]$, with $f :: \text{int} \rightarrow \text{int} \rightarrow \text{list}$. We cannot include a rule $f(F\ x)\ y \rightarrow y - 1 [x > y]$ in Q , because the two sides have a different type (list versus int). Even if they had the same type, the only dependency pair framework for LCSTRSs defined so far [24] does not support rules whose left-hand side contains applied variables, so the subterm $F\ x$ is problematic.

A solution is to not derive Q directly from the set REQS, but merely to ensure that $s\gamma (\rightarrow_{Q \cup \mathcal{R}} \cup \triangleright)^+ t\gamma$ for all $s > t [\varphi]$ or $s \geq t [\varphi]$ in REQS and γ that respect φ . For example, if we introduce a new function symbol $\text{inttolist} :: \text{int} \rightarrow \text{list}$, then the above requirement is handled by including a rule $f\ z\ y \rightarrow \text{inttolist}\ (y - 1)\ [\text{true}]$ in Q . While it is not immediately obvious how to build such an abstraction in general, at least it is easy to avoid the issue of different types through the introduction of constructors like inttolist .

An advantage of using a reduction relation is that it generates a bounding pair that satisfies the premise of Lemma 5.3.3. Hence, by taking some care with the use of (Hypothesis), (Alter) and (Generalize), we can ensure that equation contexts are always *strongly* bounded. This yields substantially fewer ordering requirements – and thus an easier proof search.

5.4.2 Separating top and inner steps

The idea explored above – of deriving a bounding pair from a single reduction ordering – is quite powerful and allows us to reuse existing termination methods. In particular the approach of letting $\sqsupset$ be a terminating relation $\rightarrow_{Q \cup \mathcal{R}}$ is reminiscent of traditional ways of using RI [21, 22, 27, 43], where the induction hypotheses in $\mathcal{H}$ are oriented as rules, and we prove termination of $\rightarrow_{\mathcal{R} \cup \mathcal{H}}$.

However, in these approaches we do not fully take advantage of the weaker requirements in the present setting. In particular, elements of REQS do not need to be oriented with a *monotonic* relation. For example, if $\mathcal{R} = \{f\ a \rightarrow f\ b\}$ and $\text{REQS} = \{b > a\}$, there is no reduction ordering $\sqsupset$ that orients REQS, since then we would need both $f\ a \sqsupset f\ b$ and $f\ b \sqsupset f\ a$ (by monotonicity). Yet, there *are* bounding pairs for these requirements: e.g., the relation $(\rightarrow_{\{b \rightarrow a\}[\text{true}]}, \text{top} \cup \rightarrow_{\mathcal{R}} \cup \triangleright)^+$, whose subrelation $\rightarrow_{Q, \text{top}}$ is defined by:

Definition 5.4.1. For a set Q of triples (ℓ, r, φ) with ℓ and r terms and φ a constraint, we define $s \rightarrow_{Q, \text{top}} t$ if there exist $(\ell, r, \varphi) \in Q$ and a substitution γ that respects φ , such that $s = \ell\gamma$ and $t = r\gamma$.

In terms of the (first-order) dependency pair framework, the difference can roughly be summarised as follows: termination of $\rightarrow_{Q \cup \mathcal{R}}$ coincides with finiteness of the DP problem $(\text{DP}(Q \cup \mathcal{R}), Q \cup \mathcal{R})$, whereas termination of $\rightarrow_{Q, \text{top}} \cup \rightarrow_{\mathcal{R}} \cup \triangleright$ coincides with finiteness of $(\text{DP}(Q \cup \mathcal{R}), \mathcal{R})$. In higher-order rewriting, there are multiple dependency pair frameworks, and their full introduction requires some theoretical development that is beyond the scope of this thesis. However, we will show how the question whether a suitable bounding pair exists can be reduced to the existence of something much like a dependency pair chain.

Dependency-pair-like chains In the following, we assume that REQS contains only requirements of the form $f \vec{\ell} > r [\varphi]$ or $f \vec{\ell} \geq r [\varphi]$ with $f \in \Sigma$. We define:

- $\text{REQS}_{\geq} = \{(\ell, r, \varphi) \mid \ell \geq r [\varphi] \in \text{REQS}\}$
 $\text{REQS}_{>} = \{(\ell, r, \varphi) \mid \ell > r [\varphi] \in \text{REQS}\}$
- $\mathcal{R}' = \left\{ (\ell \ x_1 \cdots x_i, r \ x_1 \cdots x_i, \varphi) \left| \begin{array}{l} \ell \rightarrow r [\varphi] \in \mathcal{R}, \ell :: \sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow \iota \ (\iota \in \mathcal{S}), \\ 0 \leq i \leq n, x_1 :: \sigma_1, \dots, x_i :: \sigma_i \end{array} \right. \right\}$
- $\text{Defs} = \{(f, n) \mid (f \ \ell_1 \cdots \ell_n, r, \varphi) \in \mathcal{R}' \cup \text{REQS}_{>} \cup \text{REQS}_{\geq}\}$
- A *candidate* is a term $s = f \ s_1 \cdots s_n$ with either $f \in \mathcal{V}$ and $n > 0$, or $(f, n) \in \text{Defs}$
- $\mathcal{P}_{\text{weak}} = \{\ell \rightarrow r [\varphi] \in \text{REQS}_{\geq} \cup \mathcal{R}' \mid r \text{ is a candidate}\}$
 $\mathcal{P}_{\text{strong}} = \{\ell \rightarrow r [\varphi] \in Q_1 \mid r \text{ is a candidate}\} \cup$
 $\left\{ \ell \rightarrow p [\varphi] \left| \begin{array}{l} \ell \rightarrow r [\varphi] \in \text{REQS}_{>} \cup \text{REQS}_{\geq} \cup \mathcal{R}' \wedge r \triangleright p \wedge \\ p \text{ is a candidate} \end{array} \right. \right\}$
- A *strong chain* is an infinite sequence $(s_i)_i$ of terms such that
 $s_i \rightarrow_{\mathcal{P}_{\text{strong}, \text{top}}} t_i (\rightarrow_{\mathcal{R}} \cup \rightarrow_{\mathcal{P}_{\text{weak}, \text{top}}})^* s_{i+1}$ for all i .

Essentially, $\mathcal{P}_{\text{weak}} \cup \mathcal{P}_{\text{strong}}$ is the set of all *dynamic dependency pairs* [36] of $\text{REQS} \cup \mathcal{R}'$, except that we did not mark the head symbol of the left-hand side; such a marking can always be added at a later stage in a termination proof. Correspondingly, a strong chain is a dependency pair chain in which the DPs in $\mathcal{P}_{\text{strong}}$ occur infinitely often.

Lemma 5.4.2. For ground terms s, t , let $s \geq t$ if $s (\rightarrow_{\text{REQS}_{>} \cup \text{REQS}_{\geq}, \text{top}} \cup \rightarrow_{\mathcal{R}} \cup \triangleright)^* t$ and $s > t$ if there exist u, v such that $s \geq u \rightarrow_{\text{REQS}_{>}, \text{top}} v \geq t$. Suppose no strong chain exists.

Then $(>, \geq)$ is a bounding pair that orients all requirements in REQS.

Proof. Clearly $\geq$ is transitive and reflexive, being the transitive-reflexive closure of $\geq_{\text{base}} := \rightarrow_{\text{REQS}_{>}} \cup \text{REQS}_{\geq, \text{top}} \cup \rightarrow_{\mathcal{R}} \cup \triangleright$. We have $> \subseteq \geq$ because $\rightarrow_{\text{REQS}_{>}, \text{top}} \subseteq \geq_{\text{base}} \subseteq \geq$ and $\geq$ is transitive, while transitivity of $>$ follows because $a \geq \cdot \rightarrow_{\text{REQS}_{>}, \text{top}} \cdot \geq b \geq \cdot \rightarrow_{\text{REQS}_{>}, \text{top}} \cdot \geq c$ implies $a \geq \cdot \rightarrow_{Q_1, \text{top}} \cdot \geq \cdot \geq \cdot \geq_{\text{base}} \cdot \geq c$ and $\geq$ is transitive. The properties $> \cdot \geq \subseteq \geq$ and $\geq \cdot > \subseteq \geq$ follow similarly. It is obvious that all requirements in REQs are oriented. All that remains to be shown is well-foundedness of $>$.

Towards a contradiction, suppose $>$ is not well-founded; that is, there is a sequence of $\rightarrow_{\text{REQS}_{>}, \text{top}} \cup \rightarrow_{\text{REQS}_{\geq}, \text{top}} \cup \rightarrow_{\mathcal{R}} \cup \triangleright$ steps in which $\rightarrow_{\text{REQS}_{>}, \text{top}}$ appears infinitely often. We define that a term s is *non-terminating* if there is an infinite such sequence starting in s , *terminating* if there is not, and *minimal non-terminating* (MNT) if s is non-terminating but all its strict subterms terminate; i.e., t is terminating whenever $s \triangleright t$.

Since $\geq$ includes both $\triangleright$, we first make the following important observation:

(OBS1) if u is terminating and $u \triangleright q$ then also q is terminating.

Using this property along with an induction on the size of s , we easily observe:

(OBS2) if $s\gamma$ is non-terminating but for all $x \in \text{Var}(s)$ the term $\gamma(x)$ is terminating, then there is a non-variable term t with $s \geq t$ such that $t\gamma$ is minimal non-terminating.

If an MNT term $s = f u_1 \cdots u_n$ is reduced with $\rightarrow_{\text{REQS}_{>} \cup \text{REQS}_{\geq}, \text{top}}$ then $(f, n) \in \text{Defs}$. If it is reduced with $\rightarrow_{\mathcal{R}}$ at the top or head, then it could be reduced with $\rightarrow_{\mathcal{R}', \text{top}}$, so also $(f, n) \in \text{Defs}$. If $s \triangleright t$ then by definition t is no longer MNT. And if $s \rightarrow_{\mathcal{R}} t$ by some reduction not at the head, then $t = f u_1 \cdots u'_i \cdots u_n$ with $u_i \rightarrow_{\mathcal{R}} u'_i$; so t has the same outer shape, and is terminating or MNT. This last point holds because u'_i and each u_j are terminating, as is $f u_1 \cdots u'_i \cdots u_{n-1}$ (being equal to, or a reduct of, the terminating subterm $f u_1 \cdots u_{n-1}$ of s). Hence we conclude:

(OBS3) if $s = f u_1 \cdots u_n$ is MNT then $(f, n) \in \text{Defs}$

Finally, again using (OBS1) we have:

(OBS4) for $(f \ell_1 \cdots \ell_k, r, \varphi) \in \text{REQS}_{>} \cup \text{REQS}_{\geq} \cup \mathcal{R}'$ and substitution γ : if $\ell_i \gamma$ is terminating, then so is each $\gamma(x)$ with $x \in \text{Var}(\ell_i)$.

Towards a contradiction, we will construct a strong chain. From the assumption that $>$ is not well-founded, we know that a non-terminating term exists, and use (OBS2) with $\gamma = []$ to obtain a MNT term t_0 . In the following, denote $\rightarrow_{\mathcal{R}, \text{in}}$ for a step *not* at the top or head of a term, i.e., $f s_1 \cdots s_j \cdots s_n \rightarrow_{\mathcal{R}, \text{in}} f s_1 \cdots s'_j \cdots s_n$ if $s_j \rightarrow_{\mathcal{R}} s'_j$.

Now, for natural number i , assume given a minimal non-terminating term t_i . Then, denoting $\leadsto$ for the relation $(\rightarrow_{\text{REQS}_{\geq}, \text{top}} \cup \rightarrow_{\mathcal{R}} \cup \triangleright)$, non-termination of t_i implies that there is a reduction $t_i = a_1 \leadsto a_2 \leadsto \cdots \leadsto a_m \rightarrow_{\text{REQS}_{>}, \text{top}} b$, with b still non-terminating. We prove, by induction on $m - j$ for $j \in \{1, \dots, m\}$, that if a_j is MNT then there exist MNT terms s_{i+1} and t_{i+1} such that $a_j (\rightarrow_{\mathcal{R}, \text{in}} \cup \rightarrow_{\mathcal{P}_{\text{weak}}, \text{top}})^* s_{i+1}$ and $s_{i+1} \rightarrow_{\mathcal{P}_{\text{strong}}, \text{top}} t_{i+1}$.

▷ If $j = m$ then $a_j = \ell \gamma$ for some $(\ell, r, \varphi) \in \text{REQS}_{>}$; write $\ell = f \ell_1 \cdots \ell_k$. Then $(f, k) \in \text{Defs}$, and each $\ell_n \gamma$ is terminating by minimality. By (OBS4), also all $\gamma(x)$

are terminating. As $b = r\gamma$, by (OBS2), there is a non-variable subterm $r \geq p$ such that $p\gamma$ is MNT. By (OBS3), $p\gamma$ is a candidate, and because p is not itself a variable, this means p must be a candidate. Hence, we can let $s_{i+1} := a_j = \ell\gamma$ and $t_{i+1} := p\gamma$.

- ▷ If $j < m$ and $a_j \triangleright a_{j+1}$, then $a'_j \geq a_{j+1}$ for some immediate argument term a'_j of a_j , and therefore a_{j+1} would be terminating by definition of a_j being MNT. This is not possible as a_{j+1} reduces to the non-terminating term b .
- ▷ If $j < m$ and $a_j \rightarrow_{\mathcal{R}} a_{j+1}$ by a step *not* at the head, then we write $a_j = f u_1 \cdots u_n \cdots u_m$ and $a_{j+1} = f u_1 \cdots u'_n \cdots u_m$. All arguments u_l are terminating, and therefore so is u'_n . Also, $f u_1 \cdots u_{m-1}$ is terminating (as a strict subterm of a_j), so if $n' < m$ then $f u_1 \cdots u'_n \cdots u_{m-1}$ is terminating as well. Hence, all strict subterms of a_{j+1} are terminating, and a_{j+1} is still MNT. Hence, by the induction hypothesis, $a_j \rightarrow_{\mathcal{R}, \text{in}} a_{j+1} (\rightarrow_{\mathcal{R}, \text{in}} \cup \rightarrow_{\mathcal{P}_{\text{weak}, \text{top}}})^* s_{i+1}$ and $s_{i+1} \rightarrow_{\mathcal{P}_{\text{strong}, \text{top}}} t_{i+1}$ for suitable s_{i+1}, t_{i+1} .
- ▷ Finally, if $j < m$ and either $a_j \rightarrow_{\mathcal{R}} a_{j+1}$ at the head, or $a_j \rightarrow_{\text{REQS}_{\geq}, \text{top}} a_{j+1}$, then there is some $(\ell, r, \varphi) \in \mathcal{R}' \cup \text{REQS}_{\geq}$ such that $a_j = \ell\gamma$ and $a_{j+1} = r\gamma$ and γ respects φ . By (OBS4) and minimality of a_j , all $\gamma(x)$ are terminating. Since a_{j+1} is necessarily non-terminating (as it reduces to the non-terminating term b), clearly r is not a variable. By (OBS2) there is a non-variable term p with $r \geq p$ such that $p\gamma$ is MNT. By (OBS3) and because p is not a variable, p is a candidate. There are two options:

- If $r = p$, then $(\ell, r, \varphi) \in \mathcal{P}_{\text{weak}}$. Hence, using the induction hypothesis, $a_j \rightarrow_{\mathcal{P}_{\text{weak}, \text{top}}} a_{j+1} (\rightarrow_{\mathcal{R}, \text{in}} \cup \rightarrow_{\mathcal{P}_{\text{weak}, \text{top}}})^* s_{i+1}$ and $s_{i+1} \rightarrow_{\mathcal{P}_{\text{strong}, \text{top}}} t_{i+1}$ for some MNT s_{i+1}, t_{i+1} .
- If $r \triangleright p$, then $(\ell, p, \varphi) \in \mathcal{P}_{\text{strong}}$. Hence, we let $s_{i+1} := \ell\gamma$ and $t_{i+1} := p\gamma$.

In particular (for $j = 1$), $t_i (\rightarrow_{\mathcal{R}} \cup \rightarrow_{\mathcal{P}_{\text{weak}, \text{top}}})^* s_{i+1}$ and $s_{i+1} \rightarrow_{\mathcal{P}_{\text{strong}, \text{top}}} t_{i+1}$. We have thus constructed a strong chain, which contradicts the lemma's assertion that no strong chain exists. We conclude a contradiction with the assumption that $\triangleright$ is not well-founded. $\square$

Overall, if all elements of REQS have a shape $f s_1 \cdots s_n > t [\varphi]$ or $f s_1 \cdots s_n \geq t [\varphi]$ with f not a variable, then we know that a suitable bounding pair for REQS exists so long as we can prove the absence of a strong chain.

If the elements of REQS do not all have this form, we may still be able to apply a similar approach (by changing REQS so the original requirements are captured by $(\triangleright, \geq)$ defined in Lemma 5.4.2), but such techniques are beyond the scope of this section.

Reduction triples To directly investigate the presence of strong chains we can use an extension of the notion of reduction ordering:

Corollary 5.4.3. A reduction triple is a triple $(\triangleright, \triangleright, \geq)$ of three relations such that:

- $>$ is a well-founded partial ordering on the set of terms
- $\gtrsim$ is a quasi-ordering on the set of terms, such that $s > t \gtrsim u$ implies $s > u$
- $\geq$ is a monotonic quasi-ordering on the set of terms, such that $s \geq t$ implies $s \gtrsim t$

There exists no infinite strong chain if we can find a reduction triple $(>, \gtrsim, \geq)$ such that:

- for all $(\ell, p, \varphi) \in \mathcal{P}_{\text{strong}}$ and all γ that respect φ : $\ell\gamma > p\gamma$
- for all $(\ell, p, \varphi) \in \mathcal{P}_{\text{weak}}$ and all γ that respect φ : $\ell\gamma \gtrsim p\gamma$
- for all $\ell \rightarrow r[\varphi] \in \mathcal{R}$ and all γ that respect φ : $\ell\gamma \geq r\gamma$

The advantage of a reduction triple compared to a reduction ordering is that the monotonicity requirement is replaced by *weak* monotonicity: only $\geq$ needs to be monotonic. It also allows us to more easily take advantage of the difference between $>$ and $\geq$ requirements.

Reduction triples can be constructed using for instance argument filterings or weakly monotonic algebras. While there has not yet been extensive research in this direction for higher-order constrained rewriting, this is a natural direction for future work.

5.5 Obtaining new deduction rules more easily

In Section 5.3, we set out to prove soundness and completeness of Bounded RI in a modular way: rather than merely proving that the system as given in Figure 5.1 satisfies Theorem 5.2.5, we have shown the stronger Theorem 5.3.18. This means that any extension of the system with derivation rules that satisfy both the Preserving Bounds Property (Definition 5.3.1) and Base Soundness Property (Definition 5.3.9), is still sound; and if these rules satisfy the Completeness Property (Definition 5.2.15) they can also be used in a non-equivalence proof.

Let us discuss two naturally arising situations for which we can exploit this approach to introduce a new deduction rule, and establish its soundness relatively easy.

Calculations In Bounded RI, we followed the same approach for defining (Simplify) (and (Hypothesis)) as we did in HORI. We have, however, not yet mentioned that these definitions are more basic than the corresponding definition in first-order RI for LCTRSs [22] which is based on the more sophisticated notion of “constrained reduction”. Essentially, this allows the first-order (Simplify) to incorporate (Alter) steps, although in a slightly weaker way than in Bounded RI.

The price for the simplicity in Bounded RI is that we often require extra steps, especially when it comes to calculation rules. For example, in Section 5.2.2 we encountered the equation (for the sake of clarity we omit the bounding terms of the equation context):

$$f \ i \ (\text{RD } f \ n \ (i - 1) \ a) \approx \text{TU } f \ (n + 1) \ i \ (f \ n \ a) \ [i \geq n]$$

Here, we first had to apply (Alter) to obtain the equation

$$f \ i \ (\text{RD } f \ n \ (i - 1) \ a) \approx \text{TU } f \ (n + 1) \ i \ (f \ n \ a) \ [i' = i - 1 \wedge n' = n + 1 \wedge i \geq n]$$

before we could apply two (Simplify) steps, using calculation rules $i - 1 \rightarrow i'$ [$i' = i - 1$] and $n + 1 \rightarrow n'$ [$n' = n + 1$], turning the equation into

$$f \ i \ (\text{RD } f \ n \ i' \ a) \approx \text{TU } f \ n' \ i \ (f \ n \ a) \ [i' = i - 1 \wedge n' = n + 1 \wedge i \geq n]$$

With constrained reductions, we could do the (Simplify) steps directly to obtain the same result. We can regain this facility by observing that the following deduction rule

(Calc)

$$\frac{(\mathcal{E} \uplus \{(\varsigma; C[s_1, \dots, s_n] \approx t; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \{(\varsigma; C[x_1, \dots, x_n] \approx t; \tau) [\psi \wedge (x_1 = s_1) \wedge \dots \wedge (x_n = s_n)]\}, \mathcal{H})}$$

$n > 0$
 $s_1, \dots, s_n$ theory terms
 $x_1, \dots, x_n$ distinct fresh variables

can be considered as a shortcut for an (Alter) step, followed by some (Simplify) steps, possibly followed again by some (Alter) steps. (Provided all variables in each s_i are in $\text{Var}(\psi)$, or theory sorts are inextensible.) Therefore, it satisfies the two invariants.

Example 5.5.1. We may use (Calc) to replace an equation $f(x + (y + 1)) \approx t$ by $fz \approx t$ [$z = x + (y + 1)$]. To do so without (Calc) would require four steps:

- (E0) $f(x + (y + 1)) \approx t$
- (E1) $f(x + (y + 1)) \approx t \quad [q = y + 1 \wedge z = x + q] \quad \text{by (Alter)}$
- (E2) $f(x + q) \approx t \quad [q = y + 1 \wedge z = x + q] \quad \text{by (Simplify)}$
- (E3) $fz \approx t \quad [q = y + 1 \wedge z = x + q] \quad \text{by (Simplify)}$
- (E4) $fz \approx t \quad [z = x + (y + 1)] \quad \text{by (Alter)}$

This cannot be done with only one (Alter) step because the rules in $\mathcal{R}_{calc}$ do not have composite terms on their right-hand sides; there is no rule $x + (y + 1) \rightarrow z$ [$z = x + (y + 1)$].

Axioms We can use the two properties to go beyond just composite rules, though. An interesting example can be found in *incremental Rewriting Induction (iRI)* [4, Section 5], containing a deduction rules that allows for the use of *axioms*: equations that are known

to be consequences of $\mathcal{R}$, but for which we may not have a formal proof (in our setting a proof in Bounded RI). We can introduce a similar deduction rule in Bounded RI. Let $\mathcal{A}$ be a set of such axioms. We distinguish the following two cases:

- ▷ $\mathcal{R}$ is ground confluent, and all equations in $\mathcal{A}$ are ground convertible; i.e., $s\gamma \leftrightarrow_{\mathcal{R}}^* t\gamma$ for all $s \approx t [\varphi] \in \mathcal{A}$ and ground substitutions γ that respect the equation.
- ▷ $\mathcal{A}$ is a set of equations such that every equation context in $\mathcal{E}_{\mathcal{A}} = \{(\ell; \ell \approx r; r) [\varphi] \mid \ell \approx r [\varphi] \in \mathcal{A}\}$ is $\mathcal{R}/\emptyset/\emptyset$ -bounded ground convertible with respect to the *same* ordering pair $(>, \geq)$ that is used in the RI derivation under consideration.

Either way, we introduce the following deduction rule:

(Axiom)

$$\frac{(\mathcal{E} \uplus \{(\varsigma; C[\ell\delta] \simeq t; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \{(\varsigma; C[r\delta] \approx t; \tau) [\psi]\}, \mathcal{H})} \quad \begin{array}{l} \ell \simeq r [\varphi] \in \mathcal{A} \text{ and } \psi \models^{\delta} \varphi \\ \varsigma \geq C[r\delta] [\psi] \end{array}$$

Lemma 5.5.2. The (Axiom) rule preserves bounds.

Proof. If $(\varsigma; C[\ell\delta] \simeq t; \tau) [\psi]$ is a bounded equation context, then $\tau \geq t [\psi]$. Since the derivation rule requires $\varsigma \geq C[r\delta] [\psi]$, also $(\varsigma; C[r\delta] \approx t; \tau) [\psi]$ is bounded. $\square$

Lemma 5.5.3. (Axiom) satisfies the Base Soundness Property.

Proof. To see this, we will show that if $(\varsigma; C[\ell\delta] \simeq t; \tau) [\psi]$ is a bounded equation context and γ a gsc substitution which respects it, then $C[\ell\delta]\gamma \xleftarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}']^{\{\varsigma\gamma, \tau\gamma\} *} C[r\delta]\gamma \xleftarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}']^{\{\varsigma\gamma, \tau\gamma\}} t\gamma$ holds for any $\mathcal{E}, \mathcal{H}$ and $\mathcal{E}' = \mathcal{E} \cup \{(\varsigma; C[r\delta] \approx t; \tau) [\psi]\}$.

The latter step is obvious: by Definition 5.3.5.(C) we only have to show that $\{\varsigma\gamma, \tau\gamma\} \geq_{\text{mul}} \{\varsigma\gamma, \tau\gamma\}$, which is trivially true. So consider the former step.

In the first case, where $\mathcal{R}$ is ground confluent and $\ell\chi \leftrightarrow_{\mathcal{R}}^* r\chi$ for all ground substitutions χ that respect the equation, we have in particular that $\ell\delta\gamma \xrightarrow{\mathcal{R}}^* r\delta\gamma$.

Since $\xrightarrow{\mathcal{R}}$ is included in $\geq$ by definition of a bounding pair, $\ell\delta\gamma \xleftarrow[\mathcal{R}; \emptyset; \emptyset]^{\{\ell\delta\gamma, r\delta\gamma\} *} r\delta\gamma$ holds

by Definition 5.3.5.(A). In the second case, where all equation contexts in $\mathcal{E}_{\mathcal{A}}$ are $\mathcal{R}/\emptyset/\emptyset$ -bounded ground convertible, we have this property by definition. Either way,

$C[\ell\delta]\gamma \xleftarrow[\mathcal{R}; \emptyset; \emptyset]^{\{\ell\delta\gamma, r\delta\gamma\} *} C[r\delta]\gamma$ holds by Lemma 5.3.7, and since this derivation can only use

steps with $\mathcal{R}$, $C[\ell\delta]\gamma \xleftarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}']^{\{\ell\delta\gamma, r\delta\gamma\} *} C[r\delta]\gamma$ follows regardless of $\mathcal{H}$ and $\mathcal{E}'$. Observe that

$\varsigma\gamma \geq C[\ell\delta]\gamma \geq \ell\delta\gamma$ since the original equation context is bounded and $\geq$ includes $\triangleright$; and that $\tau\gamma \geq C[r\delta]\gamma \geq r\delta\gamma$ by definition of the rule. Hence, we apply Lemma 5.3.8 to

increase the bounds and obtain $C[\ell\delta]\gamma \xleftarrow[\mathcal{R}; \mathcal{H}; \mathcal{E}']^{\{\varsigma\gamma, \tau\gamma\} *} C[r\delta]\gamma$. $\square$

5.6 Ground Confluence

We now turn our eye to *ground confluence*. This property is both needed to deduce that an equation is *not* an inductive theorem using rewriting induction (using Section 5.2.3), and interesting to study in its own right – since, in a terminating and ground confluent system, reduction essentially computes a unique result for every ground term.

Existing work on first-order unconstrained rewriting [8] has shown that RI can be used as a tool to prove ground confluence of terminating TRSs: a system is ground confluent if all its critical pairs are bounded ground convertible, and RI can be used as a method to prove bounded ground convertibility. The existing work uses a slightly different definition of bounded ground convertibility than the one employed in this paper. We extend this result to our more liberal definition of bounded ground convertibility and to higher-order constrained rewriting. First, in Section 5.6.1, we give a definition of critical pairs and the closely related notion of critical *peaks* for LCSTRSs, and establish the relation between (ground) confluence and critical peaks. In Section 5.6.2 we generalize the first-order result [8], showing that, for terminating LCSTRSs, ground confluence is implied by bounded ground convertibility of all critical pairs. Since Bounded RI proves bounded ground convertibility, it can therefore be used to obtain ground confluence.

5.6.1 Critical pairs

Critical pairs are a standard tool for proving confluence. They have also been defined for first-order LCSTRSs [44]. This definition relies on the notion of *positions*, for which we recall the definition from Chapter 2.

Positions in LCSTRSs For a term $t = a \ t_1 \cdots t_n$ with $a \in \Sigma \cup \mathcal{V}$ and $n \geq 0$, the set of positions is defined as $\text{Pos}(t) = (\bigcup_{i=0}^n \{\star i\}) \cup (\bigcup_{i=1}^n \{i \cdot p \mid p \in \text{Pos}(t_i)\})$.

The subterm $t|_p$ of t at position $p \in \text{Pos}(t)$ is defined as follows:

- (i). $(a \ t_1 \cdots t_n)|_{\star i} = (a \ t_1 \cdots t_{n-i})$
- (ii). $(a \ t_1 \cdots t_n)|_{i \cdot p} = t_i|_p$

If t is a term, $p \in \text{Pos}(t)$, and s a term of the same type as $t|_p$, then let $t[s]_p$ be the term obtained from t by replacing $t|_p$ by s . We will denote the position $\star 0$ as ϵ .

The above definition differs from first-order positions because of including positions for partial applications; in first-order terms, we only have positions of the form $i_1 \cdots i_n \star 0$.¹

¹This definition also differs from the usual presentation of positions in higher-order rewriting, which does not use the $\star$ notation but rather sets $(s \ t)|_{1p} = s|_p$ and $(s \ t)|_{2p} = t|_p$. This difference is not significant. Rather, the presentation above is chosen because in practice (and in Cora) it is convenient if the positions used for first-order rewriting refer to the same subterm when viewing the term as higher-order.

We now recall the first-order definition of critical pairs [44] (only adapting it to omit the set $\mathcal{E}Var$ which is redundant here since we have required $\text{Var}(r_i) \subseteq \text{Var}(\ell_i) \cup \text{Var}(\varphi_i)$).

Definition 5.6.1 (Critical pairs). An *overlap* in an LCTRS $\mathcal{R}$ is a triple $\langle \rho_1, p, \rho_2 \rangle$ with rules $\rho_1 := \ell_1 \rightarrow r_1 [\varphi_1]$ and $\rho_2 := \ell_2 \rightarrow r_2 [\varphi_2]$ and position p , satisfying:

- (1). ρ_1 and ρ_2 are variable-disjoint renamings of rewrite rules in $\mathcal{R} \cup \mathcal{R}_{calc}$
- (2). $p \in \text{Pos}(\ell_2)$ with $\ell_2|_p$ not a variable
- (3). $\sigma = \text{mgu}(\ell_1, \ell_2|_p)$ and $\sigma(\text{Var}(\varphi_1) \cup \text{Var}(\varphi_2)) \subseteq \mathcal{V}al \cup \mathcal{V}$
- (4). $\varphi_1\sigma \wedge \varphi_2\sigma$ is satisfiable
- (5). If $p = \epsilon$ and ρ_1 and ρ_2 are renamings of the same rule, then $\text{Var}(r_1) \setminus \text{Var}(\ell_1) \neq \emptyset$.

The critical pair of $\langle \rho_1, p, \rho_2 \rangle$ is the equation

$$\ell_2[r_1]_p\sigma \approx r_2\sigma [\varphi_1\sigma \wedge \varphi_2\sigma]$$

The “overlap” here refers to the fact that ℓ_2 has a subterm that unifies with ℓ_1 . Therefore, if σ respects $\varphi_1 \wedge \varphi_2$ then $\ell_2\sigma$ can be reduced in two ways: since $\ell_2\sigma = (\ell_2\sigma)[\ell_2|_p\sigma]_p = (\ell_2\sigma)[\ell_1\sigma]_p$, we apply ρ_1 to obtain $\ell_2\sigma \rightarrow_{\mathcal{R}} (\ell_2\sigma)[r_1\sigma]_p$. Of course, we also have $\ell_2\sigma \rightarrow_{\mathcal{R}} r_2\sigma$.

Example 5.6.2. Let us apply Definition 5.6.1 to Example 5.2.14. We have the following variable-disjoint copies of H-rules, which are not renamings of each other

$$\begin{aligned} \rho_1 &:= \text{H } f \ n \ m \ x \rightarrow \text{H } f \ (n-1) \ m \ (f \ x) \quad [n > 0] \\ \rho_2 &:= \text{H } g \ i \ j \ y \rightarrow \text{H } g \ (j-1) \ i \ (g \ y) \quad [j > 0] \end{aligned}$$

There is overlap $\langle \rho_1, \epsilon, \rho_2 \rangle$ with $\sigma = [g := f, i := n, j := m, y := x]$, yielding a critical pair

$$\text{H } f \ (n-1) \ m \ (f \ x) \approx \text{H } f \ (m-1) \ n \ (f \ x) \ [n > 0 \wedge m > 0]$$

Remark 5.6.3. In Example 5.6.2 we did not list *all* critical pairs. Since the overlap occurs at the root position ϵ , there is also an overlap $\langle \rho_2, \epsilon, \rho_1 \rangle$ yielding the critical pair $\text{H } f \ (m-1) \ n \ (f \ x) \approx \text{H } f \ (n-1) \ m \ (f \ x) \ [n > 0 \wedge m > 0]$. For confluence techniques based on symmetric joinability conditions of critical pairs – like the one we will consider in this paper – we only need one of the two. For other confluence techniques, it is important to consider both [44].

Moreover, every calculation rule $f \ x_1 \cdots x_m \rightarrow y \ [y = f \ x_1 \cdots x_m]$ yields a critical pair with itself due to the overlap at the root: $y \approx z \ [y = f \ x_1 \cdots x_m \wedge z = f \ x_1 \cdots x_m]$. However, this is merely an artefact of the definition. In practice such critical pairs cause no issues because every substitution γ that respects $y = f \ x_1 \cdots x_m \wedge z = f \ x_1 \cdots x_m$ has $\gamma(y) = \gamma(z)$.

While this definition was designed for LCTRSs – so first-order systems where there are no applied variables or partially applied function symbols – our updated definition of positions allows the definition to be used unaltered for (higher-order) LCSTRSs.

Example 5.6.4. Consider an LCSTRS containing the following variable-disjoint rules

$$\rho_1 := f \rightarrow g \quad \rho_2 := u(f\ x) \rightarrow h\ x$$

We have an overlap $(\rho_1, 1 \star 1, \rho_2)$ with σ the empty substitution. This yields the critical pair $u(g\ x) \approx h\ x$, because $\ell_2\sigma = u(f\ x)$ reduces to both sides.

In some cases, it is useful to preserve the source $\ell_2\sigma$. We do so using critical peaks.

Definition 5.6.5 (Critical peaks). The critical peak of an overlap $\langle \rho_1, p, \rho_2 \rangle$ is the tuple

$$\ell_2[r_1]_p\sigma \leftarrow \ell_2\sigma \rightarrow r_2\sigma [\varphi_1\sigma \wedge \varphi_2\sigma]$$

The set of all critical peaks in an LCSTRSs is denoted by $\text{CP}(\mathcal{R})$.

Example 5.6.6. Following Example 5.6.2, we have $\ell_2\sigma = H\ f\ n\ m\ x$, so the corresponding critical peak is given by

$$H\ f\ (n-1)\ m\ (f\ x) \leftarrow H\ f\ n\ m\ x \rightarrow H\ f\ (m-1)\ n\ (f\ x) [n > 0 \wedge m > 0]$$

We also have a critical peak

$$H\ f\ (m-1)\ n\ (f\ x) \leftarrow H\ f\ n\ m\ x \rightarrow H\ f\ (n-1)\ m\ (f\ x) [n > 0 \wedge m > 0]$$

but we do *not* obtain critical peaks for the calculation rules: we excluded these by requiring that the second rule in the overlap must be from $\mathcal{R}$. (The first rule may still be from $\mathcal{R}_{calc}$, e.g. to get a critical pair when ρ_2 is a rule like $f(1+y) \rightarrow g\ y$.)

In Example 5.6.4 we only obtain one critical peak

$$u(g\ x) \leftarrow u(f\ x) \rightarrow h\ x [\text{true}]$$

Critical pairs rely on positions, which we have otherwise not used in our definitions. To relate them to the reduction relation, we use the following lemma.

Lemma 5.6.7. Let γ be a substitution, X a finite set of variables, and $s, t \in T(\Sigma, \mathcal{V})$ such that $s\gamma \rightarrow_{\mathcal{R}} t$ and $X \supseteq \text{Var}(s)$. Then one of the following holds.

- (i). There exist a substitution δ , variable-renamed rule $\ell \rightarrow r [\varphi] \in \mathcal{R} \cup \mathcal{R}_{calc}$ with variables disjoint from X , and $p \in \text{Pos}(s)$ such that $\delta(x) = \gamma(x)$ for all $x \in X$, δ respects φ , $s|_p \notin \mathcal{V}$, $s|_p\delta = \ell\delta$ and $t = s[r]_p\delta$.
- (ii). There exist $x \in \text{Var}(s)$ and substitution γ' such that $\gamma(x) \rightarrow_{\mathcal{R}} \gamma'(x)$, and $\gamma'(y) = \gamma(y)$ for all other y and $t \rightarrow_{\mathcal{R}}^* s\gamma'$.

Proof. Let $s\gamma \rightarrow_{\mathcal{R}} t$. By induction on the term shape of s we prove that either (i) or (ii) holds. Any $s \in T(\Sigma, \mathcal{V})$ can be written as $s = a s_1 \cdots s_n$ with $a \in \Sigma \cup \mathcal{V}$ and $n \geq 0$.

► If $a \in \mathcal{V}$ and $\gamma(a) \rightarrow_{\mathcal{R}} u$ and $t = u (s_1\gamma) \cdots (s_n\gamma)$ then define γ' as follows:

$$\gamma'(a) = u \quad \text{and} \quad \gamma'(y) = \gamma(y) \text{ if } y \neq a$$

Then clearly $t = u (s_1\gamma) \cdots (s_n\gamma) \rightarrow_{\mathcal{R}}^* u (s_1\gamma') \cdots (s_n\gamma') = s\gamma'$; thus, (ii) holds.

► If the reduction occurs at the top or head, then $s\gamma = (\ell\sigma) (s_{i+1}\gamma) \cdots (s_n\gamma) \rightarrow_{\mathcal{R}} (r\sigma) (s_{i+1}\gamma) \cdots (s_n\gamma) = t$ for some $0 \leq i \leq n$, and some σ which respects a rule $\ell \rightarrow r$ [φ] $\in \mathcal{R} \cup \mathcal{R}_{calc}$. If $i = 0$ and a is a variable we are in the first case and are done. Otherwise, let $p = \star(n - i)$. As we can safely assume (by renaming) that the variables in $\ell \rightarrow r$ [φ] do not occur in X , we define $\delta(x) = \gamma(x)$ for $x \in X$ and $\delta(x) = \sigma(x)$ for all other x . Then (i) holds.

► The only remaining option is a reduction in some subterm s_i with $1 \leq i \leq n$: $s_i\gamma \rightarrow_{\mathcal{R}} t_i$ and $t = (a\gamma) (s_1\gamma) \cdots t_i \cdots (s_n\gamma)$. By the induction hypothesis there are two possibilities:

- (i). There exist $\delta, \ell \rightarrow r$ [φ] and $p_i \in \text{Pos}(s_i)$ such that δ respects φ , $\delta(x) = \gamma(x)$ for all $x \in X$ (and therefore $s\gamma = s\delta$), $s_i|_{p_i} \notin \mathcal{V}$, $s_i|_{p_i}\delta = \ell\delta$ and $t_i = s_i[r]_{p_i}\delta$. Let $p = i p_i$. Then (i) also holds for $s\gamma \rightarrow_{\mathcal{R}} t$, since $s|_p = s_i|_{p_i}$, and $t = (a\gamma) (s_1\gamma) \cdots t_i \cdots (s_n\gamma) = (a\delta) (s_1\delta) \cdots (s_i[r]_{p_i}\delta) \cdots (s_n\delta) = s[r]_p\delta$.
- (ii). There exist $x \in \text{Var}(s_i)$ and substitution γ' such that $\gamma(x) \rightarrow_{\mathcal{R}} \gamma'(x)$, $\gamma'(y) = \gamma(y)$ for all $y \in \text{dom}(\gamma) \setminus \{x\}$ and $t_i \rightarrow_{\mathcal{R}}^* s_i\gamma'$. We have $s_j\gamma \rightarrow_{\mathcal{R}}^* s_j\gamma'$ for all $j \neq i$. Conclude $t = (a\gamma) (s_1\gamma) \cdots t_i \cdots (s_n\gamma) \rightarrow_{\mathcal{R}}^* (a\gamma) (s_1\gamma) \cdots (s_i\gamma') \cdots (s_n\gamma) \rightarrow_{\mathcal{R}}^* (a\gamma') (s_1\gamma') \cdots (s_i\gamma') \cdots (s_n\gamma') = s\gamma'$. $\square$

In analogy to the Critical Pair Lemma [13, Lemma 6.2.3] we introduce the Critical Peak Lemma.

Lemma 5.6.8 (Critical Peak Lemma). If both $s \rightarrow_{\mathcal{R}} t_1$ and $s \rightarrow_{\mathcal{R}} t_2$, then either

- (i). There exists a term t such that $t_1 \xrightarrow{\mathcal{R}}^* t \xleftarrow{\mathcal{R}}^* t_2$.
- (ii). There exist a critical peak $b \leftarrow a \rightarrow c$ [φ], context C and substitution δ that respects φ such that $s = C[a\delta]$ and either $t_1 = C[b\delta]$, $t_2 = C[c\delta]$, or $t_1 = C[c\delta]$, $t_2 = C[b\delta]$.

Proof. By induction on the term-shape of s we prove that either (i) or (ii) holds. Any $s \in T(\Sigma, \mathcal{V})$ can be written as $s = f s_1 \cdots s_n$ with $f \in \Sigma \cup \mathcal{V}$ and $n \geq 0$. We consider all possibilities of $t_1 \xrightarrow{\mathcal{R}} s \rightarrow_{\mathcal{R}} t_2$.

- (1). There exist $1 \leq i, j \leq n$ and terms v_i, w_j such that $s_i \rightarrow_{\mathcal{R}} v_i, s_j \rightarrow_{\mathcal{R}} w_j$ and

$$t_1 = s[v_i]_i \quad \text{and} \quad t_2 = s[w_j]_j$$

- If $i \neq j$ then the reductions are in parallel positions; (i) holds with $t = t_1[w_j]_j = t_2[v_i]_i$.
- If $i = j$ then $v_i \leftarrow_{\mathcal{R}} s_i \rightarrow_{\mathcal{R}} w_i$. By induction hypothesis we either have
 - There is a term t' such that $v_i \xrightarrow{\mathcal{R}}^* t' \xleftarrow{\mathcal{R}}^* w_i$. Then (i) holds: take $t = s[t']_i$.
 - There exist a critical peak $b \leftarrow a \rightarrow c$ $[\varphi]$, context C' and substitution δ that respects φ such that $s_i = C'[a\delta]$, $v_i = C'[b\delta]$ and $w_i = C'[c\delta]$. Then (ii) holds: take $C[\square] = s[C'[\square]]_i = f \ s_1 \cdots s_{i-1} \ C'[\square] \ s_{i+1} \cdots s_n$.

- (2). One of the reductions is at the head. Without loss of generality, we assume it is the step $s \rightarrow_{\mathcal{R}} t_1$. Then there is a rule $\rho_1 := \ell \rightarrow r$ $[\varphi] \in \mathcal{R} \cup \mathcal{R}_{calc}$ with $\ell = f \ v_1 \cdots v_k$ and substitution γ which respects φ , and $\vec{s} = s_{k+1} \cdots s_n$ such that $k = ar(f) \leq n$ and

$$s = (\ell\gamma)\vec{s} = f \ (v_1\gamma) \cdots (v_k\gamma) \ \vec{s} \rightarrow_{\mathcal{R}} (r\gamma)\vec{s} = t_1$$

We consider the possibilities for $s \rightarrow_{\mathcal{R}} t_2$

- $s_i \rightarrow_{\mathcal{R}} s'$, for some $i > k$, and $t_2 = s[s']_i$. Then (i) holds, as we can see by choosing $t = (r\gamma) \ s_{k+1} \cdots s_{i-1} \ s' \ s_{i+1} \cdots s_n$.
- $s_i = v_i\gamma \rightarrow_{\mathcal{R}} s'$, with $1 \leq i \leq k$, and $t_2 = s[s']_i$. By Lemma 5.6.7, applied with $X = \text{Var}(\ell) \cup \text{Var}(r) \cup \text{Var}(\varphi) \supseteq \text{Var}(v_i)$, there are two possibilities:
 - $s_i = v_i\gamma = v_i\delta$ for some δ that respects a renamed rule $\rho_2 := \ell' \rightarrow r'$ $[\varphi']$ with $\ell' = g \ l'_1 \cdots l'_m$ and there is a $p' \in \text{Pos}(v_i)$ such that $v_i|_{p'} \notin \mathcal{V}$, $v_i|_{p'}\delta = \ell'\delta$ and $s' = v_i[r']_{p'}\delta$; moreover, $\delta(x) = \gamma(x)$ on all $x \in X$. Then we observe that $\rho_1 \notin \mathcal{R}_{calc}$ (as v_i is not a variable), and

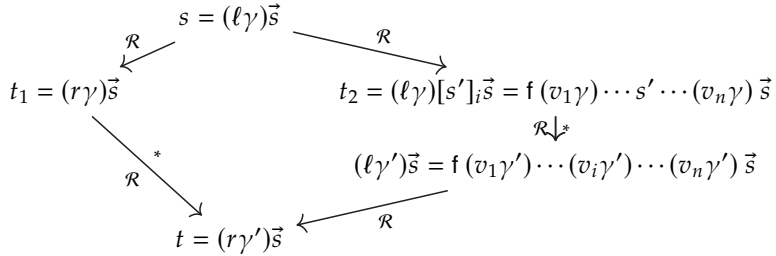
$$\begin{aligned} t_2 &= f \ (v_1\gamma) \cdots (v_{i-1}\gamma) \ (v_i[r']_{p'}\delta) \ (v_{i+1}\gamma) \cdots (v_k\gamma) \ \vec{s} \\ &= (f \ v_1 \cdots v_{i-1} \ v_i[r']_{p'} \ v_{i+1} \cdots v_k) \ \delta \ \vec{s} \end{aligned}$$

We will see that (ii) holds: let $C = \square \vec{s}$ and $p = i \ p'$. We have

$$t_2 = C \ [\ell[r']_p\delta] \leftarrow_{\mathcal{R}} s = C[\ell\delta] \rightarrow_{\mathcal{R}} t_1 = C[r\delta]$$

Since δ is a unifier of $\ell|_p = v_i|_{p'}$ and ℓ' , there exist a most general unifier σ and a substitution χ so that $\delta = \sigma\chi$. Since δ respects both φ (because it corresponds with γ on $X \supseteq \text{Var}(\varphi)$) and φ' , clearly $(\varphi' \wedge \varphi)\sigma$ is satisfiable (by the substitution χ). Hence, the peak above is an instance of the critical peak $\ell[r']_p\sigma \leftarrow \ell\sigma \rightarrow r\sigma$ $[\varphi'\sigma \wedge \varphi\sigma]$ for the overlap $\langle \rho_2, p, \rho_1 \rangle$.

- There exist $x \in \text{Var}(v_i)$ and substitution γ' such that $\gamma'(x) \rightarrow_{\mathcal{R}} \gamma'(x)$, $\gamma'(y) = \gamma(y)$ for all $y \neq x$ and $s' \xrightarrow{\mathcal{R}}^* v_i\gamma'$. Then (i) holds: choosing $t = (r\gamma')\vec{s}$ we have



- The other reduction is also at the head. Since each function symbol has a fixed arity, this means there exist a rule $\rho_2 := \ell' \rightarrow r' [\varphi']$ with $\ell' = f v'_1 \dots v'_k$ and substitution γ' which respects φ' such that $s = (\ell' \gamma') \vec{s}$ and $t_2 = (r' \gamma') \vec{s}$. There are two cases.

- If $r\gamma = r'\gamma'$, then (i) holds with $t = t_1 = t_2$. Note that this is always the case if ρ_2 is a calculation rule: then f is a calculation symbol and each $v_i\gamma = v'_i\gamma'$ a value. Since left-hand sides of rules in $\mathcal{R}$ may not be theory terms, necessarily $\rho_1 \in \mathcal{R}_{calc}$ and therefore we can only have $\rho_1 = \rho_2$. Since calculation is deterministic, $r\gamma = r'\gamma'$.

- Otherwise, (ii) holds with $C[\Box] = \Box \vec{s}$. By renaming variables we can assume that $\text{dom}(\gamma) \cap \text{dom}(\gamma') = \emptyset$, so $\delta = \gamma \cup \gamma'$ is well-defined. Thus

$$t_1 = C[r\delta] = C[\ell'[r]_e\delta] \xrightarrow{\mathcal{R}} s = C[\ell'\delta] \xrightarrow{\mathcal{R}} t_2 = C[r'\delta]$$

Let $\sigma = \text{mgu}(\ell, \ell')$, and χ such that $\delta = \sigma\chi$. The peak above corresponds to the critical peak $r\sigma \leftarrow \ell'\sigma \rightarrow r'\sigma [\varphi\sigma \wedge \varphi'\sigma]$. (If ρ_1 and ρ_2 are renamings of the same rule, then $\ell\sigma = \ell'\sigma$ implies $r\sigma = r'\sigma$ unless $\text{Var}(r) \subseteq \text{Var}(\ell)$ does not hold, so requirement (5) of Definition 5.6.1 is satisfied.) $\square$

The critical peak lemma is valuable both for proving (local) confluence and ground (local) confluence. Here, we focus on ground confluence. The following lemma gives a way to prove this property.

Lemma 5.6.9. Assume given an LCSTRS $\mathcal{L} = (\Sigma, \mathcal{R})$ which is not ground locally confluent, and let $(\succ, \succeq)$ be a bounding pair on $T(\Sigma, \emptyset)$. Then

- (i). There is a $\succ$ -minimal ground term s for which there exist terms t_1, t_2 with $t_1 \xleftarrow{\mathcal{R}} s \xrightarrow{\mathcal{R}} t_2$ but such that there is no t with $t_1 \xrightarrow{\mathcal{R}}^* t \xleftarrow{\mathcal{R}}^* t_2$.
Moreover, there exist a critical peak $v_1 \leftarrow u \rightarrow v_2 [\varphi]$ and ground substitution γ that respects φ such that $s = u\gamma$, $t_1 = v_1\gamma$ and $t_2 = v_2\gamma$.
- (ii). If in addition $\rightarrow_{\mathcal{R}}$ is terminating, then for all ground terms r with $s \succ r$: if there are q_1, q_2 with $q_1 \xleftarrow{\mathcal{R}}^* r \xrightarrow{\mathcal{R}}^* q_2$ then there is a term q such that $q_1 \xrightarrow{\mathcal{R}}^* q \xleftarrow{\mathcal{R}}^* q_2$.

Proof. We will introduce a well-founded ordering $\sqsupset$ that includes $\succ$ and prove a slightly stronger result than strictly required. For (i), we will show the existence of a $\sqsupset$ -minimal

element (rather than a $\succ$ -minimal element). For (ii), we will prove the result for every r satisfying $s \sqsupset r$ (rather than for r satisfying $s > r$). We define $\sqsupset$ as follows:

If $\rightarrow_{\mathcal{R}}$ is terminating, let $\sqsupset$ be the relation with $a \sqsupset b$ if $a > b$ or $a (\rightarrow_{\mathcal{R}}^+ \cdot \succeq) b$ or $a \triangleright b$. If $\rightarrow_{\mathcal{R}}$ is not terminating, then let $\sqsupset$ be the relation with $a \sqsupset b$ if $a > b$ or $a \triangleright b$. Either way, this relation is well-founded, because $a \rightarrow_{\mathcal{R}} b$ and $a \triangleright b$ both imply $a \geq b$, and if $\rightarrow_{\mathcal{R}}$ is terminating then so is the union of $(\rightarrow_{\mathcal{R}}^+ \cdot \succeq)$ and $\triangleright$.

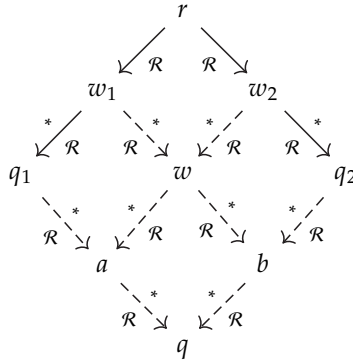
- (i). By definition of not being ground locally confluent, there are ground terms s, t_1, t_2 with $t_1 \xleftarrow{\mathcal{R}} s \xrightarrow{\mathcal{R}} t_2$, for which there is no t such that $t_1 \xrightarrow{\mathcal{R}}^* t \xleftarrow{\mathcal{R}}^* t_2$. Hence, we can take s to be a term with this property that is minimal with respect to $\sqsupset$.

By Lemma 5.6.8 there exist a critical peak $v_1 \leftarrow u \rightarrow v_2 [\varphi]$, context C and substitution γ that respects φ such that $s = C[u\gamma]$, $t_1 = C[v_1\gamma]$ and $t_2 = C[v_2\gamma]$. Since $u\gamma$ is ground – and therefore its reducts $v_1\gamma$ and $v_2\gamma$ as well – necessarily γ is ground on $\text{Var}(u, v_1, v_2)$. Thus we can safely assume that γ is ground overall. Furthermore, $C = \square$ because otherwise $s \sqsupset u\gamma$, contradicting minimality of s .

- (ii). Assume $\rightarrow_{\mathcal{R}}$ is terminating. Let r, q_1, q_2 be ground terms with $s \sqsupset r$ and $q_1 \xleftarrow{\mathcal{R}}^* r \xrightarrow{\mathcal{R}}^* q_2$. By induction on $\rightarrow_{\mathcal{R}}^+$ we will show that there is a term q such that $q_1 \xrightarrow{\mathcal{R}}^* q \xleftarrow{\mathcal{R}}^* q_2$.

If $q_1 = r$ or $q_2 = r$ then there is nothing to prove, so we might assume there are terms w_1, w_2 such that $q_1 \xleftarrow{\mathcal{R}}^* w_1 \xleftarrow{\mathcal{R}} r \xrightarrow{\mathcal{R}} w_2 \xrightarrow{\mathcal{R}}^* q_2$. Since $s \sqsupset r$ and s is a minimal term, there exists a term w such that $w_1 \xrightarrow{\mathcal{R}}^* w \xleftarrow{\mathcal{R}}^* w_2$.

But then $q_1 \xleftarrow{\mathcal{R}}^* w_1 \xrightarrow{\mathcal{R}}^* w$ and $s \sqsupset w_1$ (because $\rightarrow_{\mathcal{R}} \subseteq \succeq$), so by induction hypothesis there is a term a such that $q_1 \xrightarrow{\mathcal{R}}^* a \xleftarrow{\mathcal{R}}^* w$. In a similar way, we can apply the induction hypothesis to find a term b such that $w \xrightarrow{\mathcal{R}}^* b \xleftarrow{\mathcal{R}}^* q_2$. We complete the proof as displayed in the diagram below: since both $w \xrightarrow{\mathcal{R}}^* a$ and $w \xrightarrow{\mathcal{R}}^* b$, one additional application of the induction hypothesis will give us the term q .



□

5.6.2 Ground confluence by Bounded RI

Recall that, following the method in existing literature [8], we aim to show that (1) an LCSTRS is ground confluent when all its critical pairs are bounded ground convertible, and that (2) Bounded RI can be used as a method to prove bounded ground convertibility.

For (1), we introduce the Ground Critical Peak Theorem, analogously to the well-known Critical Pair Theorem [13, Theorem 6.2.4].

Theorem 5.6.10 (Ground Critical Peak Theorem). Let $\mathcal{R}$ be a terminating LCSTRS and let $(>, \geq)$ be a bounding pair. Then $\mathcal{R}$ is ground confluent if for all its critical peaks $v_1 \leftarrow u \rightarrow v_2 [\varphi]$, we have $u > v_1 [\varphi]$, $u > v_2 [\varphi]$, and the equation context $(v_1 ; v_1 \approx v_2 ; v_2) [\varphi]$ is $\mathcal{R}/\emptyset/\emptyset$ -bounded ground convertible using $(>, \geq)$.

Note that $u > v_1 [\varphi]$ and $u > v_2 [\varphi]$ are certainly satisfied if $\rightarrow_{\mathcal{R}} \subseteq >$, but that this is not *required*. This gives a bit more freedom in the choice of the relation $>$.

In Theorem 5.3.21 we showed that Bounded RI proves $\mathcal{R}/\emptyset/\emptyset$ -bounded ground convertibility. Therefore, we immediately obtain (2), as expressed by the following:

Corollary 5.6.11 (Ground confluence by Bounded RI). Let $\mathcal{R}$ be a terminating, quasi-reductive LCSTRS and $(>, \geq)$ a bounding pair such that $u > v_1 [\varphi]$ and $u > v_2 [\varphi]$ for all $v_1 \leftarrow u \rightarrow v_2 [\varphi] \in \text{CP}(\mathcal{R})$. Let $\mathcal{E} = \{(v_1 ; v_1 \approx v_2 ; v_2) [\varphi] \mid v_1 \leftarrow u \rightarrow v_2 [\varphi] \in \text{CP}(\mathcal{R})\}$. If $(\mathcal{E}, \emptyset) \vdash^* (\emptyset, \mathcal{H})$ for some set $\mathcal{H}$, then $\mathcal{R}$ is ground confluent.

Example 5.6.12. In Example 5.2.14 we have $\text{CP}(\mathcal{R}) = \{H f (n-1) m (f x) \leftarrow H f n m x \rightarrow H f (m-1) n (f x) [n > 0 \wedge m > 0]\}$. For $\mathcal{E} = \{(H f (n-1) m (f x) ; H f (n-1) m (f x) \approx H f (m-1) n (f x) ; H f (m-1) n (f x)) [n > 0 \wedge m > 0]\}$, it easily follows that $(\mathcal{E}, \emptyset) \vdash^* (\emptyset, \mathcal{H})$ for some $\mathcal{H}$ and bounding pair with $\rightarrow_{\mathcal{R}} \subseteq >$. So $\mathcal{R}$ is ground confluent.

Towards a proof of Theorem 5.6.10 we introduce two lemmas.

Lemma 5.6.13. Assume $C_1[a_1] = C_2[a_2]$. Then at least one of the following cases holds

- (i). $C_1[\square] = D[\square, a_2]$ and $C_2[\square] = D[a_1, \square]$ for some context $D[\square, \square]$
- (ii). $a_1 = D[a_2]$ for some context $D[\square]$
- (iii). $a_2 = D[a_1]$ for some context $D[\square]$

Proof. By induction on the shape of C_1 . Any term can be written as $f s_1 \cdots s_n$ with $f \in \Sigma \cup \mathcal{V}$ and $n \geq 0$. Therefore, any context $C[\square]$ has one of the following shapes

- (A). $C[\square] = \square s_{i+1} \cdots s_n$, for some $1 \leq i \leq n$
- (B). $C[\square] = f s_1 \cdots s_{i-1} C'[\square] s_{i+1} \cdots s_n$, for some $1 \leq i \leq n$ and context C'

Now, assume $C_1[a_1] = C_2[a_2]$. We consider the four combinations for $C_1[\square]$, $C_2[\square]$.

(AA) $C_1[\square] = \square s_{i+1} \cdots s_n$ and $C_2[\square] = \square s_{j+1} \cdots s_n$

- ▷ If $i = j$ then $C_1 = C_2$ and $a_1 = a_2 = f s_1 \cdots s_i$. Then (ii) holds with $D = \square$.
- ▷ If $i > j$ then $a_1 = f s_1 \cdots s_j \cdots s_i$ and $a_2 = f s_1 \cdots s_j$. Then (ii) holds with $D[\square] = \square s_{j+1} \cdots s_i$ (then $a_1 = D[a_2]$).
- ▷ Similarly, if $i < j$ then (iii) holds by a symmetrical reasoning.

(AB) $C_1[\square] = \square s_{i+1} \cdots s_n$ and $C_2[\square] = f s_1 \cdots s_{j-1} C'[\square] s_{j+1} \cdots s_n$. In particular $a_1 = f s_1 \cdots s_i$ and $C'[a_2] = s_j$.

- ▷ If $i \geq j$ then (ii) holds: $a_1 = D[a_2]$ with $D[\square] = f s_1 \cdots s_{j-1} C'[\square] s_{j+1} \cdots s_i$.
- ▷ If $j > i$ then (i) holds: $D[\square_1, \square_2] = \square_1 s_{i+1} \cdots s_{j-1} C'[\square_2] s_{j+1} \cdots s_n$.

(BA) Symmetrical to **(AB)**.

(BB) $C_1[\square] = f s_1 \cdots s_{i-1} C'_1[\square] s_{i+1} \cdots s_n$ and $C_2[\square] = f s_1 \cdots s_{j-1} C'_2[\square] s_{j+1} \cdots s_n$. In particular $C'_1[a_1] = s_i$ and $C'_2[a_2] = s_j$.

- ▷ If $i = j$ then $C'_1[a_1] = C'_2[a_2]$ and we apply the induction hypothesis; in case (ii) and (iii) we are done, and if the induction hypothesis gives us D' we let $D[\square_1, \square_2] = f s_1 \cdots s_{i-1} D'[\square_1, \square_2] s_{i+1} \cdots s_n$.
- ▷ If $j > i$ then (i) holds: $D[\square_1, \square_2] = f s_1 \cdots s_{i-1} C'_1[\square_1] s_{i+1} \cdots s_{j-1} C'_2[\square_2] s_{j+1} \cdots s_n$. If $i > j$ then (i) holds, using $D[\square_1, \square_2] = f s_1 \cdots s_{j-1} C'_2[\square_2] s_{j+1} \cdots s_{i-1} C'_1[\square_1] s_{i+1} \cdots s_n$. $\square$

In the following, let $u \xrightarrow{s>*}_{\mathcal{R}} v$ if we can write $u = C[a]$, $v = C[b]$, $s > a$, $s > b$, and $a \rightarrow^*_{\mathcal{R}} b$. Let $u \xleftarrow{s>*}_{\mathcal{R}} v$ if $v \xrightarrow{s>*}_{\mathcal{R}} u$. We use $u \xleftrightarrow{s>*}_{\mathcal{R}} v$ if either $u \xrightarrow{s>*}_{\mathcal{R}} v$ or $u \xleftarrow{s>*}_{\mathcal{R}} v$.

Lemma 5.6.14. Let $\mathcal{R}$ be a terminating, non-ground locally confluent LCSTRS and $(>, \geq)$ a bounding pair. Let s be a $>$ -minimal term that exists by Lemma 5.6.9.(i). Suppose $v \xleftarrow{s>*}_{\mathcal{R}} u \xrightarrow{s>*}_{\mathcal{R}} w$. Then there is a term u' such that $v \xrightarrow{s>*}_{\mathcal{R}} u' \xleftarrow{s>*}_{\mathcal{R}} w$.

Proof. There are contexts C_1, C_2 and terms a_1, a_2, b_1, b_2 such that

- ▷ $u = C_1[a_1] = C_2[a_2]$, $v = C_1[b_1]$ and $w = C_2[b_2]$
- ▷ $a_1 \rightarrow^*_{\mathcal{R}} b_1$, $a_2 \rightarrow^*_{\mathcal{R}} b_2$
- ▷ $s > a_1$, $s > b_1$, $s > a_2$, $s > b_2$.

By Lemma 5.6.13 there are three options:

(i). $C_1[\Box] = D[\Box, a_2]$ and $C_2 = D[a_1, \Box]$ for some context $D[\Box, \Box]$. Then

$$D[b_1, a_2] = v \xleftarrow{s >^*_{\mathcal{R}}} u = D[a_1, a_2] \xrightarrow{s >^*_{\mathcal{R}}} w = D[a_1, b_2]$$

We can take $u' = D[b_1, b_2]$. We have $s > b_1$ and $s > b_2$, as required.

(ii). $a_1 = D[a_2]$ for some context $D[\Box]$. Then

$$C_1[b_1] = v \xleftarrow{s >^*_{\mathcal{R}}} u = C_1[a_1] = C_1[D[a_2]] \xrightarrow{s >^*_{\mathcal{R}}} w = C_1[D[b_2]]$$

Since $s > a_1 = D[a_2]$ and $b_1 \xleftarrow{+}_{\mathcal{R}} D[a_2] \xrightarrow{+}_{\mathcal{R}} D[b_2]$ we apply Lemma 5.6.9.(ii) to obtain a term t such that $b_1 \xrightarrow{*}_{\mathcal{R}} t \xleftarrow{*}_{\mathcal{R}} D[b_2]$. We claim that with $u' = C_1[t]$ we obtain the desired result $v = C_1[b_1] \xrightarrow{s >^*_{\mathcal{R}}} C_1[t] = u' \xleftarrow{s >^*_{\mathcal{R}}} C_1[D[b_2]] = w$. The reductions within the context C_1 are already established, so it remains to verify the necessary inequalities $s > b_1, s > t$ and $s > D[b_2]$. The inequality $s > b_1$ holds by assumption. Combining this with $b_1 \xrightarrow{*}_{\mathcal{R}} t$ (and $\rightarrow_{\mathcal{R}} \subseteq \geq$) it follows that $s > t$. For the last inequality, we combine the assumption $s > a_1 = D[a_2]$ with $D[a_2] \rightarrow^*_{\mathcal{R}} D[b_2]$ to conclude $s > D[b_2]$.

(iii). $a_2 = D[a_1]$ for some context $D[\Box]$: similar to (ii). □

With this, we are ready to prove the Ground Critical Peak Theorem.

Proof of Theorem 5.6.10. In a terminating system, ground confluence is implied by ground local confluence. So, towards a contradiction, assume all equation contexts corresponding to critical pairs are bounded ground convertible but that $\mathcal{L}$ is not ground locally confluent.

By Lemma 5.6.9.(i) there is a $>$ -minimal ground term s and terms t_1, t_2 for which

- (1). $t_1 \xleftarrow{\mathcal{R}} s \xrightarrow{\mathcal{R}} t_2$ and there is no t such that $t_1 \xrightarrow{*}_{\mathcal{R}} t \xleftarrow{*}_{\mathcal{R}} t_2$
- (2). there is a critical peak $v_1 \leftarrow u \rightarrow v_2 [\varphi]$ and ground substitution γ that respects φ such that $s = u\gamma, t_1 = v_1\gamma$ and $t_2 = v_2\gamma$

Combining (2) with the assumption that $(v_1 ; v_1 \approx v_2 ; v_2) [\varphi]$ is $\mathcal{R}/\emptyset/\emptyset$ -bounded ground convertible, we have $t_1 \xleftrightarrow{\{t_1, t_2\}}^*_{\mathcal{R}, \emptyset; \emptyset} t_2$. So there exist $c_0, \dots, c_k$ such that $t_1 = c_0$ and $t_2 = c_k$, and $c_i \xleftrightarrow{\{t_1, t_2\}}^*_{\mathcal{R}, \emptyset; \emptyset} c_{i+1}$ for all $0 \leq i < k$. In particular, every such step can only be performed by case (A) of Definition 5.3.5 because $\mathcal{H} = \mathcal{E} = \emptyset$. That means that for all $0 \leq i < k$ we have $c_i = C[\ell\sigma] \leftrightarrow_{\mathcal{R}} C[r\sigma] = c_{i+1}$, for some $\ell \rightarrow r [\varphi] \in \mathcal{R}$ or $r \rightarrow \ell [\varphi] \in \mathcal{R}$, where both $\ell\sigma$ and $r\sigma$ are $\geq$ -dominated by either t_1 or t_2 . By assumption, we also have $u > v_1 [\varphi]$

and $u > v_2 [\varphi]$, which implies $s > t_1$ and $s > t_2$. Since $> \cdot \geq \subseteq >$, we conclude that both $\ell\sigma$ and $r\sigma$ are $>$ -dominated by s . Therefore, $c_i \xleftrightarrow{S>*}_{\mathcal{R}} c_{i+1}$ for all $0 \leq i < k$.

To obtain the contradiction, we show that there is a z such that $t_1 \xrightarrow{*}_{\mathcal{R}} z \xleftarrow{*}_{\mathcal{R}} t_2$. We do so, using induction on k .

- $k = 0$. Then $t_1 = t_2$ and we can choose $z = t_1$.
- $k = 1$. Then $t_1 \rightarrow_{\mathcal{R}}^* t_2$ or $t_2 \rightarrow_{\mathcal{R}}^* t_1$, so it trivially holds.
- $k \geq 2$. We distinguish the following cases:
 - If $t_1 = c_0 \xrightarrow{S>*}_{\mathcal{R}} c_1$ then the induction hypothesis provides a z such that $c_1 \rightarrow_{\mathcal{R}}^* z$ and $t_2 \rightarrow_{\mathcal{R}}^* z$.
 - If $c_{k-1} \xleftarrow{S>*}_{\mathcal{R}} c_k = t_2$ then the induction hypothesis yields z such that $t_1 \rightarrow_{\mathcal{R}}^* z$ and $c_{k-1} \rightarrow_{\mathcal{R}}^* z$.
 - Otherwise, there is an $m > 0$ such that $c_0 \xleftarrow{S>*}_{\mathcal{R}} \cdots \xleftarrow{S>*}_{\mathcal{R}} c_{m-1} \xleftarrow{S>*}_{\mathcal{R}} c_m \xrightarrow{S>*}_{\mathcal{R}} c_{m+1}$. We use a second induction on m . Observe that, by Lemma 5.6.14, there is some c'_m such that $c_{m-1} \xrightarrow{S>*}_{\mathcal{R}} c'_m \xleftarrow{S>*}_{\mathcal{R}} c_{m+1}$. Then the sequence $c_0, \dots, c_{m-1}, c'_m, c_{m+1}, \dots, c_k$ still satisfies the properties above. If $m = 1$, we have $t_1 \rightarrow_{\mathcal{R}}^* c'_1$ and we complete by the first induction hypothesis. If $m > 1$ then we observe that now $c_0 \xleftarrow{S>*}_{\mathcal{R}} \cdots \xleftarrow{S>*}_{\mathcal{R}} c_{m-1} \xrightarrow{S>*}_{\mathcal{R}} c'_m$, so we complete by the second induction hypothesis. $\square$

5.7 Implementation

Bounded RI is implemented in Cora [34] as an interactive proving procedure. Here, the user has to supply the proof steps, while Cora keeps track of all equation contexts, ensures that deduction rules are applied correctly, and searches for a bounding pair. This uses the strategy from Section 5.4.1, maintaining *strongly* bounded equation contexts by viewing each $s \geq t [\psi]$ in Figure 5.1 as $s \geq^! t [\psi]$, and using methods from existing LCSTRS-termination-literature [24] to essentially prove termination of $\text{REQS} \cup \mathcal{R}$.

Validity and satisfiability checks are delegated to an external SMT-solver (by default Z3 [38], but the user can set other preferences), on the theory of (first-order) integer arithmetic (QF_NIA) [14]. For the two highly undecidable derivation rules (Generalize) and (Alter), only limited versions are available to handle the most common cases:

- For (Generalize), Cora allows the user to change the constraint to an implied one, or to supply an equation which instantiates the current proof goal.

- For (Alter), Cora allows the user to supply an equivalent constraint, or add definitions $x = u$ into the constraint with x a fresh variable; in addition, case (II) combined with a (Delete) step can be accessed through the `eq-delete` command. In Section 5.8 we will compare `eq-delete` to the deduction rule (EQ-DELETION) [22].

The use of Cora is illustrated in Figure 5.3. By default, Cora shows only the equation in an equation context, coupled with a red $\odot$ if a side of the equation is the same as its bounding term. (Full equation contexts can be queried using the command `:equations full`.) The deduction rules are implemented through commands like `simplify` and `delete`, and explained by a `:help` command. Termination checks are always executed once the proof is complete, but can also be forced earlier by the user executing `:check`.

While Cora requires user guidance, there is some automation. As shown in Figure 5.3, commands like `simplify` can be called with more or less information: `simplify` without arguments rewrites with an arbitrary rule at some innermost position, but the user can also supply the rule, position and substitution to be used. An `auto` command is supplied to automatically perform `simplify`, `calc`, `delete`, `hdelete`, `eq-delete`, `disprove` and `semiconstructor` steps as far as possible. The use of the `auto` command is illustrated in Figure 5.4.

In future work, we hope to further automate the rewriting induction process. However, a fully automatic process will need to include a powerful lemma generation engine for automatic application of (Postulate) and (Generalize), so this is a large project.

An evaluation page with savefiles that provide full proofs for the inductive theorems in this chapter (including two proofs that will be discussed in Section 5.9) is available at <https://cs.ru.nl/~cynthiakop/experiments/lmcs26/>. (A permanent (frozen) archive of the website content is available at <https://doi.org/10.5281/zenodo.22283693>) It contains the source files, but also a pre-compiled variant, of the Cora version that was used to execute the proofs for the examples in this chapter. We included all proof files, as well as instructions on the installation and use of Cora.

5.8 Comparison to related work

As briefly mentioned in Section 5.2, improvements on the termination requirements for the basic RI system [43] were already introduced in [4–6]. These works also explore ways to have more flexibility in the construction of a well-founded ordering $>$, either fixed beforehand (e.g., the lexicographic path ordering), or constructed during or after the proof. Essentially, this already employs a bounding pair $(>, \geq)$, using a second (milder) ordering $\geq$, to allow for reduction steps with an induction hypothesis to be oriented with $\geq$ rather than the default $>$. However, our approach differs in its use of equation contexts and bounding terms. In our approach, we do not orient induction hypotheses

```

$ cora tailup.lcstrs -e
LCSTRS with only rule scheme Calc:

Signature: recdown :: (Int → Int → Int) → Int → Int → Int → Int
          tailup :: (Int → Int → Int) → Int → Int → Int → Int

Rules: recdown(F, n, i, a) → a | i < n
       recdown(F, n, i, a) → F(i, recdown(F, n, i - 1, a)) | i ≥ n
       tailup(F, i, m, a) → a | i > m
       tailup(F, i, m, a) → tailup(F, i + 1, m, F(i, a)) | i ≤ m

Please input one or more equations, or a file: recdown(F, n, m, a)=tailup(F, n, m, a)
Goal: recdown(F, n, m, a) ≈ tailup(F, n, m, a)

> induct
Goal: recdown(F, n, m, a)@ ≈ tailup(F, n, m, a)@

> case m ≥ n
Goal: recdown(F, n, m, a)@ ≈ tailup(F, n, m, a)@ | ¬(m ≥ n)

> :rules
R1: recdown(F, n, i, a) → a | i < n
R2: recdown(F, n, i, a) → F(i, recdown(F, n, i - 1, a)) | i ≥ n
R3: tailup(F, i, m, a) → a | i > m
R4: tailup(F, i, m, a) → tailup(F, i + 1, m, F(i, a)) | i ≤ m

Goal: recdown(F, n, m, a)@ ≈ tailup(F, n, m, a)@ | ¬(m ≥ n)

> simplify R1 l with [i := m]
Goal: a ≈ tailup(F, n, m, a)@ | ¬(m ≥ n)

> simplify R3 r
Goal: a ≈ a | ¬(m ≥ n)

> delete
Goal: recdown(F, n, m, a)@ ≈ tailup(F, n, m, a)@ | m ≥ n

> simplify
Goal: F(m, recdown(F, n, m - 1, a)) ≈ tailup(F, n, m, a)@ | m ≥ n

> simplify
Goal: F(m, recdown(F, n, m - 1, a)) ≈ tailup(F, n + 1, m, F(n, a)) | m ≥ n

> calc
Goal: F(m, recdown(F, n, m1, a)) ≈ tailup(F, n1, m, F(n, a)) | m ≥ n ∧ m1 = m - 1 ∧ n1 = n + 1

> hypothesis H1 l2
Goal: F(m, tailup(F, n, m1, a)) ≈ tailup(F, n1, m, F(n, a)) | m ≥ n ∧ m1 = m - 1 ∧ n1 = n + 1

> induct
Goal: F(m, tailup(F, n, m1, a))@ ≈ tailup(F, n1, m, F(n, a))@ | m ≥ n ∧ m1 = m - 1 ∧ n1 = n + 1

```

Fig. 5.3 Basic usage of Cora

```

Please input one or more equations, or a file: recdown(F, n, m, a)=tailup(F, n, m, a)
Goal: recdown(F, n, m, a) ≈ tailup(F, n, m, a)

> induct
Goal: recdown(F, n, m, a)@ ≈ tailup(F, n, m, a)@

> case m >= n
Goal: recdown(F, n, m, a)@ ≈ tailup(F, n, m, a)@ | ¬(m ≥ n)

> auto
  simplify R1 l with [F := F, a := a, i := m, n := n]
  simplify R3 r with [F := F, a := a, i := n, m := m]
  delete

Goal: recdown(F, n, m, a)@ ≈ tailup(F, n, m, a)@ | m ≥ n

> auto
  simplify R2 l with [F := F, a := a, i := m, n := n]
  calc l2.3
  simplify R4 r with [F := F, a := a, i := n, m := m]
  calc r2

Goal: F(m, recdown(F, n, m1, a)) ≈ tailup(F, n1, m, F(n, a)) | m ≥ n ∧ m1 = m - 1 ∧ n1 = n + 1

> hypothesis
Goal: F(m, tailup(F, n, m1, a)) ≈ tailup(F, n1, m, F(n, a)) | m ≥ n ∧ m1 = m - 1 ∧ n1 = n + 1

> induct
Goal: F(m, tailup(F, n, m1, a))@ ≈ tailup(F, n1, m, F(n, a))@ | m ≥ n ∧ m1 = m - 1 ∧ n1 = n + 1

> case m1 >= n
Goal: F(m, tailup(F, n, m1, a))@ ≈ tailup(F, n1, m, F(n, a))@ | m ≥ n ∧ m1 = m - 1 ∧ n1 = n + 1 ∧ ¬(m1 ≥ n)

> auto
  simplify R3 l2 with [F := F, a := a, i := n, m := m1]
  simplify R3 r with [F := F, a := F(n, a), i := n1, m := m]
  eq-delete

Goal: F(m, tailup(F, n, m1, a))@ ≈ tailup(F, n1, m, F(n, a))@ | m ≥ n ∧ m1 = m - 1 ∧ n1 = n + 1 ∧ m1 ≥ n

> auto
  simplify R4 l2 with [F := F, a := a, i := n, m := m1]
  calc l2.2
  simplify R4 r with [F := F, a := F(n, a), i := n1, m := m]
  calc r2
  hdelete H11 l with [F := F, a := F(n, a), m := m, m1 := m1, n := n1, n1 := n2]

All goals have been successfully removed!

To complete the proof, you still need to verify that a suitable ordering exists. You can do this using the
:check command. Other useful environment commands include:

:undo -- to go back to the previous proof state
:quit -- to end the proof process without termination check
:ordering -- to see the current ordering requirements

```

Fig. 5.4 Using the auto command in Cora

themselves: we only require that a particular instance of the induction hypothesis is strictly dominated by the bounding terms of the equation context under consideration. Another notable difference is that their approach imposes more bureaucracy, since derivation rules rely on several steps being done at once, by reasoning *modulo* a set of induction hypotheses. This makes it quite hard to use especially when the relation $>$ is not fixed beforehand but constructed on the fly. Finally, unlike their approach, we do not impose a ground totality requirement. This allows, for instance, to choose $> = (\rightarrow_A \cup \triangleright)^+$, or to use a construction based on dependency pairs (see Section 5.4). This is particularly important in higher-order rewriting, where very few ground-total orderings exist.

(Expansion) Many existing RI systems in the literature, including HORI, contain a deduction rule (Expansion). However, from the perspective of transparency, the (Expansion) rule is not ideal, since it combines many conceptual steps simultaneously. This motivated us to decompose the rule, making the individual proof steps explicit, which in turn naturally led to the introduction of two new deduction rules: (Induct) and (Case). In the setting of Bounded RI, the (Expansion) rule from HORI can be viewed as the consecutive execution of three deduction rules:

- (1). (Induct). Given a proof state $\mathcal{P}_0 = (\mathcal{E} \uplus \{(\zeta ; s \approx t ; \tau) [\psi]\}, \mathcal{H})$ we apply (Induct) to obtain $\mathcal{P}_1 = (\mathcal{E} \cup \{(s ; s \approx t ; t) [\psi]\}, \mathcal{H} \cup \{s \approx t [\psi]\})$.

We recall that, in HORI, $\mathcal{H}$ contains oriented equations and $\mathcal{R} \cup \mathcal{H}$ is required to be terminating. In Bounded RI we can model this by choosing one side of the equation where we will apply (Expansion), and orienting the equation in that direction. For example, if we choose the left-hand side s , then we require $s > t [\psi]$. (In this setting, we let $\geq$ be the reflexive closure of $>$ and impose $\rightarrow_{\mathcal{R}} \subseteq \geq$.)

- (2). (Case). As explained in Chapter 3, applying (Expansion) requires a position p in s such that $s|_p = f s_1 \cdots s_n$ with $f \in \mathcal{D}$, $n \geq ar(f)$, and such that s_i is a semi-constructor term for all $1 \leq i \leq ar(f)$. With these ingredients we define a cover set as follows:

$$C_{\text{EXP}} = \left\{ (\delta, \varphi\delta) \left| \begin{array}{l} \ell \rightarrow r [\varphi] \in \mathcal{R}, \delta = \text{mgu}(f s_1 \cdots s_k, \ell), \\ \delta(\text{Var}(\varphi) \cup \text{Var}(\psi)) \subseteq \mathcal{Val} \cup \mathcal{V} \end{array} \right. \right\}$$

Due to the assumption of quasi-reductivity, this really is a cover set. We apply (Case) to $\mathcal{P}_1$, using C_{EXP} to obtain the following proof state

$$\mathcal{P}_2 = (\mathcal{E} \cup \{(s\delta ; s\delta \approx t\delta ; t\delta) [\psi\delta \wedge \varphi] \mid (\delta, \varphi) \in C_{\text{EXP}}\}, \mathcal{H} \cup \{s \approx t [\psi]\})$$

- (3). (Simplify) on each of the equations obtained in (2), using the corresponding rule $\ell \rightarrow r [\varphi] \in \mathcal{R}$. Due to the choice of cover set, the conditions of (Simplify) are

satisfied. This yields $\mathcal{P}_3 = (\mathcal{E} \cup \mathcal{E}_{\text{EXP}}^{\rightarrow \mathcal{R}}, \mathcal{H} \cup \{s \approx t [\psi]\})$ with $\mathcal{E}_{\text{EXP}}^{\rightarrow \mathcal{R}} =$

$$\left\{ (s\delta ; s[r \ s_{k+1} \cdots s_n]_p \delta \approx t\delta ; t\delta) [(\psi\delta) \wedge (\varphi\delta)] \left| \begin{array}{l} \ell \rightarrow r [\varphi] \in \mathcal{R}, \\ \delta = \text{mgu}(\text{f } s_1 \cdots s_k, \ell), \\ \delta(\text{Var}(\varphi, \psi) \cup \text{Var}(\psi)) \subseteq \mathcal{Val} \cup \mathcal{V} \end{array} \right. \right\}$$

Hence, using multiple steps, we regain the (Expansion) rule in Bounded RI, and improve on it by not requiring $s > t [\psi]$. Instead, a typically weaker requirement is imposed when applying induction hypothesis $s \approx t [\psi]$ in (Hypothesis) or ($\mathcal{H}$ -Delete).

(EQ-DELETION) In Section 5.7 we mentioned the presence of a command `eq-delete` in Cora. This deduction rule allows the user to remove an equation from the current proof state and is a special use case of (EQ-DELETION) [22]:

$$\frac{(\mathcal{E} \uplus \{C[s_1, \dots, s_n] \approx C[t_1, \dots, t_n] [\varphi]\}, \mathcal{H})}{(\mathcal{E} \cup \{C[s_1, \dots, s_n] \approx C[t_1, \dots, t_n] [\varphi \wedge \neg(\wedge_{i=1}^n s_i = t_i)]\}, \mathcal{H})} \quad \begin{array}{l} s_i, t_i \in T(\Sigma_{\text{theory}}, \text{Var}(\varphi)) \\ \text{for all } 1 \leq i \leq n \end{array}$$

We can obtain the above deduction rule in Bounded RI, starting with (Case) to split $C[s_1, \dots, s_n] \approx C[t_1, \dots, t_n] [\varphi]$ into two equations: one where $\wedge_{i=1}^n s_i = t_i$ holds and one where $\neg(\wedge_{i=1}^n s_i = t_i)$ holds. Then:

- (1). the case $\wedge_{i=1}^n s_i = t_i$ is followed by (Alter) to obtain the equation $C[s_1, \dots, s_n] \approx C[s_1, \dots, s_n] [\varphi \wedge (\wedge_{i=1}^n s_i = t_i)]$, which is then removed by (Delete).
- (2). the case $\neg(\wedge_{i=1}^n s_i = t_i)$ yields the equation $C[s_1, \dots, s_n] \approx C[t_1, \dots, t_n] [\varphi \wedge \neg(\wedge_{i=1}^n s_i = t_i)]$, exactly as in the (EQ-DELETION) rule.

However, in Cora, the use case of the `eq-delete` command is narrowed down to situations where $\varphi \implies \wedge_{i=1}^n s_i = t_i$ holds, in which case $\varphi \wedge \neg(\wedge_{i=1}^n s_i = t_i)$ is unsatisfiable so that the equation in (2) can also be removed by (Delete).

5.9 Conclusion and future work

With the introduction of Bounded RI, we revised the existing RI system for LCSTRSs, making fundamental changes with the aim of reducing termination requirements. We replaced a well-founded order $>$ or terminating relation $\rightarrow_{\mathcal{R} \cup \mathcal{H}}$ with a *bounding pair* $(>, \geq)$ allowing for less strict ordering requirements with $\geq$. We replaced equations by *equation contexts*, which contain *bounding terms*, keeping track of induction bounds efficiently. As a byproduct, we obtained a more intuitive proof system with a higher degree of modularity, which makes it easy to add new deduction rules to the system.

Easier induction proofs In Bounded RI, we deviate from HORI by not requiring the induction hypotheses themselves to be oriented. In particular, the ($\mathcal{H}$ -Delete) rule is beneficial, as it often allows us to complete a proof with minimal termination requirements (e.g., requirements that are immediately satisfied if $\rightarrow_{\mathcal{R}}$ is included in $>$). This often allows us to entirely avoid challenges we encountered in HORI.

- (1). Consider the equivalence $\text{sumR } x \approx \text{suml } x$, defined by the following LCSTRS

$$\begin{array}{lll} & \text{suml } x \rightarrow v \ x \ 0 & \\ \text{sumR } x \rightarrow x + (\text{sumR } (x - 1)) & [x > 0] & v \ x \ a \rightarrow v \ (x - 1) \ (a + x) \ [x > 0] \\ \text{sumR } x \rightarrow 0 & [x \leq 0] & v \ x \ a \rightarrow a \ [x \leq 0] \end{array}$$

With matrix invariants we can generate $(\mathbf{L}) \ x + (v \ y \ z) \approx v \ y \ a \ [a = x + z]$, which in Bounded RI is sufficient to use as a lemma. In HORI, however, this is not a lemma, because neither $v \ y \ a \rightarrow x + (v \ y \ z) \ [a = x + z]$ nor $x + (v \ y \ z) \rightarrow v \ y \ a \ [a = x + z]$ is terminating. Although we can use the techniques from Section 4.3.2 and 4.3.3 to compute additional inequality-invariants $x > 0$ and $z \geq 0$ (to obtain a terminating induction rule), this is not necessary in Bounded RI: we can directly use $(\mathbf{L})$ as induction hypothesis, since ($\mathcal{H}$ -Delete) imposes lower requirements.

- (2). In Chapter 3 we proved $\text{sumfun } f \ n \approx \text{fold } [+] \ 0 \ (\text{map } f \ (\text{init } n)) \ [n \geq 0]$ in

$$\begin{array}{ll} \text{fold } g \ v \ \text{nil} \rightarrow v & \text{map } f \ \text{nil} \rightarrow \text{nil} \\ \text{fold } g \ v \ (h : t) \rightarrow \text{fold } g \ (g \ v \ h) \ t & \text{map } f \ (h : t) \rightarrow (f \ h) : \text{map } f \ t \\ \text{init } n \rightarrow \text{nil} \ [n < 0] & \text{init } n \rightarrow n : \text{init } (n - 1) \ [n \geq 0] \end{array}$$

The proof required the induction rule $x + (\text{fold } [+] \ y \ l) \rightarrow \text{fold } [+] \ z \ l \ [z = x + y]$ which is non-standard, as it has a theory symbol $+$ as root symbol on the left. Termination *can* be proven, but requires a very advanced method that is not easy to use automatically. In practice, we would like to avoid such induction rules as much as possible. Again, Bounded RI provides a solution, since the application of ($\mathcal{H}$ -Delete) with induction hypotheses $x + (\text{fold } [+] \ y \ l) \approx \text{fold } [+] \ z \ l \ [z = x + y]$ does not impose such a requirement.

Ground confluence Since Bounded RI proves *bounded* ground convertibility, rather than merely ground convertibility, we can use the system to prove ground confluence: we defined critical pairs for LCSTRSs and showed that a terminating LCSTRS is ground confluent if all its critical peaks are bounded ground convertible. Finally, as a natural application of this result, we showed that for ground confluent LCSTRSs, Bounded RI can be extended into a system for disproving inductive theorems.

We conclude with some practically motivated directions for future research.

Global rewriting induction An interesting question is whether we can extend Global RI from Section 3.3.4 to define Global Bounded RI to prove global bounded inductive theorems. The major challenge in this will be to guarantee that a bounding pair $(>, \geq)$ on $T(\Sigma, \mathcal{V})$ can be extended to a bounding pair $(>', \geq')$ on $T(\Sigma', \mathcal{V}')$, for any natural extension $\mathcal{L}' = (\mathcal{R}', \Sigma')$ of $\mathcal{L} = (\mathcal{R}, \Sigma)$. In particular, we will have to reconsider every deduction rule in Figure 5.1 that has an ordering requirement, including (Hypothesis), ($\mathcal{H}$ -Delete), (Generalize) and (Alter).

Ground convertibility modulo axioms In practice, we encounter inductive theorems which are only provable once we are allowed to utilize our knowledge about the instantiation of certain variables. For example, an equivalence $P_1 f \approx P_2 f$ may hold only under the assumption that $f :: \text{int} \rightarrow \text{int} \rightarrow \text{int}$ is instantiated by a commutative function. So instead of convertibility under *all* ground substitutions, we restrict to those ground substitutions that instantiate f as a commutative function.

Designing a corresponding RI system is non-trivial because this new notion of equivalence is no longer based on the standard notion of ground convertibility, but on a restricted form of it. This likely brings us into a setting similar to term rewriting modulo a set $\mathcal{A}$ of axioms. A natural first step is therefore to examine how the soundness proofs from Section 5.3 can be adapted to rewriting modulo $\mathcal{A}$.

Automating RI proofs Currently, any proof by Bounded RI is (almost) completely human driven. Ideally, it should incorporate some tactics in Cora for automatically proving inductive theorems in Bounded RI, building on the strategies being implemented in Ctrl [22] for first-order LCTRSs. A major obstacle in this is finding lemmas automatically. Here, we can benefit from heuristics in Chapter 4 and 6, but it remains a challenge to determine how to apply them effectively. As program equivalence in general is undecidable, we cannot expect heuristic based approaches to be fully exhaustive.

Unbounded RI The definition of Bounded RI takes advantages of the restriction that all equation contexts are bounded (and remain so because our derivation rules have the Preserving Bounds property). However, this restriction is not fundamental: as shown by the proofs in Section 5.3, what matters most is that the necessary multiset ordering conditions are imposed. To obtain these conditions we currently use the property that equation contexts are bounded, but we could alternatively require it directly in the derivation rules. This would for instance replace the (Hypothesis) rule by

$$\frac{(\mathcal{E} \uplus \{(\zeta ; C[\ell\delta] \approx t ; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \{(\zeta ; C[r\delta] \approx t ; \tau) [\psi]\}, \mathcal{H})} \quad \begin{array}{l} \ell \approx r [\varphi] \in \mathcal{H} \text{ and } \psi \models^\delta \varphi \text{ and} \\ \{(\zeta, \tau)\} \succ_{mul} \{(\ell\delta, r\delta)\} [\psi] \end{array}$$

And (Induct) would become:

$$\frac{(\mathcal{E} \uplus \{(\varsigma ; s \approx t ; \tau) [\psi]\}, \mathcal{H})}{(\mathcal{E} \cup \{(s ; s \approx t ; t) [\psi]\}, \mathcal{H} \cup \{s \approx t [\psi]\})} \quad \{\varsigma, \tau\} \succeq_{mul} \{\{s, t\} [\psi]\}$$

On the upside, we could then weaken the restrictions for (Alter)/(Generalize), since it is no longer necessary to preserve bounds. We have checked that the current proofs easily transfer to this altered system for deriving ground convertibility, though not for deriving *bounded* ground convertibility.

The advantage of this change is that we can further weaken the ordering requirements: instead of requiring that $\rightarrow_{\mathcal{R}} \subseteq \succeq$ it would suffice if $s \succeq s \downarrow_{\mathcal{R}}$ for all ground terms s . However, a downside is that it is not obviously usable for proving ground confluence.

The most important complication is that six different combinations of $>$ and $\succeq$ are admissible when realizing a multiset ordering. This is not a problem if the bounding pair $(>, \succeq)$ is fixed in advance, but *is* challenging if the search for a bounding pair is done afterwards. Naïvely exploring the entire search space – i.e. considering all options at every deduction step that imposes requirements on the ordering – would lead to exponential growth in the length of the deduction sequence. From this perspective, Bounded RI can be seen as a heuristic that provides guidance on which ordering requirements to impose. In future work, it would be interesting to further explore Unbounded RI and determine if there are strategies to take advantage of the greater generality, and still find a bounding pair easily.

Chapter 6

Program Equivalence by Templates

In Chapter 4 we introduced several invariant generation heuristics to generate lemmas for individual simple divergence patterns. In this chapter, we take a different approach: using a template-based strategy, we can describe whole classes of equivalences at once. A template is a rewriting scheme that aims to generalize a particular programming construction, such as a tail-recursive procedure. Templates for many typical programming constructs can be represented by an LCSTRS. With Bounded RI, we may prove equivalence between such LCSTRSs in general, after which they can be instantiated to prove program equivalence in practice. While templates have been used as a stand-alone method to justify program transformations, we here consider their integration in Bounded RI to obtain a complementary lemma generation heuristic. This allows us to prove equivalences for which the heuristics from Chapter 4 fail.

Motivating example We consider four implementations of the factorial function in Figure 6.1, and represent them as Logically Constrained Simply-typed Term Rewriting Systems in Figure 6.2. These implementations do not necessarily agree on inputs $x < 1$, but if we restrict to $x \geq 1$ they all compute $x \mapsto \prod_{i=1}^x i$.

With the heuristics from Chapter 4 we can only prove $\text{factTU } x \approx \text{factRD } x [x \geq 1]$ and $\text{factTD } x \approx \text{factRD } x$. The equivalence $\text{factTU } x \approx \text{factRU } x [x \geq 1]$ can also be proven, but only through a non-trivial intervention, transforming a non-simple divergence pattern into a simple divergence pattern, as required for the computation of matrix invariants. The remaining equivalences $\text{factTD } x \approx \text{factRU } x [x \geq 1]$, $\text{factTU } x \approx \text{factTD } x$ and $\text{factRU } x \approx \text{factRD } x [x \geq 1]$ remain out of reach. This can be explained by the fact that, intuitively, invariant-generation corresponds to generalization at the level of proof processes, whereas these equivalences require generalization at the level of programming constructs. This is because the corresponding divergence patterns exhibit computational behavior beyond the expressiveness of polynomial invariants. For example, in Section 4.7 we discussed that, although the proof of $\text{factTU } x \approx \text{factTD } x$ yields a simple divergence

```

int factTU(int x){
int a = 1; int i = 1;
while (i<=x){
    a = i*a; i = i+1;}
return a; }

```

(a) iterative upward

```

int factTD(int x){
int a = 1;
while (x>0){
    a = a*x; x = x-1;}
return a; }

```

(b) iterative downward

```

int factRD(int x){
if (x > 0)
    return(x*factRD(x-1));
else
    return 1; }

```

(c) recursive downward

```

int factRU(int x) = R(1, x);
int R(int i, int x){
if (i<x)
    return(i*R(i+1, x));
else
    return x; }

```

(d) recursive upward

Fig. 6.1 Four implementations of the factorial function.

$\text{factTU } x \rightarrow u \ x \ 1 \ 1$ $u \ x \ i \ a \rightarrow a \quad [i > x]$ $u \ x \ i \ a \rightarrow u \ x \ (i+1) \ (i * a) \quad [i \leq x]$

(a) Tail recursive Upward (TU)

$\text{factTD } x \rightarrow d \ x \ 1$ $d \ x \ a \rightarrow a \quad [x \leq 0]$ $d \ x \ a \rightarrow d \ (x-1) \ (a * x) \quad [x > 0]$

(b) Tail recursive Downward (TD)

$\text{factRD } x \rightarrow 1 \quad [x \leq 1]$ $\text{factRD } x \rightarrow x * (\text{factRD } (x-1)) \quad [x > 1]$
--

(c) Recursive Downward (RD)

$\text{factRU } x \rightarrow U \ 1 \ x$ $U \ i \ x \rightarrow x \quad [i > x-1]$ $U \ i \ x \rightarrow i * (U \ (i+1) \ x) \quad [i \leq x-1]$

(d) Recursive Upward (RU)

Fig. 6.2 LCSTRS translations of the implementations in Figure 6.1

pattern, we are not able to generate a corresponding lemma, as this would require the non-polynomial invariants $a = \prod_{k=1}^{i-1} k$ and $b = \prod_{k=0}^{i-2} (x-k)$. Fortunately, this does not mean we cannot prove this equivalence. The key is to adopt a more high-level strategy: using templates we can generalize over loop structures themselves, instead of the specific values of loop variables. With this approach we can easily prove all six equivalences without complicated invariants.

Templates Inspired by an existing work on program transformations [19], we consider templates: rewrite schemes that capture commonly occurring programming patterns, such as upward or downward tail recursion. Templates are modeled by higher-order recursors; we prove equivalences between recursors, which can then be applied to prove concrete instances. This provides a high-level lemma generation method,

complementary to existing low-level heuristics from Chapter 4. With templates, proving $\text{factTU } x \approx \text{factTD } x$ becomes equally easy as proving $\text{sumTU } x \approx \text{sumTD } x$, where this last equivalence involves a TailUp and TailDown implementation of $x \mapsto \sum_{i=1}^x i$. While the heuristics in Chapter 4 can brute-force a lemma for $\text{sumTU } x \approx \text{sumTD } x$, we just argued that this fails for $\text{factTU } x \approx \text{factTD } x$, due to the required non-polynomial invariants.

Contributions Templates in rewriting are not new [19], but the combination with LCSTRSs opens up possibilities for integration in the formal RI system in a manner very useful for real program verification:

- Since LCSTRS are a form of higher-order rewriting, we can perform all recursor-equivalence proofs within RI, rather than at the meta-level. This allows templates to be integrated in tools as a lemma generation method, complementary to methods that focus on lower-level behavior.
- Since constrained rewriting directly supports primitive data structures, the rewrite rules closely resemble real programming code. This in contrast to the required encodings in unconstrained rewriting, which make analysis of loop constructions like in Figure 6.1 infeasible or at least inconvenient. With LCSTRSs we can define suitable templates in a human-readable manner.

Overview The structure of this chapter is as follows:

- In Section 6.1 we introduce four templates and describe how to match them with concrete programs.
- In Section 6.2 we assume recursor equivalences as given and focus on the proof-tactical aspect of how to apply these equivalences to prove concrete template instances. We distinguish two proof tactics: *one-sided matching*, transforming a proof goal into an easier one, and *two-sided matching*, concluding equivalence directly. This section is meant to provide an intuitive explanation.
- In Section 6.3 we show in a more technical way how templates are used within Bounded RI as a lemma generation method. We discuss the corresponding ordering requirements that constitute the separate parts of template-based proofs, and describe how to recombine them in concrete instances.

To facilitate future semi-automatic application of the template method in Cora and to minimize corresponding termination requirements, we aim to integrate templates into the Bounded RI system described in Chapter 5. For the sake of clarity, however, we introduce the template method outside the specific framework of Bounded RI in

an intuitive and self-contained manner. To this end, we first recall the dependency on bounded ground convertibility: in Definition 5.3.5 we defined $\mathcal{R}/\mathcal{H}/\mathcal{E}$ -bounded convertibility as part of the technical setup for proving soundness of Bounded RI, where the notion is inherently tied to a specific proof state. However, for the treatment of the template method it is more convenient to have bounded ground convertibility as an independent definition, yielding a self-contained notion of *bounded inductive theorems*.

Definition 6.0.1 (Bounded Inductive Theorem). For terms ς, τ we say that $s \approx t [\varphi]$ is $\mathcal{R}$ -bounded ground convertible under ς, τ , denoted $s \xleftrightarrow{\varsigma, \tau}^*_{\mathcal{R}} t [\varphi]$, if $s = t$ or for every ground substitution γ that respects φ there exist ground contexts $C_0, \dots, C_n$ and ground terms $s_0, \dots, s_n, t_0, t_1, \dots, t_n$ so that $s\gamma = C_0[s_0]$, $t\gamma = C_n[t_n]$, and for all $0 \leq i \leq n$: (a) $s_i \leftrightarrow_{\mathcal{R}} t_i$, (b) $\varsigma\gamma \geq s_i$ or $\tau\gamma \geq s_i$, (c) $\varsigma\gamma \geq t_i$ or $\tau\gamma \geq t_i$, (d) if $i > 0$: $C_{i-1}[t_{i-1}] = C_i[s_i]$. (So $s\gamma = C_0[s_0] \leftrightarrow_{\mathcal{R}} C_0[t_0] = C_1[s_1] \leftrightarrow_{\mathcal{R}} \dots \leftrightarrow_{\mathcal{R}} C_n[t_n] = t\gamma$, and each s_i, t_i is dominated by at least one of $\varsigma\gamma$ or $\tau\gamma$.) We say $s \approx t [\varphi]$ is a *bounded inductive theorem* if $s \xleftrightarrow{\varsigma, \tau}^*_{\mathcal{R}} t [\varphi]$.

6.1 Templates

A template is a rewrite scheme that captures a particular programming construct. Templates are expressed using context functions.

Definition 6.1.1 (Context function). A *context function* in n parameters is a term $C(\square_1, \dots, \square_n)$ (or just: C) in $T(\Sigma \cup \{\square_1, \dots, \square_n\}, \mathcal{V})$ where each $\square_i$ may occur 0 or more times. The term obtained from C by replacing each $\square_i$ by a term t_i of the same type (multiple instances of the same $\square_i$ are replaced by the same t_i) is denoted $C(t_1, \dots, t_n)$.

Table 6.1 presents four templates that generalize the constructs in Figure 6.1, considering recursion and tail recursion on integers in either upward or downward direction. The templates are defined by parametrized contexts, whose parameters include a context function $F(\square_1, \square_2) :: \text{int}$ (used in the recursive call), and terms $l :: \text{int}$ or $u :: \text{int}$ (used as lower or upper bound), as well as possibly a term $b :: \text{int}$ (used as a base case). The symbols T and R indicate tail-recursion and general recursion, while the superscripts $+$ and $-$ indicate upward and downward direction, respectively. More precisely

- $T_{F,u}^+[\square_1, \square_2], T_{F,l}^-[\square_1, \square_2] :: \text{int}$ are contexts with $\square_1, \square_2 :: \text{int}$.
- $R_{F,u,b}^+[\square], R_{F,l,b}^-[\square] :: \text{int}$ are contexts with $\square :: \text{int}$.
- $F(\square_1, \square_2) :: \text{int}$ is a context function with $\square_1, \square_2 :: \text{int}$, while $l, u, b :: \text{int}$ are terms.

Moreover, the template structure employs one or two integer *variables*: $i :: \text{int}$ and $a :: \text{int}$. These variables may not occur inside the other terms and contexts. The variable i serves as a loop index, which is decreased or increased by 1 at each recursive call, until

Table 6.1 Templates for upward/downward recursion and tail recursion

Construct	Template	Computed term
TailUp	$T_{F,u}^+[i, a] \rightarrow a \quad [i > u]$	$F(u, F(u-1, \dots, F(i+1, F(i, a)) \dots))$
	$T_{F,u}^+[i, a] \rightarrow T_{F,u}^+[i+1, F(i, a)] \quad [i \leq u]$	
TailDown	$T_{F,l}^-[i, a] \rightarrow a \quad [i < l]$	$F(F(\dots F(F(a, i), i-1) \dots, l+1), l)$
	$T_{F,l}^-[i, a] \rightarrow T_{F,l}^-[i-1, F(a, i)] \quad [i \geq l]$	
RecUp	$R_{F,u,b}^+[i] \rightarrow b \quad [i > u]$	$F(F(\dots F(F(b, u), u-1) \dots, i+1), i)$
	$R_{F,u,b}^+[i] \rightarrow F(R_{F,u,b}^+[i+1], i) \quad [i \leq u]$	
RecDown	$R_{F,l,b}^-[i] \rightarrow b \quad [i < l]$	$F(i, F(i-1, \dots, F(l+1, F(l, b)) \dots))$
	$R_{F,l,b}^-[i] \rightarrow F(i, R_{F,l,b}^-[i-1]) \quad [i \geq l]$	

i is below the lower bound l or above the upper bound u . At that point, the computation terminates and returns either the base case b , or the variable a which serves as an *accumulator*. The last column of Table 6.1 displays a schematic presentation of the term being computed during this procedure.

Example 6.1.2. We recall the factorial systems from Figure 6.2 and demonstrate how to match them with the corresponding templates.

For Figure 6.2a choose $F(\square_1, \square_2) := \square_1 * \square_2$, $u := x$ and $T_{F,u}^+[\square_1, \square_2] := u \ x \ \square_1 \ \square_2$. Similarly, for Figure 6.2d choose $F(\square_1, \square_2) := \square_2 * \square_1$, $u := x-1$, $b := x$ and $R_{F,u,b}^+[\square] := U \ \square \ x$. These cases were easy to deal with because the syntactic shape prescribed by each of the rules for u and U exactly matches that of its corresponding template.

Figure 6.2b and Figure 6.2c are more complicated due to the syntactic mismatch in the constraints. However, since we are dealing with integers, this is solved by observing that $i \leq n$ is equivalent to $i < n+1$ and $i > n$ equivalent to $i \geq n+1$ (such transformations of integer inequalities can be implemented as part of an automatic pre-processing step). Hence, we can pre-process these two LCSTRSs to obtain equivalent systems

factTD $x \rightarrow d \ x \ 1$	
$d \ x \ a \rightarrow a \quad [x < 1]$	
$d \ x \ a \rightarrow d \ (x-1) \ (a * x) \quad [x \geq 1]$	
	factRD $x \rightarrow 1 \quad [x < 2]$
	factRD $x \rightarrow x * (\text{factRD } (x-1)) \quad [x \geq 2]$

In this representation, we match Figure 6.2b using $F(\square_1, \square_2) := \square_1 * \square_2$, $i := x$, $l := 1$ and $T_{F,l}^+[\square_1, \square_2] := d \ \square_1 \ \square_2$. We match Figure 6.2c using $F(\square_1, \square_2) := \square_1 * \square_2$, $i := x$, $l := 2$, $b := 1$ and $R_{F,l,b}^+[\square] := \text{factRD } \square$.

6.1.1 Recursors

To generalize the template shapes, we introduce four corresponding higher-order recursors. Define $\mathcal{R}_{\text{rec}}$ as the set containing the following rewrite rules

$$\begin{array}{ll}
 (\tau_b^\uparrow) \quad \text{TU } f \ i \ y \ a \rightarrow a \ [i > y] & (\tau_r^\uparrow) \quad \text{TU } f \ i \ y \ a \rightarrow \text{TU } f \ (i + 1) \ y \ (f \ i \ a) \ [i \leq y] \\
 (\tau_b^\downarrow) \quad \text{TD } f \ x \ i \ a \rightarrow a \ [i < x] & (\tau_r^\downarrow) \quad \text{TD } f \ x \ i \ a \rightarrow \text{TD } f \ x \ (i - 1) \ (f \ a \ i) \ [i \geq x] \\
 (\rho_b^\uparrow) \quad \text{RU } f \ i \ y \ z \rightarrow z \ [i > y] & (\rho_r^\uparrow) \quad \text{RU } f \ i \ y \ z \rightarrow f \ (\text{RU } f \ (i + 1) \ y \ z) \ i \ [i \leq y] \\
 (\rho_b^\downarrow) \quad \text{RD } f \ x \ i \ z \rightarrow z \ [i < x] & (\rho_r^\downarrow) \quad \text{RD } f \ x \ i \ z \rightarrow f \ i \ (\text{RD } f \ x \ (i - 1) \ z) \ [i \geq x]
 \end{array}$$

With Bounded RI, it is trivially easy to prove for instance that $u \ y \ i \ a \approx \text{TU}[*] \ i \ y \ a \ [\text{true}]$ is an inductive theorem. More generally, we have the following result:

Lemma 6.1.3 (Template-recursor equivalences). Consider a quasi-reductive LCSTRS with rules $\mathcal{R}$, and a well-founded ordering $>$ that includes $\rightarrow_{\mathcal{R}} \cup \triangleright$. Assume given a context function F and terms u, l, b (in which the variables i, a do not occur), and suppose that $\mathcal{R}$ contains a rule $\underline{F} \ x \ y \rightarrow F(x, y)$. Then:

- ▷ if $\mathcal{R}$ includes $(\tau_b^\uparrow), (\tau_r^\uparrow)$ and the rules defining TailUp for some context $T_{F,u}^+$, then $T_{F,u}^+[i, a] \approx \text{TU } \underline{F} \ i \ u \ a \ [\text{true}]$ is a bounded inductive theorem;
- ▷ if $\mathcal{R}$ includes $(\tau_b^\downarrow), (\tau_r^\downarrow)$ and the rules defining TailDown for some context $T_{F,l}^-$, then $T_{F,l}^-[i, a] \approx \text{TD } \underline{F} \ l \ i \ a \ [\text{true}]$ is a bounded inductive theorem;
- ▷ if $\mathcal{R}$ includes $(\rho_b^\uparrow), (\rho_r^\uparrow)$ and the rules defining RecUp for some context $R_{F,u,b}^+$, then $R_{F,u,b}^+[i] \approx \text{RU } \underline{F} \ i \ u \ b \ [\text{true}]$ is a bounded inductive theorem;
- ▷ if $\mathcal{R}$ includes $(\rho_b^\downarrow), (\rho_r^\downarrow)$ and the rules defining RecDown for some context $R_{F,l,b}^-$, then $R_{F,l,b}^-[i] \approx \text{RD } \underline{F} \ l \ i \ b \ [\text{true}]$ is a bounded inductive theorem.

If $F(\square_1, \square_2)$ equals $f \ \square_1 \ \square_2$ for some f , then all above statements hold with $\underline{F} = f$.

6.2 Applying Recursor Equivalences

Having matched each of the factorial implementations with a corresponding template in Table 6.1, and thereby with the higher-order recursors of Section 6.1.1, two major challenges remain: deriving equivalences between the various recursors, and applying them in practice to prove program equivalence. In this section, we focus on the latter from an informal and proof-strategic perspective. In Section 6.3 we will discuss how to formalize this informal reasoning within Bounded RI.

Two easy recursor equivalences Two recursor equivalences are immediately suggested by the computational structure of the templates:

- TailUp and RecDown: the current index i appears as the first argument of the function context F , with the accumulator or recursive call being the second at each step. In particular, $T_{F,u}^+[l, b]$ and $R_{F,l,b}^-[u]$ both compute

$$F(u, F(u-1, \dots, F(l+1, F(l, b)) \dots))$$

- TailDown and RecUp: each recursive step introduces a new outer application of F . In particular, $T_{F,l}^-[u, b]$ and $R_{F,u,b}^+[l]$ both compute

$$F(F(\dots F(F(b, u), u-1) \dots, l+1), l)$$

Lemma 6.2.1. Consider a quasi-reductive LCSTRS with $\mathcal{R} \supseteq \mathcal{R}_{\text{rec}}$, and a well-founded ordering $> \supseteq (\rightarrow_{\mathcal{R}} \cup \triangleright)$ such that $\text{TU } f \ x \ y \ z > \text{RD } f \ x' \ y \ (f \ x \ z) [x \leq y \wedge x' = x + 1]$ and $\text{TD } f \ x \ y \ z > \text{RU } f \ x \ y' \ (f \ z \ y) [x \leq y \wedge y' = y - 1]$. Then the following equations are bounded inductive theorems:

$$(1) \text{ TU } f \ x \ y \ z \approx \text{RD } f \ x \ y \ z \qquad (2) \text{ TD } f \ x \ y \ z \approx \text{RU } f \ x \ y \ z$$

We will show how to use Lemma 6.2.1 to prove $\text{factTU } x \approx \text{factRD } x$ and $\text{factTD } x \approx \text{factRU } x [x \geq 1]$. We distinguish two strategies: *one-sided matching* or *two-sided matching*.

6.2.1 One-sided matching

In one-sided matching, we apply both a template-recursor equivalence and a recursor equivalence to reduce one side of an equation to obtain an easier proof goal. Depending on the side of the equation and the recursor equivalence applied, the resulting proof goals may vary in difficulty.

Example 6.2.2. Consider the equation $\text{factTU } x \approx \text{factRD } x [x \geq 1]$. We can rewrite the left-hand side in 4 steps, obtaining the new equation $u \ x \ 2 \ 1 \approx \text{factRD } x [x \geq 1]$. We may then apply one-sided matching with lemma Lemma 6.2.1.(1) to either change the left side or the right side:

- Left side: in Example 6.1.2 we established $u \ x \ 2 \ 1 = T_{\square_1 * \square_2, x}^+[2, 1]$. By Lemma 6.1.3, $T_{\square_1 * \square_2, x}^+[2, 1] \approx \text{TU } [*] \ 2 \ x \ 1$ is an inductive theorem. Hence, it suffices to show $\text{TU } [*] \ 2 \ x \ 1 \approx \text{factRD } x [x \geq 1]$. Rewriting the left-hand side with Lemma 6.2.1.(1), we see that it suffices to show $(E_1) \text{ RD } [*] \ 2 \ x \ 1 \approx \text{factRD } x [x \geq 1]$.
- Right side: we established $\text{factRD } x = R_{\square_1 * \square_2, 2, 1}^- x$ and have an inductive theorem $R_{\square_1 * \square_2, 2, 1}^- x \approx \text{RD } [*] \ 2 \ x \ 1 [x \geq 1]$. Therefore, we should prove $u \ x \ 2 \ 1 \approx \text{RD } [*] \ 2 \ x \ 1 [x \geq 1]$. Applying Lemma 6.2.1.(1) on the right side of this last equation, we obtain $(E_2) u \ x \ 2 \ 1 \approx \text{TU } [*] \ 2 \ x \ 1$.

In both cases in Example 6.2.2, we applied a recursor equivalence to produce an equation that compares two terms of similar iterative structure (e.g. both upward tail recursive). This was effective because such comparisons are often much easier to prove than comparisons between different programming structures: both (E_1) and (E_2) can be proved in an entirely straightforward way (without requiring further lemma equations or generalizations). Such proofs are typically easily done in a fully automatic fashion (although we have not yet implemented this automation in Cora). Unfortunately, the strategy does not always suffice on its own, as illustrated in Example 6.2.3.

Example 6.2.3. To prove $\text{factTD } x \approx \text{factRU } x [x \geq 1]$, the methods from Chapter 4 do not suffice. We try making progress with one-sided matching. First, we reduce both sides of the equation, obtaining $d (x - 1) x \approx U 1 x [x \geq 1]$. Then:

- Left side: using the same strategy as before, we obtain the proof goal $RU [*] 1 (x - 1) x \approx U 1 x [x \geq 1]$. This equation is provable in Bounded RI, but does require an additional lemma: $RU [*] i (x - 1) x \approx U i x [x \geq 1]$. This lemma can be found in a systematic way, using the techniques from Chapter 4.
- Right side: here, we obtain the proof goal $d (x - 1) x \approx TD \underline{1} (x - 1) x [x \geq 1]$. Again, we *can* prove this, but it does require the generalization of the proof goal to $d i x \approx TD \underline{1} i x$. This generalization can be found using the systematic lemma generation technique from Chapter 4. (Note that we could also succeed with a lemma $d i x \approx TD \underline{1} i x [x \geq 1 \wedge i \geq 0]$, obtained from the proof goal simply by dropping the requirement that $i = x - 1$.)

Thus, although we are not immediately done by using recursor equivalences, combining them with existing lemma generation approaches allows us to prove equivalences that were out of reach before.

6.2.2 Two-sided matching

In two-sided matching, we simultaneously apply template-recursor equivalences on *both* sides of an equation, and can therefore do the subsequent RI reasoning entirely in the setting of the recursors. Doing this, we often end up with a proof goal that is an *instance* of a recursor equivalence, and can therefore be concluded in a single step (essentially: applying the recursor equivalence yields a remaining proof goal which is identical terms on both sides – meaning that there is no remaining proof obligation).

Example 6.2.4. We consider again $\text{factTU } x \approx \text{factRD } x [x \geq 1]$. As before, this equation rewrites to $u x 2 1 \approx \text{factRD } x [x \geq 1]$. If we now apply template equivalences on both sides at the same time, we obtain the proof goal $TU [*] 2 x 1 \approx RD [*] 2 x 1 [x \geq 1]$. This is an instance of the bounded inductive theorem $TU f x y z \approx RD f x y z$ from Lemma 6.2.1, so can be immediately removed.

It is not obvious that either one- or two-sided matching is preferable. However, using one over the other can lead to different lemma generation techniques being useful.

Example 6.2.5. Consider $\text{factTD } x \approx \text{factRU } x [x \geq 1]$. Rewriting both sides as far as we can, and then applying template equivalences on both sides at once, we obtain the proof goal $\text{TD } [*] 1 (x - 1) x \approx \text{RU } \underline{F} 1 (x - 1) x$, given that there is a rule $\underline{F} x y \rightarrow y * x$.

Unfortunately, while this looks *similar* to Lemma 6.2.1.(2), it is not an instance because the first arguments to TD and RU are not the same. However, the shape of the equation, and the desire to see the equation as an instance of a known inductive theorem does immediately suggest the lemmas $\text{TD } [*] i y a \approx \text{TD } \underline{F} i y a$ and $\text{RU } [*] i y a \approx \text{RU } \underline{F} i y a$, both of which can be proven in a straightforward way, and which suffice to bring the proof goal in a form that is an instance of the known equation $\text{TD } f x y z \approx \text{RU } f x y z$.

Remark 6.2.6. Note that the reasoning in Example 6.2.2–6.2.5 only proves that the equations under consideration are inductive theorems. To see that they are *bounded* inductive theorems we must impose some additional requirements on $>$ (e.g., that $T_{F,u}^+[i, a] \geq \text{TU } \underline{F} i u a$). Different choices (e.g., one-sided matching versus two-sided matching, or the side of the equation on which we apply an induction hypothesis) yield different requirements. Regardless of these choices, we believe that it is generally easy to satisfy at least the ordering requirements generated by using the template-recursor equivalences and the recursor equivalences. This will be discussed further in Section 6.3.

6.2.3 Conditional recursor equivalence

The remaining recursor equivalences can only be proved conditionally. For example, $\text{TU } f x y z \approx \text{TD } f x y z$ is *not* an inductive theorem; however, if $f :: \text{int} \rightarrow \text{int} \rightarrow \text{int}$ is a commutative and associative function symbol, then $\text{TU } f x y z \approx \text{TD } f x y z$ is one. Such assumptions are expressed through a set $\mathcal{A}$ of axioms: equations that consider specific fixed terms, required to be bounded ground convertible.

Definition 6.2.7 (Conditional bounded inductive theorems). We assume given a fixed LCSTRS with signature Σ and rules $\mathcal{R}$, and a well-founded ordering $>$ that includes $\rightarrow_{\mathcal{R}} \cup >$. Fix typed constants $\square_1, \dots, \square_n$. If C, D, E are context functions in n parameters, we call an expression $C \approx D [E]$ a *n-ary context equation*, and we call $C > D [E]$ an *n-ary context ordering requirement*. Let $\mathcal{A}$ and $\mathcal{E}$ be sets of n -ary context equations and $\mathcal{O}$ a set of n -ary context ordering requirements. We say that “ $\mathcal{A} \models \mathcal{E}$ if $\mathcal{O}$ ” is a *conditional bounded inductive theorem* if, for all terms $s_1, \dots, s_n$ of the appropriate types, the following implication holds: If both

- ▷ $C[s_1, \dots, s_n] \approx D[s_1, \dots, s_n] [E[s_1, \dots, s_n]]$ is a bounded inductive theorem for all
- $C \approx D [E] \in \mathcal{A}$

$\triangleright C[s_1, \dots, s_n] > D[s_1, \dots, s_n] [E[s_1, \dots, s_n]]$ for all $C > D [E] \in \mathcal{O}$

then $P[s_1, \dots, s_n] \approx Q[s_1, \dots, s_n] [R[s_1, \dots, s_n]]$ is a bounded inductive theorem for all $P \approx Q [R] \in \mathcal{E}$.

(Note that “is a bounded inductive theorem” implicitly requires $C[s_1, \dots, s_n] \approx D[s_1, \dots, s_n] [E[s_1, \dots, s_n]]$ to be a valid equation; e.g., if $\square_i$ occurs inside the constraint E , then s_i must be a theory term.)

Lemma 6.2.8. Let $\square_1, \square_2 :: \text{int} \rightarrow \text{int} \rightarrow \text{int}$. The following are conditional bounded inductive theorems:

- (1). $\{\square_1 x (\square_2 y z) \approx \square_2 (\square_1 x y) z, \square_1 x y \approx \square_2 y x\} \vdash \{\text{TU } \square_1 x y a \approx \text{TD } \square_2 x y a\}$ if

$$\left\{ \begin{array}{ll} \text{TU } \square_1 x y a > \text{TD } \square_2 x_1 y (\square_1 x a) & [x \leq y \wedge x_1 = x + 1 \wedge y_1 = y - 1] \\ \text{TD } \square_2 x_1 y (\square_1 x a) > \text{TD } \square_2 x_1 y_1 (\square_1 x (\square_2 a y)) & [x \leq y \wedge x_1 = x + 1 \wedge \\ & y_1 = y - 1 \wedge x_1 \leq y] \end{array} \right\}$$
- (2). $\{\square_1 x (\square_1 y z) \approx \square_1 y (\square_1 x z), \square_1 x y \approx \square_2 y x\} \vdash \{\text{TD } \square_2 x y a \approx \text{RD } \square_1 x y a\}$ if

$$\left\{ \begin{array}{ll} \text{TD } \square_2 x y a > \text{RD } \square_1 x y_1 (\square_2 a y) & [x \leq y \wedge y_1 = y - 1] \\ \text{RD } \square_1 x y_1 (\square_2 a y) > \square_1 y_1 (\square_1 y (\text{RD } \square_1 x y_2 a)) & [x \leq y \wedge x \leq y_1 \wedge y_2 = y_1 - 1] \end{array} \right\}$$
- (3). $\{\square_1 (\square_1 x y) z \approx \square_1 (\square_1 x z) y, \square_1 x y \approx \square_2 y x\} \vdash \{\text{TU } \square_2 x y a \approx \text{RU } \square_1 x y a\}$ if

$$\left\{ \begin{array}{ll} \text{TU } \square_2 x y a > \text{RU } \square_1 x_1 y (\square_2 x a) & [x \leq y \wedge x_1 = x + 1] \\ \text{RU } \square_1 x_1 y (\square_2 x a) > \square_1 (\square_1 (\text{RU } \square_1 x_2 y a) x) x_1 & [x \leq y \wedge x_1 \leq y \wedge x_2 = x_1 + 1] \end{array} \right\}$$
- (4). $\{\square_1 (\square_2 x y) z \approx \square_2 x (\square_1 y z), \square_1 x y \approx \square_2 y x\} \vdash \{\text{RU } \square_1 x y a \approx \text{RD } \square_2 x y a\}$ if

$$\left\{ \begin{array}{ll} \text{RU } \square_1 x y a > \square_1 (\text{RD } \square_2 x_1 y a) x & [x \leq y \wedge x_1 = x + 1] \\ \square_2 y (\text{RD } \square_2 x y_1 a) > \square_2 y (\square_1 (\text{RD } \square_2 x_1 y_1 a) x) & [x \leq y \wedge x_1 = x + 1 \wedge \\ & y_1 = y - 1 \wedge x_1 \leq y] \end{array} \right\}$$

Note that if $\square_1$ and $\square_2$ in Lemma 6.2.8 are instantiated by the same associative-commutative term, the axioms are always satisfied, but we only require what is actually needed for proving the equations in $\mathcal{E}$.

With Lemma 6.2.8 we can prove the four remaining factorial equivalences. Like in the previous examples, we either apply one-sided matching or two-sided matching. The only difference is that we now have to establish bounded ground convertibility of the axiom sets, which in our case involves a trivial verification as they all instantiate both $\square_1$ and $\square_2$ by $*$, or by a symbol $\underline{f}$ with $\underline{f} x y \rightarrow y * x$. We therefore skip the proof details.

6.2.4 Another example: equivalence with composite operators

We have seen how the template approach works on simple, first-order examples through the various factorial implementations. Here, we aim to illustrate that Lemmas 6.2.1 and 6.2.8 are general enough to handle more involved examples that naturally occur in real-life programming, including cases where the context functions are distinct higher-order constructs. Consider below `funfactTU` and `funfactTD` which are higher-order variants of `factTU` and `factTD`, computing $(h, x) \mapsto \prod_{i=1}^x h(i)$ for all $x \geq 1$.

`funfactTU` $h\ x \rightarrow u\ h\ x\ 1\ 1$ `funfactTD` $h\ x \rightarrow d\ h\ x\ 1$
 $u\ h\ x\ i\ a \rightarrow a$ $[i > x]$ $d\ h\ x\ a \rightarrow a$ $[x < 1]$
 $u\ h\ x\ i\ a \rightarrow u\ h\ x\ (i + 1)\ ((h\ i) * a)$ $[i \leq x]$ $d\ h\ x\ a \rightarrow d\ h\ (x - 1)\ (a * (h\ x))$ $[x \geq 1]$

Compared to `factTU` and `factTD`, the expressions in the accumulator arguments are changed: $i * a$ is replaced by $(h\ i) * a$ and $a * x$ is replaced by $a * (h\ x)$. To match them with Table 6.1, we choose:

- Tailup: $T_{F,x}^+ = u\ h\ x\ \square_1\ \square_2$ with $F := (h\ \square_1) * \square_2$; $\underline{F}\ x_1\ x_2 \rightarrow (h\ x_1) * x_2$
- Taildown: $T_{G,1}^- = d\ h\ \square_1\ \square_2$ with $G := \square_1 * (h\ \square_2)$. $\underline{G}\ x_1\ x_2 \rightarrow x_1 * (h\ x_2)$

Hence, using Lemma 6.1.3, both `funfactTU` $h\ x \approx \text{TU } \underline{F}\ 1\ x\ 1$ and `funfactTD` $h\ x \approx \text{TD } \underline{G}\ 1\ x\ 1$ are bounded inductive theorems. To prove that `funfactTU` $h\ x \approx \text{funfactTD } h\ x$ is an inductive theorem, it therefore suffices to show that $\text{TU } \underline{F}\ 1\ x\ 1 \approx \text{TD } \underline{G}\ 1\ x\ 1$ is one. This is an instance of $\text{TU } \underline{F}\ u\ v\ w \approx \text{TD } \underline{G}\ u\ v\ w$, which is a bounded inductive theorem by Lemma 6.2.8.(1), provided that the equations in the condition are. This condition can indeed be verified, hence we have shown the result using two-sided matching.

6.3 Integrating templates in Rewriting Induction

In this section, we explain the technical details of using templates as a lemma-generation method in Bounded RI. Rather than presenting the RI proofs, we discuss the corresponding induction proofs in a self-contained manner. In particular, we focus on the ordering requirements arising in the different components of template-based proofs and explain how these can be recombined in concrete instances to obtain a complete induction proof.

Nevertheless, all proofs discussed in this chapter have been carried out in Cora and are available at <https://cs.ru.nl/~cynthiakop/experiments/lfa26/>. A permanent (frozen) archive of the content on the website is available at <https://doi.org/10.5281/zenodo.22304521>.

6.3.1 Template-recursor equivalences

The equivalences of Lemma 6.1.3 can all be established by straightforward induction proofs, each following a similar induction procedure. As a representative example, we consider the equivalence $T_{F,u}^+[i, a] \approx \text{TU } \underline{F} \ i \ u \ a[\text{true}]$:

- we prove that for all i, a , by induction on $\{T_{F,u}^+[i, a], \text{TU } \underline{F} \ i \ u \ a\}$ using the multiset extension of $>$
- there are two cases: $i > u$ and $i \leq u$
- in the former case, both sides reduce to a and we are done
- in the latter case, $T_{F,u}^+[i, a] \rightarrow_{\mathcal{R}} T_{F,u}^+[i+1, F(i, a)]$ and $\text{TU } \underline{F} \ i \ u \ a \rightarrow_{\mathcal{R}} \text{TU } \underline{F} \ (i+1) \ u \ (F(i, a))$. This yields a multiset $\{T_{F,u}^+[i+1, F(i, a)], \text{TU } \underline{F} \ (i+1) \ u \ (F(i, a))\}$, which is an instance of the induction hypothesis, and since both sides have decreased in $>$ (since they are smaller by $\rightarrow_{\mathcal{R}}$, which is included in $>$), we are done.

Note that we only used the decrease in $\rightarrow_{\mathcal{R}}$ to complete the proof, which is why no additional ordering requirements are imposed. The proofs of the remaining template-recursor equivalences are similarly simple.

6.3.2 Recursor equivalences

The recursor equivalences of Lemma 6.2.1 are a bit more difficult than the template-recursor equivalences because we have to use a second induction hypothesis. For example, if it is given that $\text{TU } f \ x \ y \ z > \text{RD } f \ x' \ y \ (f \ x \ z) [x \leq y \wedge x' = x+1]$, we could prove $\text{TU } f \ x \ y \ z \approx \text{RD } f \ x \ y \ z$ from Lemma 6.2.1 as follows:

- let $\varsigma = \text{TU } f \ x \ y \ z$ and $\tau = \text{RD } f \ x \ y \ z$; we prove that $\text{TU } f \ x \ y \ z \approx \text{RD } f \ x \ y \ z$ for all f, x, y, z , by induction using the multiset extension of $>$ on the multiset $\{\varsigma, \tau\}$
- if $x > y$ then both sides reduce to z and we are done, so assume $x \leq y$
- in this case, the sides reduce to $\text{TU } f \ x' \ y \ (f \ x \ z)$ and $f \ y \ (\text{RD } f \ x \ y' \ z)$ respectively, where x' is the value corresponding to $x+1$ and y' corresponds to $y-1$
- by the assumptions that (a) $\rightarrow_{\mathcal{R}}$ is included in $>$, and (b) $\varsigma > \text{RD } f \ x' \ y \ (f \ x \ z) [x \leq y \wedge x' = x+1]$, we can use the induction hypothesis to observe that $\text{TU } f \ x' \ y \ (f \ x \ z) \approx \text{RD } f \ x' \ y \ (f \ x \ z) [x \leq y \wedge x' = x+1]$ is a bounded inductive theorem. Again using points (a) and (b), we can increase the bounds to ς, τ and obtain $\text{TU } f \ x' \ y \ (f \ x \ z) \xleftrightarrow[\mathcal{R}]{\varsigma, \tau \geq^*} \text{RD } f \ x' \ y \ (f \ x \ z) [x \leq y \wedge x' = x+1]$.

- ▷ by transitivity of $\xleftrightarrow[\mathcal{R}]{\varsigma, \tau \geq^*}$, it suffices to show $\text{RD } f \ x' \ y \ (f \ x \ z) \xleftrightarrow[\mathcal{R}]{\varsigma, \tau \geq^*} f \ y \ (\text{RD } f \ x \ y' \ z)$ $[x \leq y \wedge x' = x + 1 \wedge y' = y - 1]$. To do so, we start (a second) induction on $\{\text{RD } f \ x' \ y \ (f \ x \ z), f \ y \ (\text{RD } f \ x \ y' \ z)\}$
- ▷ if $x' > y$, which is equivalent to saying $x > y'$, then both sides reduce to $f \ y \ z$; so we assume $x' \leq y$
- ▷ then the sides reduce to $f \ y \ (\text{RD } f \ x' \ y' \ (f \ x \ z))$ and $f \ y \ (f \ y' \ (\text{RD } f \ x \ y'' \ z))$ respectively, where y'' is the value corresponding to $y' - 1$; applying the induction hypothesis to the underlined part, we are done!

The proof of $\text{TD } x \ y \ z \approx \text{RU } x \ y \ z$ is structured in the same way.

6.3.3 Conditional recursor equivalences

In Section 6.3.2, the induction hypothesis $\text{TU } f \ x' \ y \ (f \ x \ z) \approx \text{RD } f \ x' \ y \ (f \ x \ z) [x \leq y \wedge x' = x + 1]$ was applied on the proof goal $\text{TU } f \ x' \ y \ (f \ x \ z) \approx f \ y \ (\text{RD } f \ x \ y' \ z)$ to eliminate the TU-dependency, and continue with a proof goal where both sides depended on the same recursor RD. This let us start a second induction on

$$(\mathbf{H}) \quad \text{RD } f \ x' \ y \ (f \ x \ z) \approx f \ y \ (\text{RD } f \ x \ y' \ z) [x \leq y \wedge x' = x + 1 \wedge y' = y - 1]$$

resulting in an equation $(\mathbf{E})$ which is a literal instance of $(\mathbf{H})$, thus finishing the proof.

The proofs of the conditional recursor equivalences start similarly: the first induction hypothesis is used to obtain a resulting proof goal that only involves one recursor. However, the moment we want to apply the second induction hypothesis the proof procedure starts to deviate: $(\mathbf{E})$ is not a literal instance of $(\mathbf{H})$ and we first have to apply an axiom to make sure that their term shapes coincide:

Consider $\{\Box_1 x \ (\Box_2 y \ z) \approx \Box_2 (\Box_1 x \ y) \ z, \Box_1 x \ y \approx \Box_2 y \ x\} \vdash \{\text{TU } \Box_1 x \ y \ a \approx \text{TD } \Box_2 x \ y \ a\}$

- ▷ Let $\varsigma = \text{TU } \Box_1 x \ y \ a$ and $\tau = \text{TD } \Box_2 x \ y \ a$
- ▷ The proof starts similar as in Section 6.3.2, using the IH to remove the TU-dependency, which requires the assumption $\text{TU } \Box_1 x \ y \ a > \text{TD } \Box_2 x_1 \ y \ (\Box_1 x \ a) [x \leq y \wedge x_1 = x + 1 \wedge y_1 = y - 1]$.
- ▷ Therefore, we should prove $\text{TD } \Box_2 x_1 \ y \ (\Box_1 x \ a) \xleftrightarrow[\mathcal{R}]{\varsigma, \tau \geq^*} \text{TD } \Box_2 x \ y_1 \ (\Box_2 a \ y) [x \leq y \wedge x_1 = x + 1 \wedge y_1 = y - 1]$. We do so by induction on the multiset corresponding to the equation

$$(\mathbf{H}). \quad \text{TD } \Box_2 x_1 \ y \ (\Box_1 x \ a) \approx \text{TD } \Box_2 x \ y_1 \ (\Box_2 a \ y)$$

- (1). The case $x_1 > y$ leads to the proof goal $\Box_1 x \ a \approx \Box_2 a \ y [x \leq y \wedge x_1 = x + 1 \wedge y_1 = y - 1 \wedge x_1 > y]$, which we remove using the axiom $\Box_1 x \ y \approx \Box_2 y \ x$ (note that the constraint implies $x = y$).

(2). The case $x_1 \leq y$ leads to the proof goal

$$(E). \quad \begin{array}{l} \text{TD } \Box_2 x_1 y_1 (\Box_2 (\Box_1 x a) y) \approx \text{TD } \Box_2 x y_2 (\Box_2 (\Box_2 a y) y_1) \\ [x \leq y \wedge x_1 = x + 1 \wedge y_1 = y - 1 \wedge x_1 \leq y \wedge y_2 = y_1 - 1] \end{array}$$

- We cannot immediately apply the induction hypothesis **(H)** to eliminate **(E)**: matching the left-hand side fails due to the accumulator-argument (in **(E)** this accumulator is headed by $\Box_2$, while in **(H)** it is headed by $\Box_1$). Instead, we apply the axiom $\Box_1 x (\Box_2 y z) \approx \Box_2 (\Box_1 x y) z$ to obtain

$$(E'). \quad \begin{array}{l} \text{TD } \Box_2 x_1 y_1 (\Box_1 x (\Box_2 a y)) \approx \text{TD } \Box_2 x y_2 (\Box_2 (\Box_2 a y) y_1) \\ [x \leq y \wedge x_1 = x + 1 \wedge y_1 = y - 1 \wedge x_1 \leq y \wedge y_2 = y_1 - 1] \end{array}$$

We remark that this deduction step is only allowed if $\text{TD } \Box_2 x_1 y (\Box_1 x a) > \text{TD } \Box_2 x_1 y_1 (\Box_1 x (\Box_2 a y)) [x \leq y \wedge x_1 = x + 1 \wedge y_1 = y - 1 \wedge x_1 \leq y]$ is satisfied.

- The proof is finished by applying **(H)** to **(E')**

All conditional bounded inductive theorems have a similar proof structure.

6.3.4 Templates as lemma generation method

Bringing all this together, let us consider how one might use templates within Bounded RI as a lemma generation method.

Bounded ground convertibility in $\mathcal{R}$ A key issue is that Bounded RI establishes bounded ground convertibility with respect to a given term rewriting system $\mathcal{R}$, which typically does not contain the recursors from Section 6.1. However, these recursors play a crucial role in the inductive reasoning underlying the template method. For example, trying to prove an equation like $\text{factTU } x \approx \text{factRD } x [x \geq 1]$, we first have to identify one or both programs as recursor instances: $\text{factRD } x \approx \text{RD } [*] 2 x 1$ and $\text{TU } f x y z \approx \text{RD } f x y z$. To facilitate this reasoning within Bounded RI, we have to extend the rewrite system to $\mathcal{R} \cup \{(\tau_r^\uparrow), (\tau_b^\uparrow), (\rho_r^\downarrow), (\rho_b^\downarrow)\}$.

Note that Lemmas 6.1.3, 6.2.1 and 6.2.8 assume that the recursor rules are included in $\mathcal{R}$. If this is not initially the case, we must import them and consequently establish convertibility in $\mathcal{R} \cup \mathcal{R}_{\text{rec}}$. Thus, we obtain $s\gamma \xrightarrow{s\gamma, t\gamma \geq^*}^*_{\mathcal{R} \cup \mathcal{R}_{\text{rec}}} t\gamma$ rather than $s\gamma \xrightarrow{s\gamma, t\gamma \geq^*}^*_{\mathcal{R}} t\gamma$. Fortunately, in practice Bounded RI is typically applied in settings where ground confluence is either a given property, or can be established via Corollary 5.6.11. With this assumption $s\gamma \xrightarrow{s\gamma, t\gamma \geq^*}^*_{\mathcal{R} \cup \mathcal{R}_{\text{rec}}} t\gamma$ implies that there is some u such that both $s\gamma \rightarrow_{\mathcal{R} \cup \mathcal{R}_{\text{rec}}} u$ and $t\gamma \rightarrow_{\mathcal{R} \cup \mathcal{R}_{\text{rec}}} u$. If the symbols TU, TD, RU, RD do not occur in $s\gamma$ or $t\gamma$, then only rules in $\mathcal{R}$ can be applied in these derivations, so we really have $s\gamma \rightarrow_{\mathcal{R}} u \leftarrow_{\mathcal{R}} t\gamma$: the equation is also a bounded inductive theorem in $\mathcal{R}$.

Ordering requirements Regarding the ordering $>$, we claim that the ordering requirements imposed by $\mathcal{R}_{\text{rec}}$ and the lemmas in this chapter are pretty light: the rules of $\mathcal{R}_{\text{rec}}$ do not depend on those in $\mathcal{R}$, and in the lemmas the symbols in $\mathcal{R}$ dominate those in $\mathcal{R}_{\text{rec}}$. For example, all explicit ordering requirements, and the requirement that $\rightarrow_{\mathcal{R}_{\text{rec}}} \subseteq >$, are captured by a simple higher-order recursive path ordering [25] with $f > TU > TD > RU > RD$, where f is any symbol defining the T and R rules. (There is one disclaimer: for the ordering requirements in Lemma 6.2.8 this also depends on the choice for $\square_1$ and $\square_2$; RPO-orientability certainly holds if $\square_1 = \square_2$, or if these are non-recursive symbols like $\underline{f}$.) Or if we use static dependency pairs [24] to prove termination of a corresponding rewrite relation [28], the dependency pairs for the lemmas and the rules in $\mathcal{R}_{\text{rec}}$ are easily eliminated by the dependency graph and the value criterion. (Again, this does depend on what is used for $\square_1$ and $\square_2$.)

If the ordering requirements do cause difficulties, it is worth noting that the method does admit for some flexibility; for example, by applying an axiom on the other side of the equation, different ordering requirements arise, and also in Lemma 6.2.8 we could obtain different requirements by applying the lemmas slightly differently, if this is helpful for the given choice of $\square_1$ and $\square_2$.

6.4 Conclusion and future work

In this chapter, we have introduced several useful templates and shown how they can be incorporated into Bounded RI as a lemma generation method to prove program equivalence in a manual or automatic process. In particular, templates can be seen as a generalization method operating at the level of programming constructs. The advantage of this approach becomes apparent in settings where existing lemma generation methods from constrained rewriting fail due to the need for non-polynomial invariants. We used the factorial implementations as a simple running example to explain the template method, and also illustrated that it is applicable to simple higher-order variants.

The template method has not yet been implemented as an automatic lemma-generation heuristic. Nevertheless, all template-based proofs presented in this chapter have been manually verified in Cora. We believe that the template approach lends itself well towards automation, although we have not done so yet since the implementation of Bounded RI is also not yet fully automatic even without adding postulates. Once the technical prerequisites are in place, it should not be hard to recognise LCSTRSs as instances of the recursors (as this is mostly a syntactic matching problem), but the primary challenge will be to decide how to apply the lemmas: as we have seen in Section 6.2.1 and 6.2.2, there is no unique best approach, and different strategies might lead to different outcomes. We expect this to be a matter of trial and error.

6.4.1 Related work

This is not the first work on using templates to recognise particular programming structures in equivalence proofs. The most relevant related work is an existing term rewriting method [19] that defines templates to verify program transformations (including tail recursion versus non-tail recursion). We will compare this method with the approach introduced in this chapter, using an illustrative example that considers a tail-recursion template [19] being matched with a TRS to sum a list of natural numbers:

$$\begin{array}{l}
 f(u_4) \rightarrow f_1(u_4, b) \\
 f_1(a, u_5) \rightarrow u_5 \\
 f_1(c(u_6, v_6), w_6) \rightarrow f_1(v_6, g(w_6, u_6)) \\
 g(b, u_7) \rightarrow u_7 \\
 g(d(u_8, v_8), w_8) \rightarrow d(u_8, g(v_8, w_8))
 \end{array}$$

Template (tail recursion)

$$\begin{array}{l}
 \text{sum}(x_4) \rightarrow \text{sum1}(x_4, 0) \\
 \text{sum1}([], x_5) \rightarrow x_5 \\
 \text{sum1}(x_6 : y_6, z_6) \rightarrow \text{sum1}(y_6, +(z_6, x_6)) \\
 +(0, x_7) \rightarrow x_7 \\
 +(s(y_8), z_8) \rightarrow s(+(y_8, z_8))
 \end{array}$$

Matchable example (summing a list)

Unconstrained rewriting The existing method [19] is defined for first-order, unconstrained rewriting. This suffices for reasoning about programs on inductive data structures through pattern matching (like natural numbers and lists), but less so for structures like integers where cases are often investigated through constraints rather than pattern matching. Hence, the tail recursion template does not instantiate to $\text{factTU } x$ (or $\text{factTD } x$), as there is no obvious correspondence between the constructor-based recursion in f_1 and the constraint-based recursion of u .

Complete program descriptions We also notice that, in contrast to the approach in this chapter, the tailup template in principle captures a complete program description. We can subdivide the template in three parts:

- (1). *Initialization*: the rule $f(u_4) \rightarrow f_1(u_4, b)$ initializes the accumulator b
- (2). *Recursive part*: by the two f_1 -rules
- (3). *Auxiliary function definition*: by the two g -rules

Our templates only consider the recursive part. This is possible because we do not treat templates as a stand-alone approach, but as integrated within RI.

First-order versus higher-order The restriction to *first-order* term rewriting results in a more restricted setting than in our higher-order template method. For example, in the above tail recursion template the function symbol g may capture symbols such as $+$ or append , but cannot represent multiplication. In our approach:

- recursors such as TU are defined, via $(\tau_b^\uparrow), (\tau_r^\uparrow)$, for an *arbitrary* function $f :: \text{int} \rightarrow \text{int} \rightarrow \text{int}$, which is given to the recursor as a higher-order argument
- the instantiation $f := \underline{F}$ of a specific auxiliary function is handled during the RI procedure (i.e. during or after the one-sided matching or two-sided matching), and does not appear in $(\tau_b^\uparrow)$ or $(\tau_r^\uparrow)$

That is, we can take advantage of the higher-order nature of LCSTRSs to reuse the same recursors with entirely different instantiations. For example, we would recognize f_1 as an instance of `foldl`, where

$$\text{foldl } f \ a \ [] \rightarrow a \qquad \text{foldl } f \ a \ (x : xs) \rightarrow \text{foldl } f \ (f \ a \ x) \ xs$$

Although we have not considered lists recursors, the same idea applies: we can use `foldl` (and its commonly-used counterpart `foldr`) as a recursor to represent a tail recursion template on lists. We can obtain $\text{sum1 } l \ a \xrightarrow{\geq^*}_{\mathcal{R}} \text{foldl } [+] \ a \ l$ as an instance of a template-recursor equivalence to conclude $\text{sum } l \xrightarrow{\geq^*}_{\mathcal{R}} \text{foldl } [+] \ 0 \ l$. We can then introduce a (conditional) equivalence between `foldl` and `foldr`, to prove equivalence between `sum1` l a and another list-summation implementation.

Equivalence by transformation schemes A final difference with our approach is the mechanism being used to establish template equivalence. The existing approach [19] is not based on well-founded induction. Instead, equivalence is proven via *equivalent term rewriting systems*, established via specifically designed transformation rules. Correctness of these transformations relies on *confluence*. Additionally, one has to assume *sufficient completeness*, a property very similar to quasi-reductivity. However, the authors point out that in practice verifying sufficient completeness requires termination. Thus, despite the difference in proof mechanism, the prerequisites are similar to our approach: RI also requires quasi-reductivity and termination, and as observed in Section 6.3.4, ground confluence is needed to obtain an inductive theorem over $\rightarrow_{\mathcal{R}}$ rather than just $\rightarrow_{\mathcal{R} \cup \mathcal{R}_{\text{rec}}}$.

6.4.2 Future work

The templates in Table 6.1 only capture a limited class of programs. We aim to extend our templates and corresponding equivalences. We consider some natural candidates.

Foldl and foldr As explained in Section 6.4.1, `foldl` and `foldr` may serve as recursors to represent tail recursion and non-tail recursion on foldable data structures. For example, consider an alternative implementation of the list-summation program:

$$\text{sumRec } [] \rightarrow 0 \qquad \text{sumRec } (x : xs) \rightarrow x + (\text{sumRec } xs)$$

We may use `foldl` and `foldr` to prove $\text{sum } l \xrightarrow{\geq^*}_{\mathcal{R}} \text{sumRec } l$. In Section 6.4.1 we already

asserted $\text{sum } l \xrightarrow{\geq^*}_{\mathcal{R}} \text{foldl } [+] \ 0 \ l$. Similarly, we can establish $\text{sumRec } l \xrightarrow{\geq^*}_{\mathcal{R}} \text{foldr } [+] \ 0 \ l$. Then we can complete the proof by applying the following bounded conditional inductive theorem with two-sided matching:

$$\{\Box_1 x y \approx \Box_2 y x, \Box_1 (\Box_1 x y) z \approx \Box_1 (\Box_1 x z) y\} \vdash \{\text{foldl } \Box_1 a l \approx \text{foldr } \Box_2 a l\} \text{ if } \left\{ \begin{array}{l} \text{foldr } \Box_2 a (x_1 : l_1) > \Box_2 x_1 (\text{foldl } \Box_1 a l_1) \\ \text{foldl } \Box_1 (\Box_1 a x_1) (x_2 : l_2) > \text{foldl } \Box_1 (\Box_1 (\Box_1 a x_2) x_1) l_2 \end{array} \right\}$$

Extensions and variations on $\mathcal{R}_{\text{rec}}$ We may define extensions and variations of $\mathcal{R}_{\text{rec}}$. A useful extension is to allow for arbitrary step-size $k > 0$. For example, we may consider

$$\begin{array}{ll} \text{TU } f k i y a \rightarrow a \ [i > y] & \text{TU } f k i y a \rightarrow \text{TU } f k (i+k) y (f i a) \ [k > 0 \wedge i \leq y] \\ \text{RD } f k x i z \rightarrow z \ [i < x] & \text{RD } f k x i z \rightarrow f i (\text{RD } f k x (i-k) z) \ [k > 0 \wedge i \geq x] \end{array}$$

and establish $\text{TU } f k x y a \approx \text{RD } f k x y a \ [k > 0 \wedge x \equiv y \pmod{k}]$. This can for instance be used to prove the equivalence $\text{facOddTU } x \approx \text{facOddRD } x \ [x \equiv 1 \pmod{2}]$, in

$$\begin{array}{ll} \text{facOddTU } x \rightarrow u x \ 1 \ 1 & \text{facOddRD } x \rightarrow 1 \ [x \leq 1] \\ u x i a \rightarrow a \ [i > x] & \text{facOddRD } x \rightarrow x * (\text{facOddRD } (x-2)) \ [x > 1] \\ u x i a \rightarrow u x (i+2) (i * a) \ [i \leq x] & \end{array}$$

A variation of $\mathcal{R}_{\text{rec}}$ could be to replace the $+k$ by $/k$, to be used in e.g. sorting algorithms.

Recurrence relations Another programming construct are recurrence relations. For example, we can use the following recursive and tail-recursive recursors to represent recurrence relations of order 2:

$$\begin{array}{ll} \text{R } n \ \varphi \ a_0 \ a_1 \rightarrow a_0 \ [n < 1] & \text{R } n \ \varphi \ a_0 \ a_1 \rightarrow a_1 \ [n = 1] \\ \text{R } n \ \varphi \ a_0 \ a_1 \rightarrow \varphi \ n \ (\text{R } (n-2) \ \varphi \ a_0 \ a_1) (\text{R } (n-1) \ \varphi \ a_0 \ a_1) \ [n > 1] & \\ \text{T } n \ i \ \varphi \ x \ y \rightarrow y \ [i > n] & \text{T } n \ i \ \varphi \ x \ y \rightarrow \text{T } n \ (i+1) \ \varphi \ y (\varphi \ i \ x \ y) \ [i \leq n] \end{array}$$

We establish $\text{R } n \ \varphi \ a_0 \ a_1 \approx \text{T } n \ 2 \ \varphi \ a_0 \ a_1 \ [n > 1]$ and use this to prove $\text{fib}_R x \approx \text{fib}_T x$ in

$$\begin{array}{lll} \text{fib}_R x \rightarrow 0 & [x \leq 0] & \text{fib}_T x \rightarrow \text{t } x \ 1 \ 0 \ 1 \\ \text{fib}_R x \rightarrow 1 & [x = 1] & \text{t } x \ i \ a_1 \ a_2 \rightarrow a_1 \ [i > x] \\ \text{fib}_R x \rightarrow (\text{fib}_R (x-1)) + (\text{fib}_R (x-2)) \ [x > 1] & \text{t } x \ i \ a_1 \ a_2 \rightarrow \text{t } x \ (i+1) \ a_2 \ (a_1 + a_2) \ [i \leq x] \end{array}$$

Chapter 7

Conclusion and future Work

In Chapter 1 we set out the following objectives:

- (1). Define a notion of program equivalence for LCSTRSs.
- (2). Develop a formal system for proving program equivalence for LCSTRSs.
- (3). Extend this system to disprove equations that do not represent program equivalence.
- (4). Develop a method for establishing ground confluence in LCSTRSs.
- (5). Develop heuristics that facilitate semi-automatic, human-guided program equivalence proofs in practice.

Here, we revisit these objectives and assess the extent to which they have been achieved in this thesis. We also discuss several directions for future work.

Objective 1 In this thesis we employed several notions of program equivalence:

Higher-order inductive theorems (ground convertibility) (Section 3.1.2): the most basic notion of program equivalence, used in the proof system HORI.

We furthermore used two variants of ground convertibility:

- (i). *Global inductive theorems* (Section 3.3.2): the extensional variant which considers ground convertibility in all natural extensions of the given LCSTRS.
- (ii). *Bounded ground convertibility* (Section 5.3.3): a stricter version of ground convertibility with a built-in connection to a bounding pair $(>, \geq)$, which is used in Bounded RI. Essentially, a bounded ground convertibility sequence $s \xleftrightarrow[\mathcal{R}]{\varsigma, \tau \geq^*} t [\varphi]$ expresses the intuition that every rewrite-term in the corresponding convertibility path is below ς or τ . This makes it a useful notion for proving ground confluence.

Also, we discussed two other notions of equivalence:

- ▷ *Extensional equivalence* (Section 3.1.2). In the context of higher-order rewriting, we recognized the relevance of extensional inductive theorems. We explained, however, that extensionality introduces some technical challenges, making it harder to develop a corresponding proof system, and which we have not further investigated. Nevertheless, the embedding of extensional inductive theorems in Bounded RI constitutes an interesting direction for future work.
- ▷ *Equivalence modulo axioms* (Section 5.9). Another interesting equivalence notion is based on a restriction of ground convertibility: rather than quantifying convertibility over all ground substitutions we only include those instances satisfying a set of axioms. This is motivated by situations where we can only prove equivalence once allowed to assume properties on the involved function variables (such as commutativity). Thus, this version of ground convertibility is practically applicable for analyzing programs that are guaranteed to be provided with input arguments that satisfy a certain set of properties. However, we have not further investigated this notion of equivalence, and it therefore remains future work.

Objective 2 A crucial ingredient for all the RI systems introduced in this thesis is higher-order quasi-reductivity (Section 3.1.3). In this thesis we generalized the first-order notion of quasi-reductivity to the higher-order setting by replacing constructor terms with semi-constructor terms. This allowed us to extend the RI system for LCTRSs to LCSTRSs, requiring almost no changes to the underlying proof system or soundness argument. This results in HORI. Higher-order quasi-reductivity also forms the basis for the two other RI systems developed in this thesis: Global RI and Bounded RI. For Global RI, we even require a stricter version, called strong quasi-reductivity (Section 3.3.1), to exclude extensions with ill-behaved program behavior.

Thus, in this thesis we developed the following systems for proving program equivalence:

- ▷ HORI (Section 3.2) and its extensional counterpart Global RI (Section 3.3.4)
- ▷ Bounded RI (Section 5.2.1)

With three RI systems available, it is worthwhile to summarize their theoretical and practical relevance in the setting of program equivalence. From a practical perspective, the only currently available RI tool for LCSTRSs is Cora, realizing an implementation of Bounded RI. In particular, Bounded RI is preferred over HORI, as it is designed to minimize termination requirements (enhancing its applicability in practice) and proves bounded ground convertibility. Since bounded ground convertibility implies ground convertibility, there is no need to maintain HORI because every equivalence proof in HORI can also be performed in Bounded RI.

However, Bounded RI has not yet reached a form that fully accounts for all aspects of a complete treatment of program equivalence. In particular, it does not incorporate global inductive theorems, hence cannot yet replace Global RI. Moreover, like HORI and Global RI, it does not consider extensional inductive theorems.

Implementation. A prototype version of Bounded RI has been implemented in Cora. A major challenge in the automation of equivalence proofs is the construction of bounding pairs. In Section 5.2.1 we explained the benefit of strong boundedness: many of the required inequalities in Figure 5.1 can be verified by an easy syntactic check. Strong boundedness is also employed in Cora.

While strong boundedness allows us to easily eliminate part of the ordering requirements, we usually have a remaining set REQS of requirements to be verified. These are addressed via the approach from Section 5.4.1: construct a terminating rewrite relation $\rightarrow_{Q \cup \mathcal{R}}^+$, where the set of rewrite rules Q contains all rewrite-rule-representations of the requirements in REQS. Termination of $Q \cup \mathcal{R}$ is verified via the built-in Cora methods.

Bounded RI extensions. We propose two lines for future Bounded RI development:

- *Extensional Bounded RI* (Section 3.4). We remarked that we may extend HORI to prove extensional inductive theorems. The same idea can be used in Bounded RI.
- *Global Bounded RI* (Section 5.9). An interesting question is whether it is possible to extend Bounded RI to include Global RI. Not only for proving global inductive theorems, but also disproving equations which are not global inductive theorems.

Considering Global Bounded RI, we suspect that under the assumption of strong quasi-reductivity it is possible to guarantee that a bounding pair $(>, \geq)$ on $T(\Sigma, \mathcal{V})$ can be extended to a bounding pair $(>', \geq')$ on $T(\Sigma', \mathcal{V}')$, for an arbitrary natural extension $\mathcal{L}' = (\mathcal{R}', \Sigma')$ of $\mathcal{L} = (\mathcal{R}, \Sigma)$, but we have not yet investigated this.

Unbounded RI (Section 5.9). The name Bounded RI refers to the fact that the deduction rules are designed to preserve bounded equation contexts. This is convenient, as with Bounded RI we essentially incorporate a heuristic for constructing an ordering pair, allowing for a convenient presentation of deduction rules. However, bounded equation contexts are not strictly necessary for proving ground convertibility, provided that the multiset inequalities used in the soundness proof hold.

Objective 3 Similar to what has been done in [22] for LCTRSs, we showed that for ground confluent LCSTRSs we can use Bounded RI to disprove equations which are not inductive theorems (Section 5.2.3). To this end, we introduced the deduction rule (Disprove) which can be applied to conclude $\perp$ from a contradictory equation. However,

the use of (Disprove) is only allowed for *complete proof states*. This restriction essentially prevents us from introducing contradictory equations that were not already there, e.g. via the introduction of a new equation via (Postulate), but there are more subtle ways to introduce a contradictory equation as well.

Objective 4 For studying ground confluence, we extended the notion of critical pair from LCTRSs to LCSTRSs. By using the right notion of term position, we can retain the definition of critical pair of LCTRSs (Section 5.6.1). Having defined critical pairs, we have shown how to use Bounded RI to establish ground confluence of an LCSTRS, proving that all its critical pairs are bounded ground convertible.

Objective 5 With the implementation of Bounded RI in Cora (Section 5.7), and in particular the `auto` command, we can automatically perform trivial deduction steps in a RI proof, such as the application of (Simplify) with a rule in $\mathcal{R} \cup \mathcal{R}_{calc}$ or even the application of induction hypotheses with ($\mathcal{H}$ -Delete).

While the application of induction hypotheses can often be carried out mechanically, the discovery of lemmas constitutes a main challenge. Lemmas are central to RI proofs and require generalization and careful reasoning about their applicability. To address this challenge in an automatable manner, we propose two approaches:

- ▷ *Invariant generation.* Restricting attention to simple integer loop comparisons, we introduced several lemma generation heuristics based on invariant generation, aimed at producing both equalities (Section 4.2) and inequalities (Section 4.3).
- ▷ *Templates.* Templates are used to capture typical programming constructions, such as tail recursion (Section 6.1). Using LCSTRSs, we can represent very general templates via higher-order recursors (Section 6.1.1). This makes it possible to establish equivalence between non-instantiated recursors in RI, after which they can be instantiated to prove program equivalence in practice (Section 6.2).

Invariant generation. Invariant generation is an interpretation-specific problem. In this thesis, we restricted attention to integer theory terms, of which ground instances are interpreted in $\mathbb{Z}$. This allows us to use algebraic reasoning over integer polynomials to compute invariants. Although the heuristics only target simple divergence patterns, they may serve as a basis for extensions to more complex loop constructions, such as nested loops (Section 4.7). However, invariant generation should not be regarded as the holy grail for proving every integer equivalence. Non-polynomial integer relations naturally arise in real-life examples, where invariant generation is not an appropriate proof technique. These cases require an approach based on structural generalization. To this end, we introduced the template method in Chapter 6.

Templates. With the template approach, we can for instance prove an equation $s \approx t [\varphi]$, using three lemmas:

$$(1). s \approx \text{TU } f a b c [\varphi] \quad (2). \text{TU } F x y z \approx \text{RD } F x y z \quad (3). \text{RD } f a b c \approx t [\varphi]$$

In this procedure, lemma (1) and (3) are used to identify s and t as recursor instances, thereby transforming $s \approx t [\varphi]$ into a proof goal $\text{TU } f a b c \approx \text{RD } f a b c [\varphi]$, which can then be finished with the recursor equivalence expressed in lemma (2).

We introduced four templates, each represented as an LCSTRS, using a corresponding recursor. In addition, we proved six corresponding recursor equivalences inside Cora, which can be used as part of larger induction proofs. The RI-integrated approach should make it easy to formally establish equivalence between newly defined templates in Cora. In this way, we can quickly build a practically useful database of templates.

Automation. Although the heuristics for invariant generation and the template method are currently not available in an automated fashion, their implementation is largely a syntactical problem, for which we do not foresee many problems:

For matrix invariants, this involves the automatic recognition of a simple divergence pattern to extract the initial vector $\vec{e}_0$, divergence substitution χ and constrained divergence shape $\ell \approx r [\varphi]$. The actual computation of matrix invariants requires advanced algebraic computations, but can be outsourced to a computer algebra system, as discussed in Section 4.6. The generation of inequalities involves the automatic recognition of the divergence constraint φ_{div} , and possibly the computation of χ^{-1} . This inverse computation can be delegated to a computer algebra system as well, since this relies on the interpretation of χ as a multivariate polynomial mapping $\mathbb{Z}^n \rightarrow \mathbb{Z}^n$. Considering the derived invariants: the generation of these inequalities is purely syntactical, although the validity checks need to be done via SMT.

For the template method, identifying a concrete program as recursor instance amounts to solving a matching problem. The main challenge for the template method is the development of heuristics to apply one-sided matching or two-sided matching: there is no unique best approach, and different strategies might lead to different outcomes. Furthermore, to effectively utilize all available lemma generation techniques, we need to develop high-level, proof-strategic heuristics for how to combine available methods.

Overall, we believe that the remaining challenges in using our methods are well within reach through further development of Cora. Of course, effective proof-strategic heuristics remain to be developed, but once the basic techniques are in place, we can already experiment in a semi-automatic fashion, thereby gaining insights for the design of such heuristics. Therefore, this thesis establishes a promising basis for the automation of equivalence proofs.

References

- [1] W. W. Adams and P. Lounstaunau. *An Introduction to Gröbner Bases*. Vol. 3. Graduate Studies in Mathematics. American Mathematical Society, 1994. doi: 10.1090/gsm/003.
- [2] A. Aguirre et al. “A relational logic for higher-order programs”. In: *Proc. ACM Program. Lang.* 1.ICFP (2017), 21:1–21:29. doi: 10.1145/3110265.
- [3] R. Ait El Manssour, G. Kenison, M. Shirmohammadi, and A. Varonka. “Simple Linear Loops: Algebraic Invariants and Applications”. In: *PACMPL* 9.POPL (2025), 26:1–26:27. doi: 10.1145/3704862.
- [4] T. Aoto. “Dealing with Non-orientable Equations in Rewriting Induction”. In: *Proc. RTA 06*. Vol. 4098. LNCS. 2006, pp. 242–256. doi: 10.1007/11805618_18.
- [5] T. Aoto. “Designing a Rewriting Induction Prover with an Increased Capability of Non-Orientable Theorems”. In: *Proc. SCSS 08*. 2008, pp. 1–15. URL: https://www.risc.jku.at/publications/download/risc_3448/SCSS2008_Proceedings.pdf.
- [6] T. Aoto. “Soundness of Rewriting Induction Based on an Abstract Principle”. In: *Information and Media Technologies* 3.2 (2008), pp. 225–235. doi: 10.11185/IMT.3.225.
- [7] T. Aoto, N. Nishida, and J. Schöpf. “Equational Theories and Validity for Logically Constrained Term Rewriting”. In: *Proc. FSCD 24*. Vol. 299. LIPIcs. 2024, 31:1–31:21. doi: 10.4230/LIPICS.FSCD.2024.31.
- [8] T. Aoto and Y. Toyama. “Ground Confluence Prover based on Rewriting Induction”. In: *Proc. FSCD 16*. Vol. 52. LIPIcs. 2016, 33:1–33:12. doi: 10.4230/LIPICS.FSCD.2016.33.
- [9] T. Aoto, T. Yamada, and Y. Chiba. “Natural inductive theorems for higher-order Rewriting”. In: *Proc. RTA 11*. Vol. 10. LIPIcs. 2011, pp. 107–121. doi: 10.4230/LIPICS.RTA.2011.107.
- [10] T. Aoto, T. Yamada, and Y. Toyama. “Inductive Theorems for Higher-Order Rewriting”. In: *Proc. RTA 04*. Vol. 3091. LNCS. 2004, pp. 269–284. doi: 10.1007/978-3-540-25979-4_19.
- [11] T. Arts and J. Giesl. “Termination of Term Rewriting Using Dependency Pairs”. In: *TCS* 236.1–2 (2000), pp. 133–178. doi: 10.1016/S0304-3975(99)00207-8.
- [12] M. F. Atiyah and I. G. Macdonald. *Introduction to Commutative Algebra*. CRC Press, 2018. doi: 10.1201/9780429493638.
- [13] F. Baader and T. Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998. doi: 10.1017/CBO9781139172752.
- [14] C. Barrett, P. Fontaine, and C. Tinelli. *The SMT-LIB Standard: Version 2.7*. Tech. rep. Available at <http://www.SMT-LIB.org>. For QF_NIA, see https://smt-lib.org/logics-all.shtml#QF_NIA. Department of Computer Science, The University of Iowa, 2025.

- [15] E. Bayarmagnai, F. Mohammadi, and R. Prébet. “Algebraic and algorithmic methods for computing polynomial loop invariants”. In: *J. Symb. Comput.* 135 (2026), p. 102551. doi: 10.1016/j.jsc.2025.102551.
- [16] N. Benton. “Simple relational correctness proofs for static analyses and program transformations”. In: *Proc. POPL 04*. 2004, pp. 14–25. doi: 10.1145/964001.964003.
- [17] W. F. Brozius, K. Hagens, and C. Kop. *Extending a Lemma Generation Approach for Rewriting Induction on Logically Constrained Term Rewriting Systems*. Presented at WPTE 22. 2022. doi: 10.5281/zenodo.22250813.
- [18] A. Bundy, D. Basin, D. Hutter, and A. Ireland. *Rippling: Meta-Level Guidance for Mathematical Reasoning*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2005. doi: 10.1017/CBO9780511543326.
- [19] Y. Chiba, T. Aoto, and Y. Toyama. “Program Transformation Templates for Tupling Based on Term Rewriting”. In: *IEICE Transactions on Information and Systems* E93-D.5 (2010), pp. 963–973. doi: 10.1587/transinf.E93.D.963.
- [20] D. A. Cox, J. Little, and D. O’Shea. *Using Algebraic Geometry*. 2nd. Vol. 185. Graduate Texts in Mathematics. Springer, 2005. doi: 10.1007/b138611.
- [21] S. Falke and D. Kapur. “Rewriting Induction + Linear Arithmetic = Decision Procedure”. In: *Proc. IJCAR 12*. Vol. 7364. LNAI. 2012, pp. 241–255. doi: 10.1007/978-3-642-31365-3_20.
- [22] C. Fuhs, C. Kop, and N. Nishida. “Verifying Procedural Programs via Constrained Rewriting Induction”. In: *TOCL* 18.2 (2017), 14:1–14:50. doi: 10.1145/3060143.
- [23] J. Giesl, R. Thiemann, and P. Schneider-Kamp. “The Dependency Pair Framework: Combining Techniques for Automated Termination Proofs”. In: *Proc. LPAR 04*. Vol. 3452. LNCS. 2005, pp. 301–331. doi: 10.1007/978-3-540-32275-7_21.
- [24] L. Guo, K. Hagens, C. Kop, and D. Vale. “Higher-Order Constrained Dependency Pairs for (Universal) Computability”. In: *Proc. MFCS 24*. Vol. 306. LIPIcs. 2024, 57:1–57:15. doi: 10.4230/LIPIcs.MFCS.2024.57.
- [25] L. Guo and C. Kop. “Higher-Order LCTRSs and Their Termination”. In: *Proc. ESOP 24*. Vol. 14577. LNCS. 2024, pp. 331–357. doi: 10.1007/978-3-031-57267-8_13.
- [26] K. Hagens and C. Kop. “Matrix invariants for program equivalence in LCTRSs”. In: *Proc. WPTE 23*. 2023, pp. 1–13. url: <https://hdl.handle.net/2066/312790>.
- [27] K. Hagens and C. Kop. “Rewriting Induction for Higher-Order Constrained Term Rewriting Systems”. In: *Proc. LOPSTR 24*. Vol. 14919. 2024, pp. 202–219. doi: 10.1007/978-3-031-71294-4_12.
- [28] K. Hagens and C. Kop. *Bounded Rewriting Induction for LCSTRSs*. 2026. doi: 10.48550/arXiv.2601.02803.
- [29] K. Hagens and C. Kop. “Templates in Rewriting Induction”. In: *Proc. LSFA 26*. Vol. 449. EPTCS. 2026, pp. 123–140. doi: 10.4204/EPTCS.449.8.
- [30] E. Hrushovski, J. Ouaknine, A. Pouly, and J. Worrell. “Polynomial Invariants for Affine Programs”. In: *Proc. LICS 18*. 2018, pp. 530–539. doi: 10.1145/3209108.3209142.
- [31] D. Kapur and N. Sakhanenko. “Automatic generation of generalization lemmas for proving properties of tail-recursive definitions”. In: *Proc. TPHOLs 03*. Vol. 2758. LNCS. 2003, pp. 136–154. doi: 10.1007/10930755_9.
- [32] D. Kapur and M. Subramaniam. “Lemma discovery in automating induction”. In: *Proc. CADE 96*. Vol. 1104. LNCS. 1996, pp. 538–552. doi: 10.1007/3-540-61511-3_112.

- [33] H. Koike and Y. Toyama. “Inductionless Induction and Rewriting Induction”. In: *Computer Software* 17.6 (2000). (In Japanese), pp. 509–520. doi: 10.11309/jssst.17.509.
- [34] C. Kop. “CONstrained Rewriting Analyser (CORA)”. Open-source analysis tool for higher-order constrained term rewriting. 2025. URL: <https://github.com/hezzel/cora>.
- [35] C. Kop and N. Nishida. “Term Rewriting with Logical Constraints”. In: *Proc. FroCoS 13*. Vol. 8152. LNAI. 2013, pp. 343–358. doi: 10.1007/978-3-642-40885-4_24.
- [36] C. Kop and F. van Raamsdonk. “Dynamic dependency pairs for Algebraic Functional Systems”. In: *LMCS* 8.2 (2012), pp. 1–51. doi: 10.2168/LMCS-8(2:10)2012.
- [37] K. Kusakari. “On Proving Termination of Term Rewriting Systems with Higher-Order Variables”. In: *IPSJ Transactions on Programming* 42.7 (2001), pp. 35–45. URL: <https://ipsj.ixsq.nii.ac.jp/records/16864>.
- [38] L. de Moura and N. Bjørner. “Z3: An Efficient SMT Solver”. In: *Proc. TACAS 08*. Vol. 4963. LNCS. 2008, pp. 337–340. doi: 10.1007/978-3-540-78800-3_24.
- [39] N. Nakabayashi et al. “Lemma Generation Method in Rewriting Induction for Constrained Term Rewriting Systems”. In: *Computer Software* 28.1 (2011). English version available at <https://www.trs.css.i.nagoya-u.ac.jp/crisys/nakabayashi10.pdf>, pp. 173–189. doi: 10.11309/jssst.28.1_173.
- [40] N. Nishida, M. Kojima, and T. Kato. “Transforming imperative programs into bisimilar logically constrained term rewrite systems via injective functions from configurations to terms”. In: *JLAMP* 145 (2025), p. 101056. doi: 10.1016/j.jlamp.2025.101056.
- [41] S. de Oliveira, S. Bensalem, and V. Prevosto. *Polynomial invariants by linear algebra*. 2016. doi: 10.48550/arXiv.1611.07726.
- [42] A. Pitts. “Operationally-Based Theories of Program Equivalence”. In: *Semantics and Logics of Computation*. 1997, pp. 241–298. doi: 10.1017/CBO9780511526619.007.
- [43] U.S. Reddy. “Term Rewriting Induction”. In: *Proc. CADE ’90*. Vol. 449. LNCS. 1990, pp. 162–177. doi: 10.1007/3-540-52885-7_86.
- [44] J. Schöpfung and A. Middeldorp. “Confluence Criteria for Logically Constrained Rewrite Systems”. In: *Proc. CADE 23*. Vol. 14132. LNCS. 2023, pp. 474–490. doi: 10.1007/978-3-031-38499-8_27.
- [45] P. Urso and E. Kounalis. “Sound generalizations in mathematical induction”. In: *TCS* 323.1-3 (2004), pp. 443–471. doi: 10.1016/j.tcs.2004.05.022.
- [46] T. Walsh. “A Divergence Critic for Inductive Proof”. In: *JAIR* 4 (1996), pp. 209–235. doi: 10.1613/jair.275.

Samenvatting

Het garanderen dat software zich gedraagt zoals bedoeld is een centrale uitdaging in softwareontwikkeling. Formele verificatie biedt wiskundig rigoureuze methoden om te redeneren over computer code. Een van deze methoden is *termherschrijven*, waarbij de elementaire rekenstappen van een programma worden gespecificeerd door een verzameling herschrijfgeregels en het corresponderende computationele gedrag wordt vastgelegd via de herschrijfrelatie. Deze herschrijfrelatie kan worden opgevat als een taalafhankelijke operationele semantiek, die uitermate geschikt is voor het definiëren van methoden voor programma-analyse, zoals terminatie- en complexiteitsanalyse. Een tekortkoming van traditionele herschrijfsystemen is echter dat primitieve datastructuren, zoals integers, inductief gecodeerd moeten worden. Deze benadering correspondeert niet met de manier waarop dergelijke datastructuren worden verwerkt in programmeertalen. Een oplossing wordt geboden door *Logically Constrained Simply-typed Term Rewriting Systems* (LCSTRSs), waarin primitieve datastructuren semantisch kunnen worden gemodelleerd.

In deze thesis gebruiken we de hogere-orde uitbreiding van LCSTRSs, genaamd *Logically Constrained Simply-typed Term Rewriting Systems* (LCSTRSs), om *program equivalence* te bestuderen. De herschrijf relatie wordt gebruikt om een notie van equivalentie te definiëren, gebaseerd op de intuïtie dat twee programma's equivalent zijn indien zij hetzelfde input-outputgedrag vertonen. We ontwikkelen verschillende bewijssystemen om *program equivalence* in praktijk te kunnen vaststellen. Al deze systemen zijn gebaseerd op *welgefundeerde inductie* over term herschrijf systemen, die als methode afhankelijk is van terminatie, dat dus een belangrijke rol zal spelen in deze thesis.

Een van de redenen om een bewijssysteem te willen hebben is het mogelijk maken van equivalentiebewijzen via de computer. Hoewel *program equivalence* in het algemeen onbeslisbaar is (dus complete bewijs-automatisering is niet mogelijk), introduceren we heuristieken om in eenvoudigere gevallen een zekere mate van automatisering te realiseren. Daarnaast is een substantieel deel van de thesis gericht op *ground confluence*. Deze eigenschap is noodzakelijk om de correctheid van de methode voor het weerleggen van *program equivalence* te garanderen, maar is ook op zichzelf een belangrijke eigenschap: in een terminerend en *ground confluent* term herschrijf systeem berekent reductie in wezen een uniek resultaat voor elke grondterm. We ontwikkelen

een methode om ground confluence van LCSTRSs te bewijzen, en als onderdeel hiervan definiëren we kritieke paren voor LCSTRSs.

Summary

Ensuring that software behaves as intended is a central challenge in software development. Formal verification provides mathematically rigorous methods for reasoning about programming code. One such method is *term rewriting*, in which the elementary computational steps defining a program are specified by a set of rewrite rules and the corresponding program behavior is captured by the rewrite relation. The rewrite relation can be seen as a language-independent operational semantics, suitable for defining methods for program analysis, such as termination and complexity analysis. However, a limitation of traditional term rewriting systems is that primitive data structures, such as integers, must be encoded inductively. This does not correspond to the way such data structures are treated in real programming languages. A solution is provided by Logically Constrained Term Rewriting Systems (LCTRSs), where primitive data structures can be treated semantically, thereby providing a more faithful basis for reasoning about program behavior.

In this thesis, we use the higher-order extension of LCTRSs, called *Logically Constrained Simply-typed Term Rewriting Systems* (LCSTRSs), to study *program equivalence*. The rewrite relation is used to define a notion of program equivalence, based on the intuition that two programs are equivalent if they exhibit the same input-output behavior. We develop several proof systems to establish program equivalence in practice. All of them are based on *well-founded induction* over term rewriting systems, which in turn relies on termination, which will therefore play an important role in this thesis.

One of the reasons for wanting to have a proof system is to enable computer-guided equivalence proofs. Although program equivalence is undecidable in general (i.e. complete proof automation is not possible), we introduce heuristics to realize a certain degree of automation in simple cases. A substantial part of the thesis is centered around ground confluence. This property is necessary to guarantee the correctness of the method for disproving program equivalence, but it is also important in its own right: in a terminating and ground-confluent system, reduction essentially computes a unique result for every ground term. We develop a method for proving ground confluence of LCSTRSs, and as part of this, we define critical pairs for LCSTRSs.

Research Data Management

This thesis research has been carried out under the research data management policy of the Institute for Computing and Information Science of Radboud University, The Netherlands.¹ The following research code repositories have been produced during this Ph.D. research:

- Chapter 4. <https://doi.org/10.5281/zenodo.22343027>
- Chapter 5. <https://doi.org/10.5281/zenodo.22283693>
- Chapter 6. <https://doi.org/10.5281/zenodo.22304521>

¹<https://www.ru.nl/icis/research-data-management/>

Titles in the IPA Dissertation Series since 2023

S. Kochanthara. *A Changing Landscape: On Safety & Open Source in Automated and Connected Driving.* Faculty of Mathematics and Computer Science, TU/e. 2023-01

L.M. Ochoa Venegas. *Break the Code? Breaking Changes and Their Impact on Software Evolution.* Faculty of Mathematics and Computer Science, TU/e. 2023-02

N. Yang. *Logs and models in engineering complex embedded production software systems.* Faculty of Mathematics and Computer Science, TU/e. 2023-03

J. Cao. *An Independent Timing Analysis for Credit-Based Shaping in Ethernet TSN.* Faculty of Mathematics and Computer Science, TU/e. 2023-04

K. Dokter. *Scheduled Protocol Programming.* Faculty of Mathematics and Natural Sciences, UL. 2023-05

J. Smits. *Strategic Language Workbench Improvements.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2023-06

A. Arslanagić. *Minimal Structures for Program Analysis and Verification.* Faculty of Science and Engineering, RUG. 2023-07

M.S. Bouwman. *Supporting Railway Standardisation with Formal Verification.* Faculty of Mathematics and Computer Science, TU/e. 2023-08

S.A.M. Lathouwers. *Exploring Annotations for Deductive Verification.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2023-09

J.H. Stoel. *Solving the Bank, Lightweight Specification and Verification Techniques for Enterprise Software.* Faculty of Mathematics and Computer Science, TU/e. 2023-10

D.M. Groenewegen. *WebDSL: Linguistic Abstractions for Web Programming.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2023-11

D.R. do Vale. *On Semantical Methods for Higher-Order Complexity Analysis.* Faculty of Science, Mathematics and Computer Science, RU. 2024-01

M.J.G. Olsthoorn. *More Effective Test Case Generation with Multiple Tribes of AI.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2024-02

B. van den Heuvel. *Correctly Communicating Software: Distributed, Asynchronous, and Beyond.* Faculty of Science and Engineering, RUG. 2024-03

H.A. Hiep. *New Foundations for Separation Logic.* Faculty of Mathematics and Natural Sciences, UL. 2024-04

C.E. Brandt. *Test Amplification For and With Developers.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2024-05

J.I. Hejderup. *Fine-Grained Analysis of Software Supply Chains.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2024-06

J. Jacobs. *Guarantees by construction.* Faculty of Science, Mathematics and Computer Science, RU. 2024-07

O. Bunte. *Cracking OIL: A Formal Perspective on an Industrial DSL for Modelling Control Software.* Faculty of Mathematics and Computer Science, TU/e. 2024-08

R.J.A. Erkens. *Automaton-based Techniques for Optimized Term Rewriting.* Faculty of Mathematics and Computer Science, TU/e. 2024-09

J.J.M. Martens. *The Complexity of Bisimilarity by Partition Refinement.* Faculty of Mathematics and Computer Science, TU/e. 2024-10

L.J. Edixhoven. *Expressive Specification and Verification of Choreographies.* Faculty of Science, OU. 2024-11

J.W.N. Paulus. *On the Expressivity of Typed Concurrent Calculi.* Faculty of Science and Engineering, RUG. 2024-12

J. Denkers. *Domain-Specific Languages for Digital Printing Systems.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2024-13

L.H. Appelis. *Tool-Driven Quality Assurance for Functional Programming and Machine Learning.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2024-14

P. Karkhanis. *Driving the Future: Facilitating C-ITS Service Deployment for Connected and Smart Roadways.* Faculty of Mathematics and Computer Science, TU/e. 2024-15

N.W. Cassee. *Sentiment in Software Engineering.* Faculty of Mathematics and Computer Science, TU/e. 2024-16

H. van Antwerpen. *Declarative Name Binding for Type System Specifications.* Faculty

of Electrical Engineering, Mathematics, and Computer Science, TUD. 2025-01

I.N. Mulder. *Proof Automation for Fine-Grained Concurrent Separation Logic.* Faculty of Science, Mathematics and Computer Science, RU. 2025-02

T.S. Badings. *Robust Verification of Stochastic Systems: Guarantees in the Presence of Uncertainty.* Faculty of Science, Mathematics and Computer Science, RU. 2025-03

A.M. Mir. *Machine Learning-assisted Software Analysis.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2025-04

L.T. Vinkhuijzen. *Data Structures for Quantum Circuit Verification and How To Compare Them.* Faculty of Mathematics and Natural Sciences, UL. 2025-05

D. van der Wal. *What is the Point? Single-Input-Change Testing a EULYNX Controller.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2025-06

A. Rosset. *Uniform Monad Presentations and Graph Quasitoposes.* Faculty of Sciences, Department of Computer Science, VU. 2025-07

L. Guo. *Higher-Order Termination with Logical Constraints.* Faculty of Science, Mathematics and Computer Science, RU. 2025-08

D.A. Manrique Negrin. *A Model Orchestra in Digital Twins: A Model-Driven Approach to Integration and Orchestration.* Faculty of Mathematics and Computer Science, TU/e. 2025-09

C.A. Esterhuysen. *Specification-Centric Multi-Agent Systems*. Faculty of Science, UvA. 2025-10

H.M. Muctadir. *Consistency Matters: Building Consistent Digital Twin Virtual Entities*. Faculty of Mathematics and Computer Science, TU/e. 2025-11

A. Stramaglia. *Model Checking Machine-Control Applications*. Faculty of Mathematics and Computer Science, TU/e. 2025-12

M. Saeedi Nikoo. *Supporting business process management: clone detection and recommendation techniques*. Faculty of Mathematics and Computer Science, TU/e. 2025-13

R.B. Rubbens. *Bridging the Implementation Gap: Advances in Model-Based Concurrent Program Verification*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2025-14

F.I. van der Berg. *DMC Model Checker: Delta-Driven Variable-Length Next-State Generation via Recursive Compression*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2025-15

D.G. Sprokholt. *Correct Translation between Weak Memory Model Architectures*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2025-16

B. Kohlen. *Verified & Fast Probabilistic Verification: Stepwise Refinement of Probabilistic Model Checking into an IEEE 754 Floating-Point Implementation*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2026-01

D. Frolich. *Exploratory Language Development*. Faculty of Science, UvA. 2026-02

L.B. van den Haak. *Verifying Optimised Parallel Code*. Faculty of Mathematics and Computer Science, TU/e. 2026-03

A. Papotti. *Human Acceptance of Machine Generated Code Security Analysis*. Faculty of Sciences, Department of Computer Science, VU. 2026-04

N. Doorn. *O bug, where art thou? Improving software testing education*. Faculty of Science, OU. 2026-05

C.R. van der Rest. *Reusable Programming Language Components*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2026-06

T.T.P. Franken. *A Parallel Programming Perspective – Designing AuDaLa and AuDaLogic*. Faculty of Mathematics and Computer Science, TU/e. 2026-07

T.J. Smeding. *Reverse-mode Automatic Differentiation for Functional Array Languages*. Faculty of Science, UU. 2026-08

S.A. Rukmono. *Towards Automated Software Architecture Explanation: An AI-enabled method and industrial evaluation*. Faculty of Mathematics and Computer Science, TU/e. 2026-09

R.T.C. Turkenburg. *Comparing and Constructing (Co)Inductive Predicates*. Faculty of Science, Mathematics and Computer Science, RU. 2026-10

P.B.H. Tasche. *Deductive Verification Techniques for Embedded Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2026-11

W.B. Mbaka. *An Empirical Investigation Into Human Judgments of Security Threats.* Faculty of Sciences, Department of Computer Science, VU. 2026-12

S. Srinivasan. *Taming Uncertainty, Enabling Predictability: Systematic Timing Analysis of Real-Time Systems Across Computation and Communication.* Faculty of Mathematics and Computer Science, TU/e. 2026-13

S.O. Brand. *Symbolic Algorithms for the Analysis of Quantitative Systems.* Faculty of Mathematics and Natural Sciences, UL. 2026-14

D. Engel. *Understanding Binaries, Correctly: Decidability and Formal Verification in Binary Analysis.* Faculty of Science, OU. 2026-15

K. Hagens. *Rewriting Induction for Higher-Order Constrained Rewriting.* Faculty of Science, Mathematics and Computer Science, RU. 2026-16

Curriculum Vitae



Kasper Hagens was born on November 27, 1990 in Venray, the Netherlands. He entered Radboud University in 2012 to start the Bachelor's program "Science" (a combination of Chemistry, Life Science and Physics). Later, he switched to the Physics program and eventually completed a pre-master in Mathematics. In 2019, he started his Master's degree in Mathematics, which he completed in 2021. In the same year, he also started his doctorate at Radboud University under the supervision of Cynthia Kop and Herman Geuvers.

