Rewriting induction for higher-order constrained term rewriting systems

Kasper Hagens and Cynthia Kop

Radboud University Nijmegen, The Netherlands
kasper.hagens@ru.nl, c.kop@cs.ru.nl

Abstract. Logically Constrained Term Rewriting Systems (LCTRSs) provide a framework very suitable for modeling both imperative and functional languages. One may convert programs in traditional languages into LCTRSs, and then use methods from term rewriting to analyze properties such as termination or program equivalence.
In particular in functional programming, higher-order constructs arise naturally. These have been studied using *higher-order* term rewriting. The recent definition of LCSTRSs combines higher-order rewriting with logical constraints, which creates the framework to closely model functional programs, but very few methods for their analysis have thus far been defined. Here, we study program equivalence for LCSTRSs, combining the definition of *rewriting induction* for first-order constrained rewriting with insights from unconstrained higher-order equivalence analysis.

Keywords: Term Rewriting · LCSTRSs· higher-order LCTRSs· Rewriting Induction · Inductive Theorems.

1 Introduction

Consider the following two Haskell definitions of `sumfun :: (Int -> Int) -> Int -> Int` which computes the function $(f, x) \mapsto \sum_{i=0}^{x} f(i)$ for all $x \geq 0$.

```
(SFa) sumfun f x
        | x ≤ 0      = f x
        | otherwise = (f x) + (sumfun f (x - 1))
(SFb) sumfun f x = foldl (+) 0 (map f [x,x-1..0])
```

By human reasoning we know these implementations produce the same output for all inputs with $x \geq 0$. The general problem of deciding whether two arbitrary programs produce the same output, for all possible inputs that satisfy some condition, is known as *program equivalence*. This is a challenging problem, which naturally arises in software development. For example, code may be refactored for optimization purposes, to improve code maintainability, or in preparation for later updates [13]. To guarantee preservation of reliability and functionality, such transformations are expected to retain equivalence.

There is a variety of methods to prove equivalence of programs automatically, e.g. abstract interpretation [6,21], Hoare-style proof rules [10], constrained Horn clauses [1,8], and Rewriting Induction (RI) [7,9,19]. This paper builds on the latter approach. RI [22] is a proof system to prove/disprove convertibility of two terms. The idea is to translate two versions of the same program (or: program fragment) into a single term rewriting system, and use RI to prove equivalence of the terms corresponding to, for instance, the two different `sumfun` functions.

In particular, this line of work considers translations to extensions of traditional term rewriting systems with support for integers and booleans, as well as logical constraints to naturally model control flow – for example, rules like $\mathtt{sum}(x) \to x + \mathtt{sum}(x-1)\ [x > 0]$. We will focus on *Logically Constrained Term Rewriting Systems (LCTRSs)* [15], a unifying formalism that supports arbitrary theories (e.g., bitvectors, floating point numbers or integer arrays). Programs in (fragments of) imperative languages may be translated into LCTRSs automatically (see, e.g., [9,20]) and be analyzed using rewriting methods.

However, when translating *functional* programs we soon encounter the problem of higher-order constructs: functions like `foldl` and `map`, which take functions as arguments, have no counterpart in first-order term rewriting. They are, however, naturally modeled using *higher-order* term rewriting. A recent definition of *Logically Constrained Simply typed Term Rewriting Systems* (LCSTRSs) [12] combines higher-order rewriting with native support for theories and constraints. The question arises whether we can also define RI in this setting – and if so, if this is usable for program analysis.

Bringing together higher-order rewriting with (constrained) RI poses new challenges. The papers [3,4] illustrate that, already for unconstrained higher-order rewriting, it is not easy to define what equivalence of terms even means. For first-order rewriting, this notion is straightforward, but higher-order rewriting admits multiple possible definitions – each of which comes with limitations, or loses important properties which makes them harder to analyze. Thus, a core task lies in finding definitions that allow us to not only adapt RI to the higher-order setting, but also have it usable in practice.

Paper overview and contributions. After some preliminaries (Section 2), we build on the unconstrained literature to propose a basic definition of *higher-order inductive theorems* for higher-order LCTRSs (Section 3). We then extend RI for constrained TRSs to this new setting (Section 4). Unfortunately, the basic definition lacks the property of *extensibility*; to solve this, we introduce a notion of *global inductive theorems* (Section 5) and show how to make it compatible with RI (Section 6). We conclude with some thoughts on future work (Section 7).

Scientific context. Please note that the purpose of this paper is *not* to investigate equivalence in Haskell in particular: we focus on *constrained higher-order term rewriting systems*. Translating Haskell and other (functional and imperative) languages into term rewriting is a topic of active research and beyond the scope of this paper. Here, we hope to provide a foundation for a form of higher-order analysis that in the future can be used as part of a larger toolbox to analyze programs, for example in Haskell, Scala or OCaml. We have chosen the LCSTRS formalism since this is the first higher-order extension of LCTRSs, and comes with existing (fully automated) support for termination analysis. Although LCSTRSs do not support lambda-expressions (an important structure in functional programs), these can typically be encoded, and it seems likely that the theory will extend naturally when these are included in the future.

2 Preliminaries

2.1 Logically Constrained Simply Typed Rewriting Systems

We will recap LCSTRSs [12], a higher-order extension of LCTRSs. This considers *applicative* higher-order rewriting (without λ) and *first-order* constraints.

Types and terms. 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 $type_1 \rightarrow \ldots \rightarrow type_m \rightarrow sort$ with $m \geq 0$.

We assume given 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 $a :: \mathit{typeof}(a)$ for $a \in \Sigma \cup \mathcal{V}$. For a term t, let $\mathtt{Var}(t)$ be the set of variables in t. A term t is *ground* if $\mathtt{Var}(t) = \emptyset$. It is *linear* if no variable occurs more than once in t.

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

We assume that Σ is the disjoint union $\Sigma_{theory} \uplus \Sigma_{terms}$ of two sets, where $\mathit{typeof}(\mathsf{f}) \in \mathcal{T}_{theory}$ for all $\mathsf{f} \in \Sigma_{theory}$. Each $\mathsf{f} \in \Sigma_{theory}$ comes with an interpretation $[\![\mathsf{f}]\!] \in \mathcal{I}_{\mathit{typeof}(\mathsf{f})}$. For example, with a theory symbol $* :: \mathsf{int} \rightarrow \mathsf{int} \rightarrow \mathsf{int}$ its interpretation may be multiplication on $\mathbb{Z}$. Symbols in Σ_{terms} do not have an interpretation since their behavior will be defined through the rewriting system. Values are theory symbols of base type, i.e. $\mathcal{V}al = \{v \in \Sigma_{theory} \mid \mathit{typeof}(v) \in \mathcal{S}_{theory}\}$. There should be exactly one value for each element of $\mathcal{I}_\iota$ ($\iota \in \mathcal{S}_{theory}$).

The set of *theory terms* is $T(\Sigma_{theory}, \mathcal{V})$. For *ground* theory terms, we define $[\![s\ t]\!] = [\![s]\!]([\![t]\!])$, thus mapping each term of type σ to an element of $\mathcal{I}_\sigma$.

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

Example 1. In all examples in this paper, we will use $\mathcal{S}_{theory} = \{\mathsf{int}, \mathsf{bool}\}$ and $\Sigma_{theory} = \{+, -, *, <, \leq, >, \geq, =, \wedge, \vee, \neg, \mathtt{true}, \mathtt{false}\} \cup \{\mathsf{n} \mid n \in \mathbb{Z}\}$, with $+, -, *$ $:: \mathsf{int} \rightarrow \mathsf{int} \rightarrow \mathsf{int}$, $<, \leq, >, \geq, = :: \mathsf{int} \rightarrow \mathsf{int} \rightarrow \mathsf{bool}$, $\wedge, \vee :: \mathsf{bool} \rightarrow \mathsf{bool} \rightarrow \mathsf{bool}$, $\neg :: \mathsf{bool} \rightarrow \mathsf{bool}$, $\mathtt{true}, \mathtt{false} :: \mathsf{bool}$ and $\mathsf{n} :: \mathsf{int}$. We let $\mathcal{I}_{\mathsf{int}} = \mathbb{Z}$, $\mathcal{I}_{\mathsf{bool}} = \{\top, \bot\}$ and interpret all symbols as expected. We use infix notation for the binary symbols, or use $[\mathsf{f}]$ for prefix or partially applied notation (e.g., $[+]\ x\ y$ and $x + y$ are the same). The values are $\mathtt{true}, \mathtt{false}$ and all n. Theory terms are for instance $x + \mathsf{3}$, $\mathtt{true}$ and $-\mathsf{7} * \mathsf{0}$. The latter two are ground. We have $[\![-\mathsf{7} * \mathsf{0}]\!] = 0$. The theory term $x > \mathsf{0}$ is a constraint, but the theory term $(x\ y) > \mathsf{0}$ with $x :: \mathsf{int} \rightarrow \mathsf{int}$ is not, nor is $[>]\ \mathsf{0} :: \mathsf{int} \rightarrow \mathsf{bool}$ (constraints are first-order terms).

Remark 1. Most programming languages have pre-defined (non-recursive) data structures and operators, e.g. the integers with a multiplication operator $*$. This makes it possible to for instance define the factorial function without first defining multiplication. This is exactly what an LCSTRS seeks to replicate: we can think of Σ_{theory} as the set of such pre-defined operators, including constants.

Substitutions, contexts and positions. A substitution is a type-preserving mapping $\gamma : \mathcal{V} \to T(\Sigma, \mathcal{V})$. The domain of a substitution is defined as $dom(\gamma) = \{x \in \mathcal{V} \mid \gamma(x) \neq x\}$, and the image of a substitution as $im(\gamma) = \{\gamma(x) \mid x \in dom(\gamma)\}$. A substitution on 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**) $(t_0\ t_1)\gamma = (t_0\gamma)\ (t_1\gamma)$. If $M \subseteq T(\Sigma, \mathcal{V})$ then $\gamma(M)$ denotes the set $\{t\gamma \mid t \in M\}$. 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.

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. (They 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 $C[t_1, \dots, t_n]$.

For a term $t = a\ t_1 \cdots t_n$ with $a \in \Sigma \cup \mathcal{V}$ and $n \geq 0$ (all terms can be denoted this way), the set of positions $Pos(t) \subseteq \mathbb{N}^*$ is defined by: $Pos(t) = \{\epsilon\} \cup \bigcup_{i=1}^{n} \{i{\cdot}p \mid p \in Pos(t_i)\}$. We define the subterm $t|_p$ of t at position $p \in Pos(t)$ as follows: (**i**) $t|_\epsilon = t$, (**ii**) $(a\ t_1 \cdots t_n)|_{i \cdot p} = t_i|_p$. If t, s are terms and $p \in Pos(t)$ then we define $t[s]_p$ as the term obtained from t by replacing $t|_p$ by s.

Rules and reduction. A rule is an expression $\ell \to r\ [\varphi]$. Here ℓ and r are terms of the same type, ℓ has a form $\mathsf{f}\ \ell_1 \cdots \ell_k$ with $k \geq 0$ and $\mathsf{f} \in \Sigma$, φ is a constraint, and for $x \in \mathtt{Var}(r) \setminus \mathtt{Var}(\ell)$, $typeof(x) \in \mathcal{S}_{theory}$. If $\varphi = \mathtt{true}$, we may denote the rule as just $\ell \to r$. Define $LVar(\ell \to r\ [\varphi]) = \mathtt{Var}(\varphi) \cup (\mathtt{Var}(r) \setminus \mathtt{Var}(\ell))$. A substitution γ *respects* $\ell \to r\ [\varphi]$ if $\gamma(LVar(\ell \to r\ [\varphi])) \subseteq \mathcal{V}al$ and $[\![\varphi\gamma]\!] = \top$.

We assume given a set of logically constrained rewrite rules $\mathcal{R}$ such that for $\ell \to r\ [\varphi] \in \mathcal{R}$, the left-hand side ℓ is not a theory term. In addition, let $\mathcal{R}_{calc}$ be the set containing, for every $\mathsf{f} \in \Sigma_{theory} \setminus \mathcal{V}al$ with $typeof(\mathsf{f}) = \iota_1 \to \dots \to \iota_m \to \kappa$ ($\kappa \in \mathcal{S}_{theory}$) a rule $\mathsf{f}\ x_1 \cdots x_m \to y\ [y = \mathsf{f}\ x_1 \cdots x_m]$. We call these *calculation rules*. The reduction relation $\to_{\mathcal{R}}$ is defined by:

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

We say that s *has normal form* t if $s \to_{\mathcal{R}}^* t$ and t cannot be reduced.

For a fixed set of rules $\mathcal{R}$, let $\mathcal{D} = \{\mathsf{f} \in \Sigma \mid \text{there is a rule } \mathsf{f}\ \ell_1 \cdots \ell_k \to r\ [\varphi] \in \mathcal{R}\}$; we call the elements of $\mathcal{D}$ *defined symbols*, and the elements of $\mathcal{C} = \mathcal{V}al \cup (\Sigma_{terms} \setminus \mathcal{D})$ *constructors*. The elements of $\Sigma_{theory} \setminus \mathcal{V}al$ are called *calculation symbols*. A term in $T(\mathcal{C}, \mathcal{V})$ is called a constructor term. A ground constructor substitution is a substitution γ such that $im(\gamma) \subseteq T(\mathcal{C}, \emptyset)$.

For a rule $\ell \to r\ [\varphi]$ define $head(\ell \to r\ [\varphi]) = \mathsf{f}$ if ℓ is of the shape $\mathsf{f}\ \ell_1 \cdots \ell_k$. For $\mathsf{f} \in \Sigma$ define $\mathcal{R}_{\mathsf{f}} = \{\ell \to r\ [\varphi] \in \mathcal{R} \mid head(\ell) = \mathsf{f}\}$ (so $\mathcal{R}_{\mathsf{f}} = \emptyset$ if $\mathsf{f} \notin \mathcal{D}$).

A *Logically Constrained Simply-typed Term Rewriting System (LCSTRS)* is a pair $(T(\Sigma,\mathcal{V}), \to_{\mathcal{R}})$ generated by $(\mathcal{S}, \mathcal{S}_{theory}, \Sigma_{terms}, \Sigma_{theory}, \mathcal{V}, \mathit{typeof}, \mathcal{I}, [\![\cdot]\!], \mathcal{R})$. To refer to an LCSTRS we often supply just Σ and $\mathcal{R}$ and leave the rest implicit.

Example 2. Haskell function `(SFa)` can be modeled by $\mathcal{S} = \mathcal{S}_{theory} = \{\mathsf{int}, \mathsf{bool}\}$, $\Sigma_{terms} = \{\mathsf{sumfun} :: (\mathsf{int} \to \mathsf{int}) \to \mathsf{int} \to \mathsf{int}\}$, Σ_{theory} from Example 1, and

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

Then $\mathcal{D} = \{\mathsf{sumfun}\}$, and $\mathcal{C} = \mathcal{V}al = \{\mathtt{true}, \mathtt{false}\} \cup \{\mathsf{n} \mid n \in \mathbb{Z}\}$. We have $LVar((R1)) = \{x\}$ and $[x := \mathsf{0}]$ respects $(R1)$. We have $\mathsf{sumfun}\ f\ \mathsf{0} \to_{\mathcal{R}} f\ \mathsf{0}$. An example of a rewrite sequence, computing a normal form: $\mathsf{sumfun}\ ([*]\ \mathsf{2})\ \mathsf{1} \to_{(R2)}$ $[+]\ (([*]\ \mathsf{2})\ \mathsf{1})\ (\mathsf{sumfun}\ ([*]\ \mathsf{2})\ (\mathsf{1}-\mathsf{1})) \to_{\mathcal{R}_{calc}} [+]\ \mathsf{2}\ (\mathsf{sumfun}\ ([*]\ \mathsf{2})\ (\mathsf{1}-\mathsf{1})) \to_{\mathcal{R}_{calc}}$ $[+]\ \mathsf{2}\ (\mathsf{sumfun}\ ([*]\ \mathsf{2})\ \mathsf{0}) \to_{(R1)} [+]\ \mathsf{2}\ (([*]\ \mathsf{2})\ \mathsf{0}) \to_{\mathcal{R}_{calc}} [+]\ \mathsf{2}\ \mathsf{0} \to_{\mathcal{R}_{calc}} \mathsf{2}$.

2.2 Rewriting Induction

Equations. An equation is a triple $s \approx t\ [\varphi]$ with s, t terms of the same type and φ a constraint. A substitution γ respects φ if $\gamma(\mathtt{Var}(\varphi)) \subseteq \mathcal{V}al$ and $[\![\varphi\gamma]\!] = \top$. A substitution γ respects $s \approx t\ [\varphi]$ if γ respects φ and $\mathtt{Var}(s) \cup \mathtt{Var}(t) \subseteq dom(\gamma)$.

Example 3. The function `(SFb)` from the introduction is modeled as follows

$(R3).\ \mathsf{fold}\ g\ v\ \mathsf{nil} \to v$ $\quad (R6).\ \mathsf{map}\ f\ \mathsf{nil} \to \mathsf{nil}$

$(R4).\ \mathsf{fold}\ g\ v\ (h : t) \to \mathsf{fold}\ g\ (g\ v\ h)\ t$ $\quad (R7).\ \mathsf{map}\ f\ (h : t) \to (f\ h) : \mathsf{map}\ f\ t$

$(R5).\ \mathsf{init}\ n \to \mathsf{nil}\quad [n < 0]$ $\quad (R8).\ \mathsf{init}\ n \to n : \mathsf{init}\ (n-1)\quad [n \geq 0]$

Now the equivalence mentioned in the introduction is expressed as below.

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

Substitution $[n := \mathsf{0}]$ does not respect the equation, but $[n := \mathsf{0},\ f := [*]\ \mathsf{2}]$ does.

Inductive theorems. Equivalence is defined via inductive theorems. For first-order rewriting, an equation $s \approx t\ [\varphi]$ is an inductive theorem if $s\gamma \leftrightarrow^*_{\mathcal{R}} t\gamma$ for every ground substitution γ that respects this equation. Here, $\leftrightarrow_{\mathcal{R}}$ is the union $\to_{\mathcal{R}} \cup \leftarrow_{\mathcal{R}}$, and $\leftrightarrow^*_{\mathcal{R}}$ is its transitive, reflexive closure.

Rewriting Induction. Rewriting Induction (RI) is a proof system for showing that equations are inductive theorems. It proceeds by transforming proof states: pairs $(\mathcal{E}, \mathcal{H})$ where $\mathcal{E}$ is a set of equations and $\mathcal{H}$ a set of rewrite rules. For such a proof state, we can think of $\mathcal{E}$ as the set containing all current proof goals, and of $\mathcal{H}$ as the set of induction hypotheses (oriented equations) that have been assumed in the process leading to this proof state. At the start $\mathcal{E}$ consists of all equations that we want to prove to be inductive theorems, and $\mathcal{H} = \emptyset$. The goal of RI is to find a deduction sequence $(\mathcal{E}, \emptyset) \vdash^* (\emptyset, \mathcal{H})$, for some set $\mathcal{H}$.

There are subtle variations in the rules defining $\vdash$ between different first-order variants of RI (e.g. in [22,7,9]). We will not state the rules here, but provide a higher-order extension in Section 4. They typically satisfy the following property:

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.

The proof of this result relies on well-founded induction over the relation $\to_{\mathcal{R}\cup\mathcal{H}}$, and therefore the method is limited to *terminating* LCTRSs (i.e. there are no infinite reductions $s_0 \to_{\mathcal{R}} s_1 \to_{\mathcal{R}} s_2 \to_{\mathcal{R}} \ldots$). The result also relies on *quasi-reductivity*; that is, that every ground normal form is a constructor term. Quasi-reductivity ensures that evaluation on ground terms cannot get "stuck"; roughly, that pattern matching is exhaustive. Termination and quasi-reductivity together ensure that every ground term reduces to a constructor term.

3 Higher-order Inductive Theorems

While the first-order definition of inductive theorems is straightforward, it is not immediately obvious how it should be extended to a higher-order setting. In particular, the question of *extensionality* comes into play. We will present some ideas from the literature, and then posit our definition, before also extending the notions of constructor term and quasi-reductivity, which will be needed for RI.

3.1 Inductive theorems and extensionality

A first definition of higher-order inductive theorems (without constraints) appears in [4]. Aside from letting $s \approx t$ be an inductive theorem if $s\gamma \leftrightarrow^*_{\mathcal{R}} t\gamma$ for all ground substitutions γ, the authors consider two functions equivalent if they are *extensionally* equivalent: their value on all inputs is equivalent. That is, for terms s, t of type $\sigma_1 \to \ldots \to \sigma_m \to \iota$, they consider $s \approx t$ an inductive theorem if $(s\ x_1 \cdots x_m)\gamma \leftrightarrow^*_{\mathcal{R}} (t\ x_1 \cdots x_m)\gamma$. Hence, e.g. $\mathsf{map}\ ([+]\ 0) \approx \mathsf{map}\ ([*]\ 1)$ is an inductive theorem since $\mathsf{map}\ ([+]\ 0)\ l \leftrightarrow^*_{\mathcal{R}} \mathsf{map}\ ([*]\ 1)\ l$ for all ground terms l. This is intuitive since functions are often viewed in an extensional way.

Unfortunately, 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. This is illustrated by the following example:

Example 4. Let $\Sigma_{terms} = \{\mathsf{add} :: \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat},\ \mathsf{s} :: \mathsf{nat} \to \mathsf{nat},\ \mathsf{0} :: \mathsf{nat},\ \mathsf{fnil} :: \mathsf{funclist},\ \mathsf{fcons} :: (\mathsf{nat} \to \mathsf{nat}) \to \mathsf{funclist} \to \mathsf{funclist}\}$, $\Sigma_{theory} = \emptyset$ and $\mathcal{R} = \{\ \mathsf{add}\ x\ \mathsf{0} \to x\ ,\ \ \mathsf{add}\ x\ (\mathsf{s}\ y) \to \mathsf{s}\ (\mathsf{add}\ x\ y)\ \}$. Every ground term of type nat has a normal form of the shape $\mathsf{s}^n\ \mathsf{0}$, for some $n \in \mathbb{N}$, so we can restrict to ground substitutions of the shape $\gamma_n = [x := \mathsf{s}^n\ \mathsf{0}]$. Note that $(\mathsf{add}\ (\mathsf{s}\ \mathsf{0})\ x)\gamma_n \xrightarrow{*}_{\mathcal{R}} \mathsf{s}^{n+1}\ \mathsf{0} \xleftarrow{*}_{\mathcal{R}} (\mathsf{s}\ x)\gamma_n$. Hence $\mathsf{add}\ (\mathsf{s}\ \mathsf{0}) \approx \mathsf{s}$ is an extensional inductive theorem. However, for $C[\Box] = \mathsf{fcons}\ \Box\ \mathsf{fnil}$ we do *not* have $C[\mathsf{add}\ (\mathsf{s}\ \mathsf{0})] \leftrightarrow^*_{\mathcal{R}} C[\mathsf{s}]$.

The authors of [4] tackle the problem by imposing limitations on the systems they consider. Aside from other consequences (which are similar to the ones we will consider in Section 5), their restrictions essentially block constructors from taking higher-order arguments – thus for instance disallowing lists of functions as used in Example 4. 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 1 (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 $\mathcal{R} \vDash s \approx t$ 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 result on all input is equivalent.

The authors of [2] define such a semantics for first-order LCTRSs, and prove that convertibility of ground terms corresponds to their semantic notion; i.e., an equation $s \approx t\ [\varphi]$ is "CE-valid" (which corresponds to our notion of inductive theorem) if and only if $\mathcal{R} \vDash s \approx t\ [\varphi]$. However, in *higher-order* rewriting, this does not typically hold – even if we include abstraction and the η rule scheme.

A very relevant paper with regards to higher-order equivalence is [3]. This paper defines "extensional theorems" (for unconstrained higher-order rewriting) as equivalence in a model, and shows that this semantic equivalence corresponds to syntactic equivalence of ground instances of equations in an inference system. This definition solves the monotonicity problem of [4] 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 1 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 paper can be used to derive extensional theorems as well.

It is worth noting that if a system is *ground confluent* – i.e., if $s \rightarrow^*_{\mathcal{R}} t$ and $s \rightarrow^*_{\mathcal{R}} u$ for a ground term s, then there is a term w such that $t \rightarrow^*_{\mathcal{R}} w$ and $u \rightarrow^*_{\mathcal{R}} w$ – two ground terms are convertible if and only if they reduce to the same term (using $\rightarrow^*_{\mathcal{R}}$). This property is typically satisfied in LCSTRSs obtained from (deterministic) programs. Thus, in a terminating and ground confluent system, two terms are convertible if and only if they compute the same result.

3.2 Higher-order quasi-reductivity

Quasi-reductivity is an essential component of rewriting induction, 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. And Example 4, which seems relatively innocuous, even admits base-type ground non-constructor normal forms (e.g., $\mathsf{fcons}\ (\mathsf{add}\ (\mathsf{s}\ \mathsf{0}))\ \mathsf{fnil}$).

Thus, we will update our 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:
(Rule Arity) every $\mathsf{f} \in \mathcal{D}$ with $\mathsf{f} :: \sigma_1 \to \ldots \to \sigma_m \to \iota$, $\iota \in \mathcal{S}$ has a *rule arity* $k = ar(\mathsf{f}) \leq m$, meaning that every rule in $\mathcal{R}_\mathsf{f}$ is of the shape $\mathsf{f}\ l_1\ \ldots\ l_k \to r\ [\varphi]$.

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

Definition 2 (Semi-constructor terms). *Let $\mathcal{L}$ be some LCSTRS $(\Sigma, \mathcal{R})$ (leaving $\mathcal{V}$, typeof, etc. implicit). 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 $\mathsf{f} \in \Sigma$ with $\mathsf{f} :: \sigma_1 \to \ldots \to \sigma_m \to \iota$, $\iota \in \mathcal{S}$ and $s_1 :: \sigma_1, \ldots, s_n :: \sigma_n \in \mathcal{SCT}_\mathcal{L}$ with $n \leq m$, then $\mathsf{f}\ s_1 \cdots s_n \in \mathcal{SCT}_\mathcal{L}$ if: **(ii.a)** $\mathsf{f} \in \mathcal{C}$, or **(ii.b)** $\mathsf{f} \in \mathcal{D}$ and $n < ar(\mathsf{f})$, or **(ii.c)** $\mathsf{f} \in \Sigma_{theory} \setminus \mathcal{D}$ and $n < m$. The set $\mathcal{SCT}^\emptyset_\mathcal{L}$ 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 $\mathsf{fcons}\ (\mathsf{add}\ (\mathsf{s}\ \mathsf{0}))\ \mathsf{fnil}$, are all (ground) semi-constructor terms as well. Non-examples of semi-constructor terms are for instance $[+]\ \mathsf{0}\ y$ and $\mathsf{fcons}\ (\mathsf{add}\ (\mathsf{add}\ \mathsf{0}\ \mathsf{0}))$. Using (ii.b) we easily obtain:

Lemma 1. *Every semi-constructor term is in 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 $\mathcal{SCT}^\emptyset_\mathcal{L}$.

Definition 3 (Quasi-reductivity). *An LCSTRS $\mathcal{L} = (\Sigma, typeof, \mathcal{R})$ is quasi-reductive if for every $t \in T(\Sigma, \emptyset)$ we have $t \in \mathcal{SCT}^\emptyset_\mathcal{L}$ or t reduces with $\to_\mathcal{R}$.*

Note that for quasi-reductive LCSTRSs, we can limit the substitutions in Definition 1 to substitutions such that $im(\gamma) \subseteq \mathcal{SCT}^\emptyset_\mathcal{L}$, without loss of generality. We call such a substitution a *ground semi-constructor substitution.*

4 Higher-order Rewriting Induction

We will now give the derivation rules for higher-order RI. These are obtained from first-order RI [22,19,7,15,9], and adapted to the higher-order setting. We particularly build on RI, as defined in [15,9]. As a running example, we will use the LCSTRS and equation from Example 3. Thus, we start with the proof state:

$$(\mathcal{E}_0, \emptyset) := (\{\ (\mathrm{A})\ \mathsf{sumfun}\ f\ n\ \approx\ \mathsf{fold}\ [+]\ \mathsf{0}\ (\mathsf{map}\ f\ (\mathsf{init}\ n))\quad [n \geq 0]\ \}\quad ,\quad \emptyset)$$

and show that we can derive some proof state $(\emptyset, \mathcal{H})$. Theorem 1, presented in Section 4.4, will guarantee the correctness of this procedure.

4.1 Simplifying equations

The first, core rule of rewriting induction is to rewrite an equation using a rule in $\mathcal{R} \cup \mathcal{R}_{calc} \cup \mathcal{H}$. We view an equation with variables as a way to represent all *ground semi-constructor instances* of that equation. Hence, we can use "constrained term reduction" [15], which takes the constraint of the equation into account.

(Simplification) Let $\ell \to r\ [\varphi] \in \mathcal{R} \cup \mathcal{R}_{calc} \cup \mathcal{H}$ with C a context, δ a substitution such that $\delta(LVar(\ell \to r\ [\varphi])) \subseteq \mathcal{V}al \cup \mathtt{Var}(\psi)$, and $C[\ell\delta] \approx t\ [\psi]$ an equation. If the implication $\psi \Longrightarrow \varphi\delta$ is valid, then

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

There is an analogous Simplification rule to apply a rewrite rule to the right-hand side of an equation.

Example 5. Starting with our running example, simplifying on the right-hand side of (A) using rule (R8) yields $(\mathcal{E}_0, \emptyset) \vdash (\mathcal{E}_1, \emptyset)$ with $\mathcal{E}_1 = \{$(B) $\mathsf{sumfun}\ f\ n \approx \mathsf{fold}\ [+]\ \mathsf{0}\ (\mathsf{map}\ f\ (n : (\mathsf{init}\ (n-1))))\ [n \geq \mathsf{0}]\ \}$. Using subsequent Simplification steps with (R7) and (R4), we have $(\mathcal{E}_1, \emptyset) \vdash^* (\mathcal{E}_3, \emptyset)$ with $\mathcal{E}_3 = \{$(C) $\mathsf{sumfun}\ f\ n \approx \mathsf{fold}\ [+]\ (\mathsf{0} + (f\ n))\ (\mathsf{map}\ f\ (\mathsf{init}\ (n-1)))\ [n \geq \mathsf{0}]\ \}$.

4.2 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. Our second rule allows us to do a case analysis and create an induction hypothesis at the same time.

(Expansion) Let $s \approx t\ [\varphi]$ be an equation and $p \in Pos(s)$ a position such that $s|_p = \mathsf{f}\ s_1 \cdots s_n$ with $\mathsf{f} \in \mathcal{D}$, $n \geq k = ar(\mathsf{f})$ and every argument s_i is a semi-constructor term. Suppose that $\mathcal{R} \cup \mathcal{H} \cup \{s \to t\ [\varphi]\}$ is terminating. Then

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

where $Expd(s \approx t\ [\varphi], p)$ is the set:

$$\{s[r\ s_{k+1} \cdots s_n]_p\gamma \approx t\gamma\ [(\varphi\gamma) \wedge (\psi\gamma)] \mid \ell \to r\ [\psi] \in \mathcal{R}, \mathrm{mgu}(\mathsf{f}\ s_1 \cdots s_k, \ell) = \gamma\}$$

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

Example 6. In our 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:

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

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

Viewing an equation as a way to represent a set of ground equations, this definition essentially allows us to split this set into multiple subsets, by considering the possible instances at position p of s. Since we have assumed quasi-reductivity, some rule is applicable at position p of $s\gamma$ for any ground semi-constructor substitution γ. The set $Expd(s \approx t\ [\varphi], p)$ contains a representative result equation for any of the rules that might have been chosen.

Note that, after applying Expansion, the equation $s \approx t\ [\varphi]$ 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 $s|_p$.

4.3 Altering (and generalizing) equations

In Example 6, intuitively we should be able to simplify $\mathsf{init}\ (n-1)$ in the first equation of $\mathcal{E}_4$ to nil, as the constraint implies $n - 1 < 0$. However, the Simplification rule does not allow this: the variable in the init rule must be instantiated by a value or variable. Nor can we apply Simplification with a calculation rule.

Of course we could adapt the definition of Simplification, but this 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) Suppose 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 $t\gamma = v\delta$. Then

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

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 refer to the deduction step as **(Alter)** instead.

Example 7. Since $n \geq 0 \wedge n \leq 0$ is logically equivalent to $-1 = n-1$, we can use Alter to replace (D) by $f\ n \approx\ \mathsf{fold}\ [+]\ (0{+}(f\ n))\ (\mathsf{map}\ f\ (\mathsf{init}\ (n{-}1)))\ [-1 = n-1]$. Then we can use Simplification using the calculation rule $x-y \to z\ [z = x-y]$ and substitution $\gamma = [x := n, y := 1, z := -1]$ to obtain

$f\ n \approx\ \mathsf{fold}\ [+]\ (0 + (f\ n))\ (\mathsf{map}\ f\ (\mathsf{init}\ (-1)))\ [-1 = n-1]$.

Similarly, since $n \geq 0 \wedge n > 0 \iff \exists m[n > 0 \wedge m = n-1]$, we can change the constraint of (E) to $[n > 0 \wedge m = n-1]$, and use two calculation steps to obtain:

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

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

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

Neither Simplification nor Expansion can be applied to (F), and it is not obviously removable (see Section 4.4) since f could be instantiated by anything in $\mathcal{SCT}^{\emptyset}_{\mathcal{L}}$. However, using Generalize, we can replace (F) by: $x \approx (\mathsf{0}+x)$. Then, using Alter and Simplification with a calculation rule, we obtain: (H) $x \approx x\ [x = \mathsf{0}+x]$.

Discussion. Despite their similarity, the Alter and Generalize rules are used in very different ways. Alter is more innocent: replacing an equation by an equivalent one cannot harm the proof process – in contrast with 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 Simplification step with a calculation as done in Example 7. 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 equi-satisfiable one)* $s = u$ and $t = v$ and $(\exists\vec{x}.\varphi) \iff (\exists\vec{y}.\psi)$ is valid, where $\{\vec{x}\} = \mathtt{Var}(\varphi) \setminus (\mathtt{Var}(s) \cup \mathtt{Var}(t))$ and $\{\vec{y}\} = \mathtt{Var}(\psi) \setminus (\mathtt{Var}(u) \cup \mathtt{Var}(v))$. (This is particularly done before a Simplification 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 $\mathtt{Var}(\varphi)$, and $\varphi \Longrightarrow \bigwedge_{i=1}^{n} x_i = y_i$ is valid. For example, replacing $\mathsf{f}\ n \approx \mathsf{f}\ \mathsf{0}\ [n = \mathsf{0}]$ by $\mathsf{f}\ \mathsf{0} \approx \mathsf{f}\ \mathsf{0}\ [n = \mathsf{0}]$. (This 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 $x_1, \dots, x_n$ are variables in s, u that do not occur in φ, but whose type is a theory sort $\iota \in \mathcal{S}_{theory}$ such that no constructors of a type $\sigma_1 \to \ldots \to \sigma_m \to \iota$ 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 Section 4.4, 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. Generalization is also useful to abstract away from variable applications, as done for $f\ n$ in Example 7.

4.4 Finishing up

Thus far, we have only modified equations; to remove them from $\mathcal{E}$, we can use:

(Deletion) Let $s \approx t\ [\varphi]$ be such that either $s = t$ or φ is unsatisfiable, then

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

Example 8. We apply Deletion to (H) and obtain $(\mathcal{E}_6, \mathcal{H}_1)$ with $\mathcal{E}_6 = \{(\mathrm{G})\}$.

We have now defined all the inference rules necessary to complete our running example. The process is *mostly* straightforward; we detail only the harder steps. First, in equation (G), we use Simplification with the induction rule (C') to get:

$$\begin{array}{c}(f\ n)\ +\ (\mathsf{fold}\ [+]\ (\mathsf{0} + (f\ m))\ (\mathsf{map}\ f\ (\mathsf{init}\ (m-1))))\ \approx \\ \mathsf{fold}\ [+]\ ((\mathsf{0} + (f\ n)) + (f\ m))\ (\mathsf{map}\ f\ (\mathsf{init}\ (m-1)))\ [n > \mathsf{0} \wedge m = n-1]\end{array}$$

After Alter and a calculation step, we arrive at:

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

Now, we could continue to do Expansions, but doing so would result in a loop: none of the induction rules this process generates ends up being applicable. To avoid this issue, we instead use Generalize to obtain:

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

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

$x + (\mathsf{fold}\ [+]\ y\ l) \to \mathsf{fold}\ [+]\ z\ l\ [z = x + y]$

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.

Although we did not need it for our running example, we also adapt the Constructor rule of [9] because it changes in the higher-order setting.

(Semi-constructor) Let $\vec{s} = s_1 \dots s_n$ and $\vec{t} = t_1 \dots t_n$ be terms with $n > 0$. If c is either a variable, a constructor, a defined symbol with $ar(\mathsf{c}) > n$ or a non-defined calculation symbol of type $\sigma_1 \to \dots \to \sigma_m \to \iota$ with $m > n$ then

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

Example 9. In an extension of the LCSTRS of Example 2 with extra symbols, we could deduce $(\{\mathsf{fold}\ \mathsf{g}\ (\mathsf{h}\ 0\ x) \approx \mathsf{fold}\ \mathsf{h}\ (\mathsf{g}\ 0\ x)\}, \emptyset) \vdash (\{\mathsf{g} \approx \mathsf{h},\ \mathsf{h}\ 0\ x \approx \mathsf{g}\ 0\ x\}, \emptyset)$.

Theorem 1. *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 Theorem 1 (in Appendix A) follows the same outline as in first-order RI [14,9]. It proceeds by showing that certain properties are invariant through every proof step, and uses an induction on $\to_{\mathcal{R} \cup \mathcal{H}}$ to prove $\leftrightarrow_{\mathcal{E}} \subseteq \leftrightarrow_{\mathcal{R}}$.

4.5 Comparison to the first-order literature

Surprisingly few changes were needed to adapt the first-order definitions in [9] to the higher-order setting. The most important changes are the new definitions of quasi-reductivity and semi-constructor terms. The proof was also adapted to take these changes into account, but its overall structure remains the same.

The most significant change compared to [9] could already have been made in the first-order setting: the introduction of Alter, and updating Generalize to quantify over ground *semi-constructor* substitutions, rather than *all* substitutions. In [9], scenario I was combined with Simplification, and a separate rule (Eq-Deletion) was used to handle II, but III was not supported – thus leaving it impossible to prove for instance $\mathsf{init}\ (n + 1) \approx \mathsf{init}\ (1 + n)$ 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)), 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.

5 Global induction theorems

A very desirable property we do not yet have 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 terms 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 (see, e.g., [5]). The authors of [3] give the following example to illustrate the issue:

Example 10. Let $\Sigma_{terms} = \{\mathsf{zero} :: (\mathsf{nat} \to \mathsf{nat}) \to \mathsf{nat},\ \mathsf{add} :: \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat},\ \mathsf{s} :: \mathsf{nat} \to \mathsf{nat},\ \mathsf{0} :: \mathsf{nat}\}$, $\Sigma_{theory} = \emptyset$ and let $\mathcal{R}$ consist of

$$\mathsf{add}\ \mathsf{0}\ y \to y \qquad \mathsf{add}\ (\mathsf{s}\ x)\ y \to \mathsf{s}\ (\mathsf{add}\ x\ y) \qquad \mathsf{zero}\ \mathsf{s} \to \mathsf{0}$$

Then $\mathsf{add}\ x\ y \approx \mathsf{add}\ y\ x$ is an inductive theorem, since $(\mathsf{add}\ x\ y)\gamma \leftrightarrow^*_{\mathcal{R}} (\mathsf{add}\ y\ x)\gamma$ holds for any ground substitution γ. However, if we introduce a new defined symbol $\mathsf{f} :: \mathsf{nat} \to \mathsf{nat}$ and rule $\mathsf{f}\ x \to \mathsf{0}$ then $\mathsf{add}\ x\ y \approx \mathsf{add}\ y\ x$ is not an inductive theorem since $\mathsf{add}\ \mathsf{0}\ (\mathsf{zero}\ \mathsf{f}) \leftrightarrow^*_{\mathcal{R}'} \mathsf{add}\ (\mathsf{zero}\ \mathsf{f})\ \mathsf{0}$ does not hold.

A key problem in Example 10 is that the extension breaks quasi-reductivity: by importing f we create a missing pattern in $\mathcal{R}_{\mathsf{zero}}$. This is caused by *pattern matching on a function*: the last rule matches on the expression s which is a non-variable term of type $\mathsf{nat} \to \mathsf{nat}$. 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 4 (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* $\mathtt{Var}(s)$ *contains only variables of higher type.*

Definition 5 (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 $\mathsf{f}\ s_1 \cdots s_n$ with $\mathsf{f} \in \mathcal{D}$, $n \geq ar(\mathsf{f})$ and each s_i a CHV term over $\mathcal{L}$ reduces with $\to_{\mathcal{R}}$.*

Any strong quasi-reductive LCSTRS is also quasi-reductive (see Appendix B). 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) the *quasi-reducibility* notion in [3]. On the other hand, strong quasi-reductivity is weaker than the notion of higher-order sufficient completeness (HSC) in [4].

Unfortunately, limiting interest to strong quasi-reductive systems does not suffice to obtain extensibility:

Example 11. Let $\Sigma_{terms} = \{\mathsf{a} :: \mathsf{A},\ \mathsf{b} :: \mathsf{A},\ \mathsf{c} :: \mathsf{C},\ \mathsf{f} :: (\mathsf{C} \to \mathsf{A}) \to \mathsf{A},\ \mathsf{g} :: \mathsf{C} \to \mathsf{A}\}$, $\Sigma_{theory} = \emptyset$ and consider the LCSTRS with rules $\mathsf{f}\ F \to F\ \mathsf{c}$ and $\mathsf{g}\ x \to \mathsf{b}$. Then $\mathsf{f}\ F \approx \mathsf{b}$ is an inductive theorem, since the only ground term that can instantiate F is g, and indeed $\mathsf{f}\ \mathsf{g} \to_{\mathcal{R}} \mathsf{g}\ \mathsf{c} \to_{\mathcal{R}} \mathsf{b}$. However, this is not an inductive theorem if we extend the signature with a defined symbol $\mathsf{h} :: \mathsf{C} \to \mathsf{A}$ and rule $\mathsf{h}\ x \to \mathsf{a}$.

Here, $\mathsf{f}\ F \approx \mathsf{b}$ is a (naive) inductive theorem because of a *global* reasoning over the original signature: there is only one possible instance of F. This is of course no longer true in the extension. To avoid examples like this, we follow the approach of [3] and directly define a kind of inductive theorems that are preserved under extensions – provided they satisfy reasonable restrictions:

Definition 6 (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:*

- *$\mathcal{S}' \supseteq \mathcal{S}$ and $\mathcal{S}'_{theory} \supseteq \mathcal{S}_{theory}$ and $\Sigma'_{theory} \supseteq \Sigma_{theory}$ and $\Sigma'_{terms} \supseteq \Sigma_{terms}$ and $\mathcal{V}' \supseteq \mathcal{V}$ and $\mathcal{R}' \supseteq \mathcal{R}$*
- *$\mathcal{I}'_\iota = \mathcal{I}_\iota$ for all $\iota \in \mathcal{S}_{theory}$, and $[\![\mathsf{f}]\!]' = [\![\mathsf{f}]\!]$ for all $\mathsf{f} \in \Sigma_{theory}$*
- *for all $a \in \Sigma \cup \mathcal{V}$: $\mathit{typeof}'(a) = \mathit{typeof}(a)$*
- *for all $\mathsf{f} \in \Sigma$: $\mathcal{R}'_{\mathsf{f}} = \mathcal{R}_{\mathsf{f}}$ (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)*
- *for all $\mathsf{f} :: \sigma_1 \to \ldots \to \sigma_m \to \iota \in \Sigma'$, all i: there is a ground term of type σ_i*
- *for all constructor symbols $\mathsf{c} :: \sigma_1 \to \ldots \to \sigma_m \to \iota \in \mathcal{C}' \setminus \mathcal{C}$ we have $\iota \notin \mathcal{S}$*

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

Definition 7 (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: $s\gamma \leftrightarrow^*_{\mathcal{R}'} t\gamma$.*

6 Global rewriting induction

We now aim to extend higher-order RI 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), but 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 8 (Oracle extension). *An* Oracle extension *of an LCSTRS $\mathcal{L} = (\Sigma, \mathcal{R})$ is a 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 $\mathsf{f}\ v_1 \cdots v_m \to w$ where $\mathsf{f} :: \sigma_1 \to \ldots \to \sigma_m \to \iota$ ($\iota \in \mathcal{S}'$), all v_i ground, and each w is a ground semi-constructor term over $\mathcal{Q}$ that contains no defined symbols of $\mathcal{R}$. Moreover, $\mathcal{Q}$ is quasi-reductive and terminating.*

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 essentially 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 $\mathsf{f}\ v_1 \cdots v_m$, and including $\mathsf{f}\ v_1 \cdots v_m \to w$ as a rule. To ensure that the right-hand sides of the oracle rules do not use the defined symbols of $\mathcal{R}$, we also include copies versions of these defined symbols, and corresponding rules. The full construction is in Appendix C.1.

Lemma 2. *An equation $s \approx t\ [\varphi]$ over a terminating, strong quasi-reductive LCSTRS is a global inductive theorem of $\mathcal{L}$ if for every Oracle extension $\mathcal{Q}$ and ground substitution γ over $\mathcal{Q}$ that respects this equation: $s\gamma \leftrightarrow^*_{\mathcal{R}^{\mathcal{Q}}} t\gamma$.*

We now update higher-order RI in such a way that we can prove global inductive theorems of terminating, strong quasi-reductive LCSTRSs. Since the only function symbols occurring in equations and rules are those in the original signature, the Simplification rule is unchanged. The Deletion and Semi-constructor rules are also the same. For the Alter and Generalize rule, we now quantify over all ground semi-constructor 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 $p \in Pos(s)$ a position such that $s|_p = \mathsf{f}\ s_1 \cdots s_n$ with $\mathsf{f} \in \mathcal{D}$, $n \geq k = ar(\mathsf{f})$ and for all $1 \leq i \leq k$, $q \in Pos(s_i)$: if $s_i|_q$ has base type and is not a variable, then $s_i|_q$ has a form $\mathsf{c}\ t_1 \cdots t_m$ with c a constructor symbol. If $\mathcal{R}^{\mathcal{Q}} \cup \mathcal{H} \cup \{s \to t\ [\varphi]\}$ is terminating for every Oracle extension $\mathcal{Q}$ of $\mathcal{L}$ then

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

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 [18,17] (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 [11]. 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.

6.1 Soundness result

We let "global rewriting induction" be the proof process obtained from the Simplification, Deletion, Semi-constructor and updated Generalization and Alter rules, along with Global Expansion. We then obtain the main result:

Theorem 2. *Let $\mathcal{L}$ be a terminating, strong quasi-reductive LCSTRS and let $\mathcal{E}$ be a set of equations. If, by global rewriting induction, $(\mathcal{E}, \emptyset) \vdash^* (\emptyset, \mathcal{H})$, for some set $\mathcal{H}$, then every equation in $\mathcal{E}$ is a global inductive theorem of $\mathcal{L}$.*

The proof of Theorem 2 (Appendix C) follows a very similar outline as the soundness proof of Theorem 1: for an arbitrary Oracle extension $\mathcal{Q}$ of $\mathcal{L}$ we use induction on $\rightarrow_{\mathcal{R}^{\mathcal{Q}} \cup \mathcal{H}}$ to prove that $\leftrightarrow_{\mathcal{E}} \subseteq \leftrightarrow^*_{\mathcal{R}^{\mathcal{Q}}}$. Using this and Lemma 2 we find that the same holds for any quasi-reductive and terminating natural extension. Compared to the proof in Appendix A the most important changes are:

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

7 Discussion and future work

In this paper we proposed two variations of higher-order rewriting induction for constrained term rewriting systems. This includes two adaptations of inductive theorems, based on quasi-reductivity for higher-order LCSTRSs, and in the latter case, also on extensibility. We do not claim that the proof system is finished, but it provides a solid foundation for further work.

An obvious extension is to use rewriting induction not just to prove that equations are (global) inductive theorems, but also to prove that they are not. The mechanism for this exists [9] (in first-order RI) and we do not foresee major issues. It uses an additional flag in proof states to keep track of when we are allowed to derive non-equivalence (since the Generalize rule sometimes creates unsolvable equations). This extension does require *ground confluence* (as defined in Section 3.1). A new challenge is whether we can also prove that something is not a global inductive theorem, even if it *is* an inductive theorem.

A second idea is to admit extensionality; that is, to allow a rule (or: induction rule) $s\ x_1 \cdots x_n \approx t\ x_1 \cdots x_n\ [\varphi]$ to be used to reduce a term $C[s\gamma]$. If we restrict constructors to have base-type arguments, we postulate that such a deduction rule is also sound in our setting (perhaps under additional restrictions like ground confluence). It could also be used in an alternative rewriting induction approach designed for proving *extensional inductive theorems* following [3].

A very useful extension could be to weaken the termination requirement. In many cases, an obvious lemma cannot be used because the resulting induction rule would not be terminating. We postulate that, under reasonable restrictions, termination of such a rule is unnecessary if the rule is always followed by Deletion.

Related, our current definitions only support finite data. Using *coinduction* rather than *induction* may allow us to consider systems with streams, and replace the termination requirement by one of productivity.

We intend to implement rewriting induction in our tool Cora, which already supports (Oracle) termination. Fully automatic proof search could build on the ideas in [16], but will require more work on automatic strategies.

References

1. Angelis, E.d., Fioravanti, F., Pettorossi, A., Proietti, M.: Relational verification through Horn clause transformation. In: Proc. SAS 16. LNPSE, vol. 9837, pp. 147–196 (2016). https://doi.org/10.1007/978-3-662-53413-7_8
2. Aoto, T., Nishida, N., Schöpf, J.: Equational theories and validity for logically constrained term rewriting. In: Proc. FSCD 24. LIPIcs, vol. 299, pp. 31:1–31:21 (2024). https://doi.org/10.4230/LIPICS.FSCD.2024.31
3. Aoto, T., Yamada, T., Chiba, Y.: Natural inductive theorems for higher-order rewriting. In: Proc. RTA 11. LIPIcs, vol. 10, pp. 107–121 (2011). https://doi.org/10.4230/LIPIcs.RTA.2011.107
4. Aoto, T., Yamada, T., Toyama, Y.: Inductive theorems for higher-order rewriting. In: Proc. RTA 04. LNCS, vol. 3091, pp. 269–284 (2004). https://doi.org/10.1007/978-3-540-25979-4_19
5. Chiba, Y., Aoto, T., Toyama, Y.: Program transformation templates for tupling based on term rewriting. IEICE TRANSACTIONS on Information and Systems **E93-D**(5), 963–973 (2010). https://doi.org/10.1587/transinf.E93.D.963
6. Delmas, D., Miné, A.: Analysis of software patches using numerical abstract interpretation. In: Proc. SAS 19. LNPSE, vol. 11822, pp. 225–246 (2019). https://doi.org/10.1007/978-3-030-32304-2_12
7. Falke, S., Kapur, D.: Rewriting induction + linear arithmetic = decision procedure. In: Proc. IJCAR 12. LNAI, vol. 7364, pp. 241–255 (2012). https://doi.org/10.1007/978-3-642-31365-3_20
8. Felsing, D., Grebing, S., Klebanov, V., Rümmer, P., Ulbrich, M.: Automating regression verification. In: Proc. ASE 14. pp. 349–360. ACM (2014). https://doi.org/10.1145/2642937.2642987
9. Fuhs, C., Kop, C., Nishida, N.: Verifying procedural programs via constrained rewriting induction. ACM Transactions On Computational Logic (TOCL) **18**(2), 14:1–14:50 (2017). https://doi.org/10.1145/3060143
10. Godlin, B., Strichman, O.: Inference rules for proving the equivalence of recursive procedures. Acta Informatica **45**(6), 403–439 (2008). https://doi.org/10.1007/s00236-008-0075-2
11. Guo, L., Hagens, K., Kop, C., Vale, D.: Higher-order constrained dependency pairs for (universal) computability. In: Proc. MFCS 24 (2024). https://doi.org/10.48550/arXiv.2406.19379, to Appear
12. Guo, L., Kop, C.: Higher-order LCTRSs and their termination. In: Proc. ESOP 24. LNCS, vol. 14577, pp. 331–357 (2024). https://doi.org/10.1007/978-3-031-57267-8_13
13. Huth, M., Ryan, M.D.: Modelling and reasoning about systems. Cambridge University Press (2004). https://doi.org/10.1017/CBO9780511810275
14. Koike, H., Toyama, Y.: Inductionless induction and rewriting induction. Computer Software **17**(6), 509–520 (2000). https://doi.org/10.11309/jssst.17.509, (In Japanese)
15. Kop, C., Nishida, N.: Term rewriting with logical constraints. In: Proc. FroCoS 13. LNAI, vol. 8152, pp. 343–358 (2013). https://doi.org/10.1007/978-3-642-40885-4_24
16. Kop, C., Nishida, N.: ConsTrained Rewriting tooL. In: Proc. LPAR 15. LNCS, vol. 9450, pp. 549–557 (2015). https://doi.org/10.1007/978-3-662-48899-7_38
17. Kop, C., Raamsdonk, F.v.: Dynamic dependency pairs for algebraic functional systems. LMCS **8**(2), 1–51 (2012). https://doi.org/10.2168/LMCS-8(2:10)2012

18. Kusakari, K.: On proving termination of term rewriting systems with higher-order variables. IPSJ Transactions on Programming **42**(7), 35–45 (2001), http://id.nii.ac.jp/1001/00016864/
19. Nakabayashi, N., Nishida, N., Kusakari, K., Sakabe, T., Sakai, M.: Lemma generation method in rewriting induction for constrained term rewriting systems. Computer Software **28**(1), 173–189 (2010), https://www.trs.css.i.nagoya-u.ac.jp/crisys/nakabayashi10.pdf
20. Nishida, N., Kojima, M., Kato, T.: On transforming imperative programs into logically constrained term rewrite systems via injective functions from configurations to terms. In: Proc. WPTE 22 (2022), https://wvvw.easychair.org/publications/preprint_download/DbM2
21. Partush, N., Yahav, E.: Abstract semantic differencing via speculative correlation. In: Proc. OOPSLA 14. pp. 811–828. ACM (2014). https://doi.org/10.1145/2660193.2660245
22. Reddy, U.: Term rewriting induction. In: Proc. CADE '90. LNCS, vol. 449, pp. 162–177 (1990). https://doi.org/10.1007/3-540-52885-7_86

A Proofs for Section 4

This appendix is split over three parts:

- In Appendix A.1, we prove the claim made in the text that Alter may be applied in Scenarios I–III (on page 11).
- Appendix A.2 and Appendix A.3 together provide the proof of Theorem 1.
  - In Appendix A.2, we explain the proof strategy, which relies on an invariant on $\mathcal{E}$ and $\mathcal{H}$ being preserved throughout the proof process.
  - In Appendix A.3 we show that this invariant is indeed preserved in every derivation step.

The proof of Theorem 1 is quite elaborate, but not very new: the proof barely differs from its first-order counterpart in [9]. Nevertheless, we supply it here to show that indeed the proof goes through.

A.1 Alter scenarios

In Section 4.3 we claimed that we often use Alter in the scenarios I, II and III. Here, we will prove that each of them satisfies the requirements of the Alter rule.

Lemma 3 (Scenario I: replacing a constraint by an equi-satisfiable 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 $\{\vec{x}\} = \mathtt{Var}(\varphi) \setminus (\mathtt{Var}(s) \cup \mathtt{Var}(t))$ and $\{\vec{y}\} = \mathtt{Var}(\psi) \setminus (\mathtt{Var}(s) \cup \mathtt{Var}(t))$. 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 $\vec{z} = (\mathtt{Var}(\varphi) \cup \mathtt{Var}(\psi)) \setminus \{\vec{x}, \vec{y}\}$, 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. $\gamma(\mathtt{Var}(\varphi)) \subseteq \mathcal{V}al$, $[\![\varphi\gamma]\!] = \top$ and $\mathtt{Var}(s) \cup \mathtt{Var}(t) \subseteq dom(\gamma)$. Let $a_i := \gamma(x_i)$, and for $1 \leq j \leq k$: if $z_j \in \mathcal{V}(\varphi)$ let $c_j := \gamma(z_j)$, otherwise let c_j be an arbitrary value in $\mathcal{I}_{typeof(z_j)}$ (since we assumed that the sets $\mathcal{I}_\iota$ are non-empty, this can always be done). Moreover, let $\eta := [w := \gamma(w) \mid w \in (\mathtt{Var}(s) \cup \mathtt{Var}(t)) \setminus \mathtt{Var}(\varphi)]$. Then $dom(\eta) \cap \{\vec{z}\} = \emptyset$.

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 $\varphi[y_1 := b_1, \dots, y_m := b_m, z_1 := c_1, \dots, z_k := c_k]$ is valid. Since $\vec{y} \cap (\mathtt{Var}(s) \cup \mathtt{Var}(t)) = \emptyset$, we can define $\delta := \eta \cup [\vec{y} := \vec{b}, \vec{z} := \vec{c}]$.

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

Lemma 4 (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 $\texttt{Var}(\varphi)$. Assume that $\varphi \Longrightarrow \bigwedge_{i=1}^{n} x_i = y_i$ 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. $\gamma(\texttt{Var}(\varphi)) \subseteq \mathcal{V}al$, $[\![\varphi\gamma]\!] = \top$ and $\texttt{Var}(s) \cup \texttt{Var}(t) \subseteq dom(\gamma)$. Since all x_i, y_i are values or variables in φ, by definition of *respects* we know that each $x_i\gamma$ and $u_i\gamma$ is a value. By validity of $\varphi \Longrightarrow \bigwedge_{i=1}^{n} x_i = y_i$, and the fact that $[\![\varphi\gamma]\!] = \top$, we obtain that $[\![x_i\gamma = y_i\gamma]\!] = \top$ 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 $\delta = \gamma$. □

Lemma 5 (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, u that do not occur in φ, but whose type is a theory sort $\iota \in \mathcal{S}_{theory}$ such that no constructors of a type $\sigma_1 \to \ldots \to \sigma_m \to \iota$ 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. $\gamma(\texttt{Var}(\varphi)) \subseteq \mathcal{V}al$, $[\![\varphi\gamma]\!] = \top$ and $\texttt{Var}(s) \cup \texttt{Var}(t) \subseteq dom(\gamma)$. Consider $\gamma(x_i)$. Because $x_i \in dom(\gamma)$ and γ is a gsc substitution, $\gamma(s_i)$ must be a gsc term; having base type, it must have a form $\mathtt{c}\ u_1 \cdots u_k$ with $\mathtt{c}$ a constructor. Since there are no non-value constructors with a theory sort as output sort, we can only have $k = 0$ and $\mathtt{c}$ 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 $\delta := \gamma$. □

A.2 Proof strategy

Next, we consider the proof outline of Theorem 1. In the remainder of Appendix A, 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 $\mathcal{L}$.

First, for a set of equations $\mathcal{E}$, we define the symmetric relation $\leftrightarrow_{\mathcal{E}}$ as follows: for any context C over $\mathcal{L}$ we define

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

Reasoning over inductive theorems is really reasoning about the inclusion of $\leftrightarrow_{\mathcal{E}}$ in $\leftrightarrow_{\mathcal{R}}^*$:

Lemma 6. *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 our assumption to conclude $s\gamma \leftrightarrow_{\mathcal{R}}^* t\gamma$. □

We also let $\mathrel{\leftarrow\!\!\!\parallel\!\!\!\rightarrow}_{\mathcal{E}}$ denote the parallel application of zero or more $\leftrightarrow_{\mathcal{E}}$ steps, and additionally define:

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

Towards a proof of Theorem 1, we now claim that the following lemma holds:

Lemma 7 (Main lemma). *Suppose that $(\mathcal{E}, \mathcal{H}) \vdash (\mathcal{E}', \mathcal{H}')$. Then*

(1) $\leftrightarrow_{\mathcal{E}\setminus\mathcal{E}'}^{root,semi} \subseteq \rightarrow_{\mathcal{R}\cup\mathcal{H}'}^* \cdot \mathrel{\leftarrow\!\!\!\parallel\!\!\!\rightarrow}_{\mathcal{E}'} \cdot \leftarrow_{\mathcal{R}\cup\mathcal{H}'}^*$

(2) $\rightarrow_{\mathcal{R}\cup\mathcal{H}} \subseteq \rightarrow_{\mathcal{R}} \cdot \rightarrow_{\mathcal{R}\cup\mathcal{H}}^* \cdot \mathrel{\leftarrow\!\!\!\parallel\!\!\!\rightarrow}_{\mathcal{E}} \cdot \leftarrow_{\mathcal{R}\cup\mathcal{H}}^*$ *on ground terms over $\mathcal{L}$, implies* $\rightarrow_{\mathcal{R}\cup\mathcal{H}'} \subseteq \rightarrow_{\mathcal{R}} \cdot \rightarrow_{\mathcal{R}\cup\mathcal{H}'}^* \cdot \mathrel{\leftarrow\!\!\!\parallel\!\!\!\rightarrow}_{\mathcal{E}'} \cdot \leftarrow_{\mathcal{R}\cup\mathcal{H}'}^*$ *on ground terms over $\mathcal{L}$.*

This lemma will be proved in Appendix A.3. For now, we will explicitly assume that it holds. We first note:

Lemma 8. *Suppose $\leftrightarrow_{\mathcal{E}\setminus\mathcal{E}'}^{root,semi} \subseteq \rightarrow_{\mathcal{R}\cup\mathcal{H}'}^* \cdot \mathrel{\leftarrow\!\!\!\parallel\!\!\!\rightarrow}_{\mathcal{E}'} \cdot \leftarrow_{\mathcal{R}\cup\mathcal{H}'}^*$. Then $\mathrel{\leftarrow\!\!\!\parallel\!\!\!\rightarrow}_{\mathcal{E}} \subseteq \rightarrow_{\mathcal{R}\cup\mathcal{H}'}^* \cdot \mathrel{\leftarrow\!\!\!\parallel\!\!\!\rightarrow}_{\mathcal{E}'} \cdot \leftarrow_{\mathcal{R}\cup\mathcal{H}'}^*$ on ground terms over $\mathcal{L}$.*

Proof. Suppose $\leftrightarrow_{\mathcal{E}\setminus\mathcal{E}'}^{root,semi} \subseteq \rightarrow_{\mathcal{R}\cup\mathcal{H}'}^* \cdot \mathrel{\leftarrow\!\!\!\parallel\!\!\!\rightarrow}_{\mathcal{E}'} \cdot \leftarrow_{\mathcal{R}\cup\mathcal{H}'}^*$, and let s, t be ground terms such that $s \mathrel{\leftarrow\!\!\!\parallel\!\!\!\rightarrow}_{\mathcal{E}} t$. 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 $1 \le i \le n$, $s_i \approx t_i\ [\varphi_i] \in \mathcal{E}$ and γ_i respects φ_i. Consider some i.

If $s_i \approx t_i\ [\varphi_i] \in \mathcal{E}'$, then clearly $s_i\gamma \leftrightarrow_{\mathcal{E}'}^* t_i\gamma$, so also $s_i\gamma \rightarrow_{\mathcal{R}\cup\mathcal{H}'}^* \cdot \mathrel{\leftarrow\!\!\!\parallel\!\!\!\rightarrow}_{\mathcal{E}'} \cdot \leftarrow_{\mathcal{R}\cup\mathcal{H}'}^*$. 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 $\gamma_i^{\downarrow} = [x := \gamma_i(x){\downarrow_{\mathcal{R}}} \mid x \in dom(\gamma_i)]$; since $\mathcal{L}$ is quasi-reductive, $\gamma_i^{\downarrow}$ is a gsc substitution. Hence, by the assumption on $\leftrightarrow_{\mathcal{E}\setminus\mathcal{E}'}^{root,semi}$, we have: $s_i\gamma_i \rightarrow_{\mathcal{R}}^* s_i\gamma_i^{\downarrow} \rightarrow_{\mathcal{R}\cup\mathcal{H}'}^* \cdot \mathrel{\leftarrow\!\!\!\parallel\!\!\!\rightarrow}_{\mathcal{E}'} \cdot \leftarrow_{\mathcal{R}\cup\mathcal{H}'}^* t_i\gamma_i^{\downarrow} \leftarrow_{\mathcal{R}}^* t_i\gamma$.

We complete by sequentializing all $\rightarrow_{\mathcal{R}\cup\mathcal{H}}$ reductions in s, and doing all $\mathrel{\leftarrow\!\!\!\parallel\!\!\!\rightarrow}_{\mathcal{E}'}$ steps in parallel. □

We use this to see:

Lemma 9. *Suppose that $(\mathcal{E}_0, \emptyset) \vdash^* (\mathcal{E}_n, \mathcal{H}_n)$, and assume that Lemma 7 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) $\mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}_0} \subseteq \rightarrow^*_{\mathcal{R}\cup\mathcal{H}_n} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}_n} \cdot \leftarrow^*_{\mathcal{R}\cup\mathcal{H}_n}$
(2) $\rightarrow_{\mathcal{R}\cup\mathcal{H}_n} \subseteq \rightarrow_{\mathcal{R}} \cdot \rightarrow^*_{\mathcal{R}\cup\mathcal{H}_n} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}_n} \cdot \leftarrow^*_{\mathcal{R}\cup\mathcal{H}_n}$ *on ground terms over* $\mathcal{L}$*.*

Proof. We prove both statements by induction on n, using that for all $i \in \{0, \dots, n-1\}$: $\mathcal{H}_i \subseteq \mathcal{H}_{i+1}$. (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 $\mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}_0}$ is allowed to be empty).

Now, assume $n > 0$ and the lemma holds for $n-1$. So we have $\mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}_0} \subseteq \rightarrow^*_{\mathcal{R}\cup\mathcal{H}_{n-1}} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}_{n-1}} \cdot \leftarrow^*_{\mathcal{R}\cup\mathcal{H}_{n-1}}$. By the combination of Lemma 7 and Lemma 8: $\mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}_{n-1}} \subseteq \rightarrow^*_{\mathcal{R}\cup\mathcal{H}_n} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}_n} \cdot \leftarrow^*_{\mathcal{R}\cup\mathcal{H}_n}$. Therefore $\mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}_0} \subseteq \rightarrow^*_{\mathcal{R}\cup\mathcal{H}_{n-1}} \cdot (\rightarrow^*_{\mathcal{R}\cup\mathcal{H}_n} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}_n} \cdot \leftarrow^*_{\mathcal{R}\cup\mathcal{H}_n}) \cdot \leftarrow^*_{\mathcal{R}\cup\mathcal{H}_{n-1}}$. We complete (1) because $\rightarrow^*_{\mathcal{R}\cup\mathcal{H}_{n-1}} \cdot \rightarrow^*_{\mathcal{R}\cup\mathcal{H}_n}$ is included in $\rightarrow^*_{\mathcal{R}\cup\mathcal{H}_n}$. As for (2): this follows immediately by induction hypothesis (2) and Lemma 7.(2). ◻

We will also use 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 10 ([14, Lemma 3.4]). *Let* $\rightarrow_1 \subseteq \rightarrow_2$ *be binary relations with* $\rightarrow_2$ *well-founded and* $\rightarrow_2 \subseteq \rightarrow_1 \cdot \rightarrow_2^* \cdot \leftrightarrow_1^* \cdot \leftarrow_2^*$*. Then* $\leftrightarrow_1^* = \leftrightarrow_2^*$*.*

Proof. The direction $\leftrightarrow_1^* \subseteq \leftrightarrow_2^*$ is implied by $\rightarrow_1 \subseteq \rightarrow_2$. To prove $\leftrightarrow_2^* \subseteq \leftrightarrow_1^*$ it suffices to prove $\rightarrow_2 \subseteq \leftrightarrow_1^*$. So assume $s \rightarrow_2 t$. We use well-founded induction on $s \rightarrow_2$ to show $s \leftrightarrow_1^* t$. First, we use our assumption: there exist terms $a_1, \dots, a_n$ and $b_1, \dots b_m$ such that

$$s \rightarrow_1 a_1 \rightarrow_2 a_2 \rightarrow_2 \dots \rightarrow_2 a_n \leftrightarrow_1^* b_m \leftarrow_2 \dots \leftarrow_2 b_2 \leftarrow_2 b_1 = t$$

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 by induction hypothesis we can conclude $a_i \leftrightarrow_1^* a_{i+1}$ for all $1 \leq i < n$. By assumption we have $s \rightarrow_2 t$, hence (again by induction hypothesis) $b_i \leftrightarrow_1^* b_{i+1}$ for all $1 \leq i < m$. This gives us the desired sequence $s \rightarrow_1 a_1 \leftrightarrow_1^* a_2 \leftrightarrow_1^* \dots \leftrightarrow_1^* a_n \leftrightarrow_1^* b_m \leftrightarrow_1^* \dots \leftrightarrow_1^* b_2 \leftrightarrow_1^* b_1 = t$. ◻

With these preparations, we can now prove Theorem 1 (conditional on Lemma 7):

Theorem 1 – conditionally. *Let* $\mathcal{E}$ *be a set of equations. Assume that, by higher-order rewriting induction,* $(\mathcal{E}, \emptyset) \vdash^* (\emptyset, \mathcal{H})$*, for some set* $\mathcal{H}$*, and Lemma 7 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 9 with $\mathcal{E}_0 = \mathcal{E}$, $\mathcal{E}_n = \emptyset$ and $\mathcal{H}_n = \mathcal{H}$ to obtain

1. $\mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}} \subseteq \rightarrow^*_{\mathcal{R}\cup\mathcal{H}} \cdot \leftarrow^*_{\mathcal{R}\cup\mathcal{H}}$ 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 10 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:

$$\leftrightarrow_{\mathcal{E}} \subseteq \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}} \subseteq \rightarrow^*_{\mathcal{R}\cup\mathcal{H}} \cdot \leftarrow^*_{\mathcal{R}\cup\mathcal{H}} \subseteq \leftrightarrow^*_{\mathcal{R}\cup\mathcal{H}} \cdot \leftrightarrow^*_{\mathcal{R}\cup\mathcal{H}} = \leftrightarrow^*_{\mathcal{R}\cup\mathcal{H}} = \leftrightarrow^*_{\mathcal{R}}$$

on ground terms over $\mathcal{L}$. Finally, use Lemma 6 to conclude Theorem 1. ◻

A.3 Soundness proof

It remains to be seen that Lemma 7 holds for every single-step deduction. 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

$$\leftrightarrow^{root,semi}_{\mathcal{E}\setminus\mathcal{E}'} \subseteq \rightarrow^*_{\mathcal{R}\cup\mathcal{H}} \cdot \mathrel{\leftarrow\!\!\mid\!\mid\!\!\rightarrow}_{\mathcal{E}'} \cdot \leftarrow^*_{\mathcal{R}\cup\mathcal{H}}$$

We will start with these cases.

Simplification. To prove correctness of the Simplification rule, we introduce the following helper lemma.

Lemma 11. *Let $\mathcal{L} = (\Sigma, \mathcal{V}, \mathcal{R})$ 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 $\ell \to r\ [\varphi] \in \mathcal{X} \cup \mathcal{R}_{calc}$, context C over $\mathcal{L}$ and substitution δ over $\mathcal{L}$ such that $\delta(LVar(\ell \to r\ [\varphi])) \subseteq \mathcal{V}al \cup \mathtt{Var}(\psi)$ and $\psi \Longrightarrow \varphi\delta$ valid. Then for any substitution γ over $\mathcal{L}$ which respects ψ we have $C[\ell\delta]\gamma \to_{\mathcal{X}} C[r\delta]\gamma$.*

Proof. Let γ be a substitution over $\mathcal{L}$ respecting ψ and define $\eta := \gamma \circ \delta$. We check that $C[\ell\delta]\gamma \to_{\mathcal{X}} C[r\delta]\gamma$ with substitution η and rule $\ell \to r\ [\varphi]$.
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 $\ell \to r\ [\varphi]$, i.e. $\eta(LVar(\ell \to r\ [\varphi])) \subseteq \mathcal{V}al$ and $[\![\varphi\eta]\!] = \top$. Now, since γ respects ψ we have $\gamma(\mathtt{Var}(\psi)) \subseteq \mathcal{V}al$ and $[\![\psi\gamma]\!] = \top$. Therefore $\eta(LVar(\ell \to r\ [\varphi])) = \gamma(\delta(LVar(\ell \to r\ [\varphi]))) \subseteq \gamma(\mathcal{V}al \cup \mathtt{Var}(\psi)) = \gamma(\mathcal{V}al) \cup \gamma(\mathtt{Var}(\psi)) = \mathcal{V}al \cup \gamma(\mathtt{Var}(\psi)) = \mathcal{V}al$ and, because of $\psi \Longrightarrow \varphi\delta$, we have that $\top = [\![(\varphi\delta)\gamma]\!] = [\![\varphi\eta]\!]$. □

Lemma 12. *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 **(Simplification)**, using the rule $\ell \to r\ [\varphi]$. Then $\leftrightarrow^{root,semi}_{\{C[\ell\delta] \approx t\ [\psi]\}} \subseteq \rightarrow^*_{\mathcal{R}\cup\mathcal{H}} \cdot \mathrel{\leftarrow\!\!\mid\!\mid\!\!\rightarrow}_{\mathcal{E}\cup\{C[r\delta]\approx t\ [\psi]\}} \cdot \leftarrow^*_{\mathcal{R}\cup\mathcal{H}}$.*

Proof. Suppose $C[\ell\delta]\gamma \leftrightarrow^{root,semi}_{\{C[\ell\delta]\approx t\ [\psi]\}} t\gamma$, with γ a gsc substitution over $\mathcal{L}$ that respects ψ. Then, by definition of Simplification, we have $\delta(LVar(\ell \to r\ [\varphi])) \subseteq \mathcal{V}al \cup \mathtt{Var}(\psi)$ and $\psi \Longrightarrow \varphi\delta$. Now, Lemma 11 (take $\mathcal{X} = \mathcal{R} \cup \mathcal{H}$) guarantees $C[\ell\delta]\gamma \to_{\mathcal{R}\cup\mathcal{H}} C[r\delta]\gamma$. Therefore, $C[\ell\delta]\gamma \to_{\mathcal{R}\cup\mathcal{H}} C[r\delta]\gamma \mathrel{\leftarrow\!\!\mid\!\mid\!\!\rightarrow}_{\{C[r\delta]\approx t\ [\psi]\}} t\gamma$. □

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

Lemma 13. *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 $\leftrightarrow^{root,semi}_{\{s\approx t\ [\varphi]\}} \subseteq \rightarrow^*_{\mathcal{R}\cup\mathcal{H}} \cdot \mathrel{\leftarrow\!\!\mid\!\mid\!\!\rightarrow}_{\mathcal{E}\cup\{u\approx v\ [\psi]\}} \cdot \leftarrow^*_{\mathcal{R}\cup\mathcal{H}}$.*

Proof. Suppose $s\gamma \leftrightarrow^{root,semi}_{\{s\approx t\ [\varphi]\}} t\gamma$, 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, $s\gamma \leftrightarrow_{\{u\approx v\ [\psi]\}} t\gamma$. □

Deletion.

Lemma 14. *If* $(\mathcal{E} \uplus \{s \approx t\ [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E}, \mathcal{H})$ *by* ***(Deletion)*** *then* $\leftrightarrow^{root,semi}_{\{s \approx t\ [\varphi]\}} \subseteq \rightarrow^*_{\mathcal{R}\cup\mathcal{H}} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}} \cdot \leftarrow^*_{\mathcal{R}\cup\mathcal{H}}$.

Proof. Assume $s\gamma \leftrightarrow^{root,semi}_{\{s\approx t\ [\varphi]\}} t\gamma$, 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. □

Semi-constructor.

Lemma 15. *If* $\left(\mathcal{E} \uplus \{\mathsf{c}\ \vec{s} \approx \mathsf{c}\ \vec{t}\ [\varphi]\}, \mathcal{H}\right) \vdash \left(\mathcal{E} \cup \{s_i \approx t_i\ [\varphi] \mid 1 \le i \le n\}, \mathcal{H}\right)$ *by* ***(Semi-constructor)*** *then* $\leftrightarrow^{root,semi}_{\{\mathsf{c}\ \vec{s}\ \approx\ \mathsf{c}\ \vec{t}\ [\varphi]\}} \subseteq \rightarrow^*_{\mathcal{R}\cup\mathcal{H}} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}\cup\{s_i\approx t_i\ [\varphi] \mid 1\le i\le n\}} \cdot \leftarrow^*_{\mathcal{R}\cup\mathcal{H}}$.

Proof. Suppose that $(\mathsf{c}\ \vec{s})\gamma \leftrightarrow^{root,semi}_{\{\mathsf{c}\ \vec{s}\ \approx\ \mathsf{c}\ \vec{t}\ [\varphi]\}} (\mathsf{c}\ \vec{t})\gamma$, with γ a gsc substitution over $\mathcal{L}$ that respects φ. By definition we have $s_i\gamma \leftrightarrow_{\mathcal{E}\cup\{s_i\approx t_i\ [\varphi] \mid 1\le i\le n\}} t_i\gamma$. As all s_i occur in parallel, it follows that $(\mathsf{c}\ \vec{s})\gamma \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}\cup\{s_i\approx t_i\ [\varphi] \mid 1\le i\le n\}} (\mathsf{c}\ \vec{t})\gamma$. □

Expansion. Since Expansion adds a rewrite rule to $\mathcal{H}$, here part (2) of Lemma 7 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 16. *Let* $\mathsf{f}\ s_1 \cdots s_n$ *be a ground term over* $\mathcal{L}$ *such that* $\mathsf{f} \in \mathcal{D}$, $n \ge ar(\mathsf{f}) = k$ *and every* $s_i \in \mathcal{SCT}_{\mathcal{L}}$. *Then there is a rule* $\ell \to r\ [\psi] \in \mathcal{R}$ *and a substitution* δ *over* $\mathcal{L}$ *respecting this rule such that* $\mathsf{f}\ s_1 \cdots s_k = l\delta$.

Proof. Quasi-reductivity of $\mathcal{L}$ implies that $\mathsf{f}\ s_1 \cdots s_n$ reduces ($n \ge ar(\mathsf{f})$ 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 $\ell \to r\ [\psi] \in \mathcal{R}$ and a substitution δ over $\mathcal{L}$ respecting this rule such that $\mathsf{f}\ s_1 \cdots s_k = \ell\delta$. □

Lemma 17. *Let* $s \approx t\ [\varphi]$ *be an equation such that* $s|_p = \mathsf{f}\ s_1 \cdots s_n$ *with* $\mathsf{f} \in \mathcal{D}$, $n \ge ar(\mathsf{f})$ *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*

$$s\gamma \rightarrow_{\mathcal{R}} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}\cup Expd(s\approx t\ [\varphi],p)} t\gamma$$

Proof. Let γ be a gsc substitution over $\mathcal{L}$ which respects φ. By Lemma 16, $s|_p\gamma$ reduces at root position with some rule $\ell \to r\ [\psi] \in \mathcal{R}$ and substitution δ over $\mathcal{L}$ respecting ψ. We can assume the variables in $\ell \to r\ [\psi]$ 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 $s[\ell\ s_{k+1} \cdots s_n]_p\delta' \rightarrow_{\mathcal{R}} s[r\ s_{k+1} \cdots s_n]_p\delta'$.

Now, since $\mathsf{f}\ s_1 \cdots s_k$ and ℓ are unifiable with unifier δ', there is also a most general unifier η; "most general" implies that $\delta' = \eta\zeta$ for some substitution ζ. Hence, $Expd(s \approx t\ [\varphi], p)$ includes an equation $s[r\ s_{k+1} \cdots s_n]_p\eta \approx t\eta$. Conclude $s\gamma = s[l\ s_{k+1} \cdots s_n]_p\delta' \to_{\mathcal{R}} s[r\ s_{k+1} \cdots s_n]_p\delta' = s[r\ s_{k+1} \cdots s_n]_p\eta\zeta \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{Expd(s \approx t\ [\varphi], p)} t\eta\zeta = t\gamma$. ◻

Now part (1) of Lemma 7 is proved by the following:

Lemma 18.
If $(\mathcal{E} \uplus \{s \approx t\ [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E} \uplus Expd(s \approx t\ [\varphi], p), \mathcal{H} \cup \{s \to t\ [\varphi]\})$ *by* ***(Expansion)*** *then* $\leftrightarrow^{root,semi}_{\{s \approx t\ [\varphi]\}} \subseteq \to^*_{\mathcal{R} \cup \mathcal{H} \cup \{s \to t\ [\varphi]\}} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E} \uplus Expd(s \approx t\ [\varphi], p)} \cdot \leftarrow^*_{\mathcal{R} \cup \mathcal{H} \cup \{s \to t\ [\varphi]\}}$.

Proof. Suppose $s\gamma \leftrightarrow^{root,semi}_{\{s \approx t\ [\varphi]\}} t\gamma$, with γ a gsc substitution over $\mathcal{L}$ that respects φ. By Lemma 17 we have $s\gamma \to_{\mathcal{R}} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{Exp(s \approx t\ [\varphi], p)} t\gamma$. ◻

And part (2) of Lemma 7 is proved by the following:

Lemma 19.
Suppose $(\mathcal{E} \uplus \{s \approx t\ [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E} \cup Expd(s \approx t\ [\varphi], p), \mathcal{H} \cup \{s \to t\ [\varphi]\})$ *by* ***(Expansion)****, and* $\to_{\mathcal{R} \cup \mathcal{H}} \subseteq \to_{\mathcal{R}} \cdot \to^*_{\mathcal{R} \cup \mathcal{H}} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E} \cup \{s \approx t\ [\varphi]\}} \cdot \leftarrow^*_{\mathcal{R} \cup \mathcal{H}}$ *on ground terms over* $\mathcal{L}$*. Then:*

$$\begin{aligned}\to_{\mathcal{R} \cup \mathcal{H} \cup \{s \to t\ [\varphi]\}} \subseteq\ & \to_{\mathcal{R}} \cdot \to^*_{\mathcal{R} \cup \mathcal{H} \cup \{s \to t\ [\varphi]\}} \cdot \\ & \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E} \cup Expd(s \approx t\ [\varphi], p)} \cdot \leftarrow^*_{\mathcal{R} \cup \mathcal{H} \cup \{s \to t\ [\varphi]\}}\end{aligned}$$

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

Proof. Suppose $\to_{\mathcal{R} \cup \mathcal{H}} \subseteq \to_{\mathcal{R}} \cdot \to^*_{\mathcal{R} \cup \mathcal{H}} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E} \cup \{s \approx t\ [\varphi]\}} \cdot \leftarrow^*_{\mathcal{R} \cup \mathcal{H}}$ on ground terms over $\mathcal{L}$. Then by Lemma 18 and Lemma 8:

$$\begin{aligned}\to_{\mathcal{R} \cup \mathcal{H}} \subseteq \to_{\mathcal{R}} \cdot \to^*_{\mathcal{R} \cup \mathcal{H}} \cdot \big(& \to^*_{\mathcal{R} \cup \mathcal{H} \cup \{s \to t\ [\varphi]\}} \cdot \\ & \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E} \cup Expd(s \approx t\ [\varphi], p)} \cdot \leftarrow^*_{\mathcal{R} \cup \mathcal{H} \cup \{s \to t\ [\varphi]\}} \big) \cdot \leftarrow^*_{\mathcal{R} \cup \mathcal{H}}\end{aligned}$$

Or equivalently

$$\to_{\mathcal{R} \cup \mathcal{H}} \subseteq \to_{\mathcal{R}} \cdot \to^*_{\mathcal{R} \cup \mathcal{H} \cup \{s \to t\ [\varphi]\}} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E} \cup Expd(s \approx t\ [\varphi], p)} \cdot \leftarrow^*_{\mathcal{R} \cup \mathcal{H} \cup \{s \to t\ [\varphi]\}}$$

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

$$\to_{\{s \to t\ [\varphi]\}} \subseteq \to_{\mathcal{R}} \cdot \to^*_{\mathcal{R} \cup \mathcal{H} \cup \{s \to t\ [\varphi]\}} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E} \cup Expd(s \approx t\ [\varphi], p)} \cdot \leftarrow^*_{\mathcal{R} \cup \mathcal{H} \cup \{s \to t\ [\varphi]\}}$$

So suppose $C[s\gamma], C[t\gamma]$ are ground terms over $\mathcal{L}$ such that $C[s\gamma] \to_{\{s \to t\ [\varphi]\}} C[t\gamma]$, for some ground substitution γ over $\mathcal{L}$ that respects $\ell \to r\ [\varphi]$. Then $\gamma^{\downarrow}$ with $\gamma^{\downarrow}(x) = \gamma(x){\downarrow_{\mathcal{R}}}$ for all $x \in dom(\gamma)$ is a gsc substitution over $\mathcal{L}$, so by Lemma 17 we have $s\delta \to_{\mathcal{R}} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{Exp(s \approx t\ [\varphi], p)} t\delta$. Therefore

$$C[s\gamma] \to^*_{\mathcal{R}} C[s\delta] \to_{\mathcal{R}} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{Exp(s \approx t\ [\varphi], p)} C[t\delta] \leftarrow^*_{\mathcal{R}} C[t\gamma]$$

◻

B Proofs for Section 5

Although there are no lemmas or theorems in Section 5, we did claim in the text that strong quasi-reductivity implies general quasi-reductivity. In this appendix, we prove that statement.

We first introduce a helper function μ, that replaces every higher-order subterm in a ground semi-constructor term by a variable.

Definition 9 (μ). *For a ground semi-constructor term s, choose variables F_p for every $p \in Pos(s)$ 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 $\mathsf{c}\ t_1 \cdots t_m$ with c a constructor, so we let $\mu_p(t) := \mathsf{c}\ \mu_{p\cdot 1}(t_1) \cdots \mu_{p\cdot m}(t_m)$*

Then clearly $\mu(s)$ 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: $\mu(s)[F_p := s|_p \mid p \in Pos(s) \wedge s|_p \text{ has a higher type}] = s$. With this, we easily prove our desired result.

Lemma 20. *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 $\mathsf{f}\ s_1 \cdots s_n$ for some $n \geq ar(\mathsf{f})$, where $\mathsf{f} \in \mathcal{D}$ and all s_i are semi-constructor terms, but $\mathsf{f}\ s_1 \cdots s_n$ does not reduce. However, by strong quasi-reductivity, $\mathsf{f}\ \mu_1(s_1) \cdots \mu_n(s_n)$ does reduce! But then $(\mathsf{f}\ \mu_1(s_1) \cdots \mu_n(s_n))[F_p := s|_p \mid p \in Pos(s) \wedge s|_p \text{ has higher type}] = \mathsf{f}\ s_1 \cdots s_n$ must also reduce. This gives the required contradiction. ◻

C Proofs for Section 6

C.1 Oracle extensions

Towards a proof of lemma 2, we show how an Oracle extension can be constructed from an arbitrary terminating quasi-reductive natural extension.

Construction 1 Let $\mathcal{L}$ be an LCSTRS and $\mathcal{L}'$ be a natural extension of $\mathcal{L}$. Let

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

Moreover, let:

- $\mathcal{S}^{\mathcal{Q}} = \mathcal{S}'$ and $\mathcal{V}^{\mathcal{Q}} = \mathcal{V}'$
- $\mathcal{S}^{\mathcal{Q}}_{theory} = \mathcal{S}_{theory}$ and $\Sigma^{\mathcal{Q}}_{theory} = \Sigma_{theory}$ and $\mathcal{I}^{\mathcal{Q}} = \mathcal{I}$ and $[\![\cdot]\!]^{\mathcal{Q}} = [\![\cdot]\!]$ (so we use the theory of the *original* signature, not the extension)
- $\Sigma^{\mathcal{Q}}_{terms} = \Sigma_{terms} \cup \{\mathsf{f}' \mid \mathsf{f} \in \Sigma'_{terms} \setminus \mathcal{C}\}$ (so there is a symbol f' for each $\mathsf{f} \in \Sigma'_{terms}$ that is not a constructor in the original LCSTRS $\mathcal{L}$)
- $\mathit{typeof}^{\mathcal{Q}}(x) = \mathit{typeof}'(x)$ for $x \in \mathcal{V}'$;
  $\mathit{typeof}^{\mathcal{Q}}(\mathsf{f}) = \mathit{typeof}'(\mathsf{f}) = \mathit{typeof}(\mathsf{f})$ for $\mathsf{f} \in \Sigma_{terms}$; and
  $\mathit{typeof}^{\mathcal{Q}}(\mathsf{f}') = \mathit{typeof}'(\mathsf{f})$ for $\mathsf{f} \in \Sigma'_{terms} \setminus \mathcal{C}$

For a term $s \in T(\Sigma^{\mathcal{Q}}, \mathcal{V}^{\mathcal{Q}})$, we let $\zeta(s) \in T(\Sigma', \mathcal{V}')$ be the term that is obtained by replacing each f' by the corresponding f. For $t \in T(\Sigma', \mathcal{V}')$ or in $T(\Sigma^{\mathcal{Q}}, \mathcal{V}^{\mathcal{Q}})$, we let $\chi(t) \in T(\Sigma^{\mathcal{Q}}, \mathcal{V}^{\mathcal{Q}})$ be the term obtained by replacing each f that occurs in $\Sigma'_{terms} \setminus \mathcal{C}$ by f' (note that all elements of $\mathcal{C}$ are in $\Sigma^{\mathcal{Q}}$, 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 $s \in T(\Sigma^{\mathcal{Q}}, \mathcal{V}^{\mathcal{Q}})$ We define:

$$\mathcal{R}_{\mathsf{orac}} = \left\{ \mathsf{f}'\ s_1 \cdots s_m \to \chi(w) \quad \text{where} \quad \begin{array}{l} s_1, \dots, s_m, w \in T(\Sigma^{\mathcal{Q}}, \emptyset) \\ (\mathsf{f}\ \zeta(s_1) \cdots \zeta(s_m), w) \in \mathcal{P} \end{array} \right\}$$

The *oracle variant of* $\mathcal{L}'$ is the LCSTRS $\mathcal{Q}_{\mathsf{orac}}$ generated from $\mathcal{S}^{\mathcal{Q}}, \mathcal{V}^{\mathcal{Q}}, \mathcal{S}^{\mathcal{Q}}_{theory}, \Sigma^{\mathcal{Q}}_{theory}, \mathcal{I}^{\mathcal{Q}}, [\![\cdot]\!]^{\mathcal{Q}}, \Sigma^{\mathcal{Q}}_{terms}, \mathit{typeof}^{\mathcal{Q}}$ and rules $\mathcal{R} \cup \mathcal{R}_{\mathsf{orac}}$.

We can now make several observations:

Lemma 21. *Suppose $\mathcal{L}'$ is terminating and quasi-reductive. Then the oracle variant of $\mathcal{L}'$ is also a natural extension of $\mathcal{L}$.*

Proof. By definition, all inclusions are satisfied.

The set $\mathcal{R}_{\mathsf{orac}}$ only defines symbols f', which do not occur in $\mathcal{L}$.

For all symbols $\mathsf{g} :: \sigma_1 \to \ldots \to \sigma_m \to \iota$, $\zeta(\mathsf{g})$ is a function symbol in $\mathcal{L}'$, so for $1 \le i \le m$ 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 $\mathcal{Q}_{\mathsf{orac}}$.

Finally, if $\mathsf{c} :: \sigma_1 \to \ldots \to \sigma_m \to \iota$ is a constructor of $\mathcal{Q}_{\mathsf{orac}}$, there are three possibilities:

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

Lemma 22. *If $s \to_{\mathcal{R} \cup \mathcal{R}_{\mathsf{orac}}} t$ in $\mathcal{Q}_{\mathsf{orac}}$ (s, t ground), then $\zeta(s) \to^{+}_{\mathcal{R}'} \zeta(t)$ in $\mathcal{L}'$.*

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

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

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

Corollary 1. *If $\mathcal{L}'$ is terminating, so is $\mathcal{Q}_{\mathtt{orac}}$.*

Lemma 23. *If $\mathcal{L}$ is strong quasi-reductive and $\mathcal{L}'$ is both terminating and quasi-reductive, then $\mathcal{Q}_{\mathtt{orac}}$ is quasi-reductive.*

Proof. We prove by induction on the size of a ground term s in $\mathcal{Q}_{\mathtt{orac}}$: if s does not reduce in $\mathcal{Q}_{\mathtt{orac}}$, then s is a semi-constructor term. We can always denote $s = \mathsf{g}\ s_1 \cdots s_n$ with $\mathsf{g} :: \sigma_1 \to \ldots \to \sigma_m \to \iota$ ($m \geq n$, $\iota \in \mathcal{S}'$). 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 $\mathcal{R} \cup \mathcal{R}_{\mathtt{orac}}$, then we are immediately done.
- If g is defined in $\mathcal{R}_{\mathtt{orac}}$ but $n < m$ then we are done, since all left-hand sides of rules in $\mathcal{R}_{\mathtt{orac}}$ are maximally applied.
- If g is defined in $\mathcal{R}_{\mathtt{orac}}$ and $n = m$, then $\mathsf{g} = \mathsf{f}'$ for some defined or calculation symbol $\mathsf{f} \in \mathcal{L}'$. Either way, note that $\zeta(s) = \mathsf{f}\ \zeta(s_1) \cdots \zeta(s_n)$ 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 $s \to \chi(w) \in \mathcal{R}_{\mathtt{orac}}$, 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 $\mathsf{g} \in \Sigma^{\mathcal{Q}}_{theory} = \Sigma_{theory}$, so $\zeta(s) = \mathsf{g}\ \zeta(s_1) \cdots \zeta(s_n)$. 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 $\mathsf{c}\ t_1 \cdots t_l$ with c a constructor symbol in $\mathcal{Q}_{\mathtt{orac}}$ (since s_i is a ground semi-constructor term of base type), so $\zeta(s_i) = \zeta(\mathsf{c})\ \zeta(t_1) \cdots \zeta(t_l)$.
  We claim (**) that $\zeta(\mathsf{c})$ is a constructor in $\mathcal{L}'$, which means that any reduct of $\zeta(s_i)$ still has $\zeta(\mathsf{c})$ as root symbol. Since $\mathcal{R}'_{\mathsf{g}} = \mathcal{R}_{\mathsf{g}}$ (a natural extension cannot add cases to a defined symbol or calculation of $\mathcal{L}$), there are no rules in $\mathcal{R}'$ to reduce $\mathsf{g}\ (\zeta(s_1) \downarrow_{\mathcal{R}'}) \cdots (\zeta(s_m) \downarrow_{\mathcal{R}'})$ and yet it is not a semi-constructor term; which means that the calculation rule must apply, so $\zeta(\mathsf{c})$ can only be a value. But if $\zeta(\mathsf{c})$ is a value, then c is a value, so we see that all s_i are values. Hence, $s = \mathsf{g}\ s_1 \cdots s_m$ reduces (using a calculation rule).

To prove (**), suppose that $\zeta(\mathsf{c})$ 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 $\mathcal{R}_{\mathtt{orac}}$—and therefore s_i cannot be a semi-constructor term in $\mathcal{Q}_{\mathtt{orac}}$.

- Finally, if g is defined in $\mathcal{R}$, then we are done if $n < ar(\mathsf{g})$. If $n \geq ar(\mathsf{g})$, then consider $\mu\zeta(s) = \mathsf{g}\ \mu(\zeta(s_1)) \cdots \mu(\zeta(s_n))$. 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 $s_i = \mu(s_i)[F_p := s_i|_p \mid p \in Pos(s_i) \wedge s_i|_p$ has higher type$]$.

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

Lemma 24. *Suppose $\mathcal{L}'$ is terminating and quasi-reductive. Then the oracle variant of $\mathcal{L}'$ is an Oracle extension of $\mathcal{L}$.*

Proof. By Lemma 21, it is a natural extension. By Corollary 1, it is terminating. By Lemma 23, it is quasi-reductive.

Clearly all rules in $\mathcal{R}_{\mathtt{orac}}$ have a shape $\mathsf{f}'\ s_1 \cdots s_m \to \chi(w)$ with f' a defined symbol of $\mathcal{R}_{\mathtt{orac}}$ 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 $ar(\mathsf{f}) = ar^{\mathcal{Q}}(\mathsf{f})$ for $\mathsf{f} \in \Sigma_{terms}$, and $ar(\mathsf{f}) \leq ar^{\mathcal{Q}}(\mathsf{f}')$ for $\mathsf{f} \in \Sigma'_{terms} \setminus \mathcal{C}$, this means $\chi(w)$ is a semi-constructor term for $\mathcal{Q}$. □

Lemma 25. *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 $s\gamma^\chi \leftrightarrow^*_{\mathcal{R} \cup \mathcal{R}_{\mathtt{orac}}} t\gamma^\chi$, 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) $\gamma(x) \in \mathcal{C}$, so $\gamma^\chi(x) = \gamma(x)$.
- $[\![\varphi\gamma]\!] = \top$; since $\gamma^\chi(x) = \gamma(x)$ for all $x \in \mathtt{Var}(\varphi)$, clearly also $[\![\varphi\gamma^\chi]\!] = \top$.

Now suppose $s\gamma^\chi \leftrightarrow^*_{\mathcal{R} \cup \mathcal{R}_{\mathtt{orac}}} t\gamma^\chi$. We see by Lemma 22 that $\zeta(s\gamma^\chi) \leftrightarrow^*_{\mathcal{R}'} \zeta(t\gamma^\chi)$, and since $\zeta(s\gamma^\chi) = s\gamma$ and $\zeta(t\gamma^\chi) = t\gamma$, this is exactly what we need. □

Thus, we finally have the prerequisites to prove Lemma 2.

Lemma 2. *An equation $s \approx t\ [\varphi]$ over a terminating, strong quasi-reductive LCSTRS is a global inductive theorem of $\mathcal{L}$ if for every Oracle extension $\mathcal{Q}$ and ground substitution γ over $\mathcal{Q}$ that respects this equation: $s\gamma \leftrightarrow^*_{\mathcal{R}^{\mathcal{Q}}} t\gamma$.*

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 see that for every ground substitution δ over $\mathcal{L}'$ that respects the equation: $s\delta \leftrightarrow^*_{\mathcal{R}'} t\delta$.

Let $\mathcal{Q}_{\mathtt{orac}}$ be the oracle variant of $\mathcal{L}'$. By Lemma 24, $\mathcal{Q}_{\mathtt{orac}}$ is an Oracle extension of $\mathcal{L}$, and by Lemma 25, δ^χ respects the equation. Hence, by the assumptions in the lemma (choosing δ^χ for γ and $\mathcal{R} \cup \mathcal{R}_{\mathtt{orac}}$ for $\mathcal{R}^{\mathcal{Q}}$), $s\delta^\chi \leftrightarrow^*_{\mathcal{R}\cup\mathcal{R}_{\mathtt{orac}}} t\delta^\chi$. Again by Lemma 25, this implies the required property $s\delta \leftrightarrow^*_{\mathcal{R}'} t\delta$. ◻

Proving oracle termination. In the text, we also 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 [12] (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 $\sqsupset$ is well-founded, and define the following relation between sorts and arbitrary types:

$$\iota \sqsupseteq^+ \sigma_1 \to \ldots \to \sigma_m \to \kappa \text{ if } \iota \sqsupseteq \kappa \wedge \forall i \in \{1,\ldots,m\}.\iota \sqsupset^- \sigma_i$$
$$\iota \sqsupset^- \sigma_1 \to \ldots \to \sigma_m \to \kappa \text{ if } \iota \sqsupset \kappa \wedge \forall i \in \{1,\ldots,m\}.\iota \sqsupseteq^+ \sigma_i$$

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

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

Assume given a natural extension $\mathcal{L}'$ of $\mathcal{L}$. Given a sort ordering and accessibility function $\mathtt{Acc}$, 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 then proves termination of the extension (for more details on the method and the assumptions on the extension, we refer the reader to [12]). Thus, to use this method, we only have to prove computability of all oracle symbols.

Now, we assume given an Oracle extension $\mathcal{Q} = (\Sigma^{\mathcal{Q}}, \mathcal{R}^{\mathcal{Q}})$ 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 $\mathtt{Acc}$ that for any constructor $\mathsf{c} :: \sigma_1 \to \ldots \to \sigma_m \to \iota$ of the original signature, if $i \in \mathtt{Acc}(\mathsf{c})$ and σ_i is an arrow type $\tau_1 \to \ldots \to \tau_n \to \kappa$, then $\iota \sqsupset \kappa$. This restriction still allows us to for instance use lists of functions. We then assign $\mathtt{Acc}(\mathsf{f}) = \emptyset$ for all $\mathsf{f} \in \Sigma^{\mathcal{Q}} \setminus \Sigma$, and extend the computability notion to terms over $\mathcal{Q}$. This gives us the following property:

Lemma 26. *Fix a sort* ι*. Suppose that for all* $\mathsf{f} :: \sigma_1 \to \ldots \to \sigma_m \to \kappa$ *in* $\Sigma^{\mathcal{Q}} \setminus \Sigma$ *with* $\iota \sqsupset \kappa$*, and all terminating ground terms* $s_1 :: \sigma_1, \ldots, s_m :: \sigma_m$*, the term* $\mathsf{f}\ s_1 \cdots s_m$ *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 $s = \mathsf{c}\ s_1 \cdots s_m$ with $\mathsf{c} :: \sigma_1 \to \ldots \to \sigma_m \to \iota'$ a constructor. Since a gsc term does not reduce, we only need to show that s_i is computable for $i \in \mathtt{Acc}(\mathsf{c})$. If c is in $\Sigma^{\mathcal{Q}} \setminus \Sigma$, this is immediate because then $\mathtt{Acc}(\mathsf{c}) = \emptyset$; so assume that $\mathsf{c} \in \Sigma$. Let $i \in \mathtt{Acc}(\mathsf{c})$, so $\kappa \sqsupseteq^+ \sigma_i$ by definition of $\mathtt{Acc}$. 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 $s_i = \mathsf{g}\ u_1 \cdots u_n$ with $\mathsf{g} :: \tau_1 \to \ldots \to \tau_p \to \kappa$, with $\sigma_i = \tau_{n+1} \to \ldots \to \tau_p \to \kappa$. 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 $u_{n+1} :: \tau_{n+1}, \ldots, u_p :: \tau_p$ are terminating by the computability notion. Hence, by the induction hypothesis, $\mathsf{g}\ u_1 \cdots u_p$ is computable for all computable $u_{n+1}, \ldots, u_p$, which implies computability of $\mathsf{g}\ u_1 \cdots u_p = s_i$ as desired. ☐

This allows us to prove computability of all our extra function symbols:

Lemma 27. *If* $\mathsf{f} :: \sigma_1 \to \ldots \to \sigma_m \to \iota \in \Sigma^{\mathcal{Q}} \setminus \Sigma$*, and* $s_1 :: \sigma_1, \ldots, s_m :: \sigma_m$ *are computable terms, then* $\mathsf{f}\ s_1 \cdots s_m$ *is computable.*

Proof. We assume given *terminating* terms $s_1, \ldots, s_m$ under $\to_{\mathcal{R}^{\mathcal{Q}}}$ (by definition of computability, computable terms satisfy this property), and prove that $\mathsf{f}\ s_1 \cdots s_m$ is computable by induction first on ι (ordered with $\sqsupset$), second by $(s_1, \ldots, s_m)$ (ordered placewise with $\to_{\mathcal{R}^{\mathcal{Q}}}$).

Since $\mathtt{Acc}(\mathsf{f}) = \emptyset$ and s has base type, we only need to prove that t is computable whenever $s \to_{\mathcal{R}^{\mathcal{Q}}} t$. 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; we complete by Lemma 26 (since the induction hypothesis gives the prerequisites to apply the lemma). ☐

C.2 Proof strategy

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

Theorem 2. *Let* $\mathcal{L}$ *be a terminating, strong quasi-reductive LCSTRS and let* $\mathcal{E}$ *be a set of equations. If, by global rewriting induction,* $(\mathcal{E}, \emptyset) \vdash^* (\emptyset, \mathcal{H})$*, for some set* $\mathcal{H}$*, then every equation in* $\mathcal{E}$ *is a global inductive theorem of* $\mathcal{L}$*.*

By Lemma 2, 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 $\mathcal{Q} = (\Sigma^{\mathcal{Q}}, \mathcal{R}^{\mathcal{Q}})$ of $\mathcal{L}$. The proof is very similar to the proof of Theorem 1 in Appendix A:

- We define $\leftrightarrow_{\mathcal{E}}$ as in Appendix A, but with the difference that C is now allowed to be a context over $\mathcal{Q}$, and γ a substitution over $\mathcal{Q}$. We do the same for $\mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}}$ and $\leftrightarrow_{\mathcal{E}}^{root,semi}$.
- We adjust Lemma 6 to this new setting, to say: if $\leftrightarrow_{\mathcal{E}} \subseteq \leftrightarrow^*_{\mathcal{R}^{\mathcal{Q}}}$ on ground terms over $\mathcal{Q}$, then every equation in $\mathcal{E}$ is an inductive theorem of $\mathcal{Q}$. (The proof is immediate.)

Using the same reasoning as in Appendix A.2 (but with $\mathcal{Q}$ in place of $\mathcal{L}$), it is straightforward to check that Theorem 2 holds if the following lemma holds:

Lemma 28 (Main lemma). *Suppose that $(\mathcal{E}, \mathcal{H}) \vdash (\mathcal{E}', \mathcal{H}')$, using global rewriting induction. Then*

(1) $\leftrightarrow^{root,semi}_{\mathcal{E}\setminus\mathcal{E}'} \subseteq \rightarrow^*_{\mathcal{R}^{\mathcal{Q}}\cup\mathcal{H}'} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}'} \cdot \leftarrow^*_{\mathcal{R}^{\mathcal{Q}}\cup\mathcal{H}'}$

(2) $\rightarrow_{\mathcal{R}^{\mathcal{Q}}\cup\mathcal{H}} \subseteq \rightarrow_{\mathcal{R}^{\mathcal{Q}}} \cdot \rightarrow^*_{\mathcal{R}^{\mathcal{Q}}\cup\mathcal{H}} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}} \cdot \leftarrow^*_{\mathcal{R}^{\mathcal{Q}}\cup\mathcal{H}}$ *on ground terms over* $\mathcal{Q}$, *implies*
$\rightarrow_{\mathcal{R}^{\mathcal{Q}}\cup\mathcal{H}'} \subseteq \rightarrow_{\mathcal{R}^{\mathcal{Q}}} \cdot \rightarrow^*_{\mathcal{R}^{\mathcal{Q}}\cup\mathcal{H}'} \cdot \mathrel{\leftarrow\!\!\Vert\!\!\rightarrow}_{\mathcal{E}'} \cdot \leftarrow^*_{\mathcal{R}^{\mathcal{Q}}\cup\mathcal{H}'}$ *on ground terms over* $\mathcal{Q}$.

C.3 Soundness proof

Similar to the proof in Appendix A.3, we show that Lemma 28 holds for every deduction rule, but this time we have to work with $\mathcal{Q}$ instead of $\mathcal{L}$. Except for Expansion (which now becomes Global Expansion), it is not difficult to see that the proofs in Appendix A are easily adapted to this new situation: we just replace the gsc substitutions over $\mathcal{L}$ by gsc substitutions over $\mathcal{Q}$, and we are allowed to reduce with $\mathcal{R}^{\mathcal{Q}}$ instead of only $\mathcal{R}$. Aside from this:

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

Global Expansion Similar to Expansion, we need to prove both (1) and (2) of Lemma 28. To prove (1), we first introduce two helper lemmas.

Lemma 29. *Let* $\mathsf{f}\ s_1 \cdots s_n$ *be a ground term over* $\mathcal{Q}$ *such that*

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

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

Proof. Let $s := \mathsf{f}\ s_1 \cdots s_n$, $s' := \mathsf{f}\ \mu_1(s_1) \cdots \mu_n(s_n)$ and $\delta := [F_p := s|_p \mid$ all variables F_p in $\mathtt{Var}(s')]$. Since $\mathsf{f} \in \mathcal{D}$, 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 $s' = \mathsf{f}\ \mu_1(s_1) \cdots \mu_n(s_n)$ has a defined symbol at the head, with $n \geq ar(\mathsf{f})$ and all $\mu(s_i)$ are CHV terms over the original signature, strong quasi-reductivity indicates 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 $\mathsf{f}\ \ell_1 \cdots \ell_k \to r\ [\varphi] \in \mathcal{R}$ and a substitution η over $\mathcal{L}$ respecting this rule such that $\mathsf{f}\ \mu_1(s_1) \cdots \mu_k(s_k) = \mathsf{f}\ \ell_1 \cdots \ell_k$. We are done choosing $\gamma := \eta\delta$. ☐

Lemma 30. *If $(\mathcal{E} \uplus \{s \approx t\ [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E} \cup Expd(s \approx t\ [\varphi], p), \mathcal{H} \cup \{s \to t\ [\varphi]\})$ by **(Global Expansion)**, and γ is a ground semi-constructor substitution over $\mathcal{Q}$ that respects the equation $s \approx t\ [\varphi]$, then $s\gamma \to_{\mathcal{R}} \cdot \leftrightarrow\!\!\!\!\!\!\mid\!\mid_{\mathcal{E} \cup Expd(s \approx t\ [\varphi], p)} t\gamma$.*

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

But then any subterm $(s_i\gamma)|_q$ of base type is of the shape $\mathsf{c}\ (t_1\gamma) \cdots (t_m\gamma)$ as well, either because $q \in Pos(s_i)$ 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 $q' \in Pos(\gamma(x))$ (and γ is a gsc substitution). Therefore, by Lemma 29, $\mathsf{f}\ (s_1\gamma) \cdots (s_n\gamma) = \ell\delta$ for some $\ell \to r\ [\psi] \in \mathcal{R}$ and substitution δ over $\mathcal{Q}$ that respects ψ. Now, we finish the proof exactly as in Lemma 17. ☐

Part (1) of Lemma 28 is now proved by the following:

Corollary 2. *If $(\mathcal{E} \uplus \{s \approx t\ [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E} \cup Expd(s \approx t\ [\varphi], p), \mathcal{H} \cup \{s \to t\ [\varphi]\})$ by **(Global Expansion)** then*

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

And part (2) of Lemma 28 is proved by the following:

Lemma 31.
*Suppose $(\mathcal{E} \uplus \{s \approx t\ [\varphi]\}, \mathcal{H}) \vdash (\mathcal{E} \cup Expd(s \approx t\ [\varphi], p), \mathcal{H} \cup \{s \to t\ [\varphi]\})$ by **(Global Expansion)**, and $\to_{\mathcal{R}^{\mathcal{Q}} \cup \mathcal{H}}\ \subseteq\ \to_{\mathcal{R}^{\mathcal{Q}}} \cdot \to^*_{\mathcal{R}^{\mathcal{Q}} \cup \mathcal{H}} \cdot \leftrightarrow\!\!\!\!\!\!\mid\!\mid_{\mathcal{E} \cup \{s \approx t\ [\varphi]\}} \cdot \leftarrow^*_{\mathcal{R}^{\mathcal{Q}} \cup \mathcal{H}}$ on ground terms over $\mathcal{Q}$. Then:*

$$\begin{aligned}\to_{\mathcal{R}^{\mathcal{Q}} \cup \mathcal{H} \cup \{s \to t\ [\varphi]\}}\ \subseteq\ &\to_{\mathcal{R}^{\mathcal{Q}}} \cdot \to^*_{\mathcal{R}^{\mathcal{Q}} \cup \mathcal{H} \cup \{s \to t\ [\varphi]\}} \cdot \\ &\leftrightarrow\!\!\!\!\!\!\mid\!\mid_{\mathcal{E} \cup Expd(s \approx t\ [\varphi], p)} \cdot \leftarrow^*_{\mathcal{R}^{\mathcal{Q}} \cup \mathcal{H} \cup \{s \to t\ [\varphi]\}}\end{aligned}$$

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

Proof. With a similar reasoning as in the proof of Lemma 19: because of Corollary 2 it suffices to show that on ground terms over $\mathcal{Q}$ we have

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

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 $\delta = \gamma\downarrow_{\mathcal{R}^{\mathcal{Q}}}$ is a ground semi-constructor substitution over $\mathcal{Q}$, so by Lemma 30 (and the $\mathcal{Q}$-variant of Lemma 8) we have $s\delta \rightarrow_{\mathcal{R}} \cdot \leftrightarrow\!\!\!\!\!\!\mid\mid_{Exp(s\approx t\ [\varphi],p)} t\delta$. Therefore

$$C[s\gamma] \rightarrow^{*}_{\mathcal{R}^{\mathcal{Q}}} C[s\delta] \rightarrow_{\mathcal{R}} \cdot \leftrightarrow\!\!\!\!\!\!\mid\mid_{Exp(s\approx t\ [\varphi],p)} C[t\delta] \leftarrow^{*}_{\mathcal{R}^{\mathcal{Q}}} C[t\gamma]$$

So for sure

$$C[s\gamma] \rightarrow^{*}_{\mathcal{R}^{\mathcal{Q}}} C[s\delta] \rightarrow_{\mathcal{R}^{\mathcal{Q}}} \cdot \leftrightarrow\!\!\!\!\!\!\mid\mid_{Exp(s\approx t\ [\varphi],p)} C[t\delta] \leftarrow^{*}_{\mathcal{R}^{\mathcal{Q}}} C[t\gamma]$$

□