Inductive Theorem Proving in Non-terminating Rewriting Systems and Its Application to Program Transformation

Kentaro Kikuchi
Research Institute of Electrical
Communication
Tohoku University
Sendai, Japan
kentaro.kikuchi@iec.tohoku.ac.jp

Takahito Aoto
Faculty of Engineering
Niigata University
Niigata, Japan
aoto@ie.niigata-u.ac.jp

Isao Sasano
Department of Computer Science and
Engineering
Shibaura Institute of Technology
Tokyo, Japan
sasano@sic.shibaura-it.ac.jp

Abstract

We present a framework for proving inductive theorems of first-order equational theories, using techniques of implicit induction developed in the field of term rewriting. In this framework, we make use of automated confluence provers, which have recently been developed intensively, as well as a novel condition of sufficient completeness, called local sufficient completeness. The condition is a key to automated proof of inductive theorems of term rewriting systems that include non-terminating functions. We also apply the technique to showing the correctness of program transformation that is realised as an equivalence transformation of term rewriting systems.

CCS Concepts • **Theory of computation** → **Equational logic and rewriting; Program verification; Rewrite systems; Automated reasoning.**

Keywords Inductive Theorem Proving, Program Transformation, Infinite List, Term Rewriting, Non-termination, Confluence, Sufficient Completeness

ACM Reference Format:

Kentaro Kikuchi, Takahito Aoto, and Isao Sasano. 2019. Inductive Theorem Proving in Non-terminating Rewriting Systems and Its Application to Program Transformation. In *Principles and Practice of Declarative Programming 2019 (PPDP '19)*, October 7–9, 2019, Porto, Portugal. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3354166.3354178>

1 Introduction

Reasoning on programs heavily relies on some kind of induction. In first-order equational theory, reasoning using

induction coincides with initial algebra semantics, and such validity is better known as *inductive theorems* in the field of term rewriting. Proving inductive theorems has been not only especially of interest in the field of algebraic specifications but also heavily used in formal proof assistant systems. In this paper, we present a framework for proving inductive theorems of first-order equational theories, making use of various techniques from the field of term rewriting.

Many methods for proving inductive theorems have been developed in the field of term rewriting (e.g. [1, 3, 7, 11–13, 15, 19, 21, 24]). In most cases, efforts have been devoted to those function specifications for which computations are always terminating. It is, however, common for computation to be non-terminating. Let us consider the Haskell program in Figure 1 as an example that induces non-terminating computation. The function `replicate` takes a natural number n of type `Nat` and returns a list of ones (`Suc Z`'s) of length n . The list can also be constructed by taking the first n elements from the infinite list `ones`, each element of which is one (`Suc Z`). Although evaluating `ones` induces non-terminating computation, when `ones` is given as an argument together with n to `mytake`, the computation terminates and results in the same list, by utilising the lazy evaluation, the evaluation strategy in Haskell. As in this example, it is often the case that programs have expressions or statements for which computations do not terminate. Thus, proving inductive theorems for such programs is out of the scope of, e.g., rewriting induction methods [7, 19] which require termination (strong normalisation) of term rewriting systems corresponding to programs.

For proving inductive theorems for such programs, *inductionless induction methods* [12, 13, 17] may be employed. This is because, in contrast to rewriting induction methods, confluence rather than termination plays a key role in inductionless induction methods. However, in previous work on these methods, it has been assumed that every computation from a function with an input yields some value (known as the *sufficient completeness* property). This condition makes the methods not applicable to the program above, in which,

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

PPDP '19, October 7–9, 2019, Porto, Portugal

© 2019 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-7249-7/19/10.

<https://doi.org/10.1145/3354166.3354178>

```

data Nat = Z | Suc Nat deriving Show
replicate Z = []
replicate (Suc n) = Suc Z : replicate n
mytake (Z, xs) = []
mytake (Suc n, []) = []
mytake (Suc n, x:xs) = x : mytake (n, xs)
ones = Suc Z : ones
{- conjecture
  replicate n = mytake (n, ones)
-}

```

Figure 1. An inductive theorem for a Haskell program

for example, no computation starting from `ones` yields a value.

Our first key observation is that in inductionless induction methods, it is not essential that every computation yields a value, but only some restricted class of computations have to do it. In the case of an equational inductive theorem, the restricted class consists of every computation starting from instances of the left and right hand sides of the equation. We call the notion of sufficient completeness with such kind of restriction *local sufficient completeness*. The main novelty of the present paper is the use of this notion to prove inductive theorems for programs which admit computations not yielding any value, and to prove the correctness of transformation of such programs, which is realised as equivalence transformation of term rewriting systems [9, 24].

Another point that motivates our study is the recent development of automated confluence provers for term rewriting systems. The confluence property, a general notion of determinism, has a long history for its study in the fields of lambda calculus and term rewriting. Recently, renewed interest for proving confluence of term rewriting systems *automatically* arises by development of several confluence tools and starting of confluence competitions [2], which have been held yearly from 2012. We can benefit from availability of such confluence tools to verify the confluence property, which plays a key role in inductionless induction methods.

Overview of our work In our framework, programs are modelled as (monomorphic) first-order many-sorted term rewriting systems (TRSs, for short). Thus, for example, the Haskell program in Figure 1 is modelled as the following TRS $\mathcal{R}_{repl}$. (For a precise signature of $\mathcal{R}_{repl}$, see Example 3.5.)

$$\mathcal{R}_{repl} \left\{ \begin{array}{ll} \text{replicate}(0) & \rightarrow [] \\ \text{replicate}(s(n)) & \rightarrow s(0) : \text{replicate}(n) \\ \text{take}(0, xs) & \rightarrow [] \\ \text{take}(s(n), []) & \rightarrow [] \\ \text{take}(s(n), x : xs) & \rightarrow x : \text{take}(n, xs) \\ \text{ones} & \rightarrow s(0) : \text{ones} \end{array} \right.$$

The conjecture indicated in Figure 1 turns into the equation:

$$\text{replicate}(n) \doteq \text{take}(n, \text{ones})$$

Our method for proving inductive theorems aims to ensure that the results of evaluation (in Haskell) of the left and right hand sides of the equation, with n instantiated as an input, always coincide. To apply our inductionless induction method to this case, there are two main requirements: local sufficient completeness for the left and right hand sides of the equation, and confluence of the TRS $\mathcal{R}_{repl}$ augmented with the rewrite rule $\text{replicate}(n) \rightarrow \text{take}(n, \text{ones})$ (Theorem 3.2). We aim to automate checking these conditions. To check the former, we present a derivation system to infer whether terms satisfy local sufficient completeness (Figure 3). To check the latter, we make use of an automated confluence prover ACP [4]. We demonstrate several examples of proving inductive theorems in this way.

We then apply the technique to showing the correctness of program transformation, called equivalence transformation of TRSs [9, 24]. This transformation consists of inference rules that are enough to simulate the well-known program transformation by Burstall and Darlington [8]. Our notion of local sufficient completeness is useful in ensuring the correctness of the transformation of TRSs that are not necessarily sufficiently complete for all terms.

Contributions of the paper The contributions of this paper are summarised as follows:

- We introduce the notion of local sufficient completeness, and a derivation system to infer whether terms satisfy the condition of local sufficient completeness. A proof of the correctness of the system is presented.
- We present an inductionless induction method that is effective in proving inductive theorems of TRSs that are not necessarily sufficiently complete for all terms.
- We prove the correctness of equivalence transformation of TRSs. The proof improves on the previous one [9] so that it can work on any subset of terms that are sufficiently complete under the TRSs.

Related works Sufficient completeness, which is a special case of our local sufficient completeness, is originally defined not w.r.t. reduction but w.r.t. conversion; this property has been studied in many works in the context of abstract data types as well as in the context of inductionless induction (e.g. [10, 12–14, 16]). These results concern the quasi-reducibility property (in possibly non-free constructor case). Since we restrict our attention to TRSs with free constructors, decidability of quasi-reducibility and more efficient decision procedure of quasi-reducibility for left-linear TRSs only concern our setting. For sufficient completeness w.r.t. reduction, as in the present paper, an automated proof method has been investigated in [6]. However, it only deals with terminating (conditional) TRSs, and the method is not applicable to non-terminating TRSs. In [24], a proof method for sufficient completeness w.r.t. reduction that can deal with possibly non-terminating TRSs has been reported. The method is,

however, not applicable to local sufficient completeness in general, and some of the conditions for applying the method are not suitable to automation.

It is usual that inductionless induction methods are accomplished with some kind of completion, namely, transformation of TRSs adding and modifying rewrite rules to obtain an equivalent complete (i.e. terminating and confluent) TRSs (e.g. [12, 13]). Since our target programs may contain non-terminating functions that intend to generate infinite data, it is not possible that completion generates terminating TRSs, and so it is not effective in our setting.

Organisation of the paper The paper is organised as follows. In Section 2, we explain basic notions and notations of term rewriting. In Section 3, we present our inductionless induction method and give examples of inductive theorem proving. In Section 4, we discuss equivalence transformation of TRSs. In Section 5, we introduce a derivation system to check local sufficient completeness, and conclude in Section 6.

2 Preliminaries

In this section, we introduce notations and notions from (many-sorted) term rewriting, which are used throughout the paper. For detailed information about term rewriting, see, e.g. [22].

A *many-sorted signature* is given by a non-empty set $\mathcal{S}$ of *sorts* and a set $\mathcal{F}$ of *function symbols*; each $f \in \mathcal{F}$ has its *sort declaration* $f : \alpha_1 \times \dots \times \alpha_n \rightarrow \alpha_0$ where $\alpha_0, \dots, \alpha_n \in \mathcal{S}$ ($n \geq 0$), and n is called the *arity* of f , denoted by $\text{arity}(f)$. We use $\mathcal{V}$ to denote the set of *variables* where each $x \in \mathcal{V}$ has a unique sort $\alpha \in \mathcal{S}$. Then the set $T^\alpha(\mathcal{F}, \mathcal{V})$ of *terms of sort α* is defined inductively as follows:

1. If a variable $x \in \mathcal{V}$ has a sort α , then $x \in T^\alpha(\mathcal{F}, \mathcal{V})$.
2. If a function symbol $f \in \mathcal{F}$ is equipped with its sort declaration $f : \alpha_1 \times \dots \times \alpha_n \rightarrow \alpha$ and $t_i \in T^{\alpha_i}(\mathcal{F}, \mathcal{V})$ for all $i \in \{1, \dots, n\}$, then $f(t_1, \dots, t_n) \in T^\alpha(\mathcal{F}, \mathcal{V})$.

We define $T(\mathcal{F}, \mathcal{V})$ as $\bigcup_{\alpha \in \mathcal{S}} T^\alpha(\mathcal{F}, \mathcal{V})$. The symbol $\equiv$ is used to denote syntactic equality of terms.

For a term $t \equiv f(t_1, \dots, t_n)$, its *root symbol* f is denoted by $\text{root}(t)$. The sets of function symbols and variables in a term t are denoted by $\mathcal{F}(t)$ and $\mathcal{V}(t)$, respectively. A term t is *ground* if $\mathcal{V}(t) = \emptyset$; the set of ground terms is denoted by $T(\mathcal{F})$. A term t is *linear* if each variable occurs at most once in t . We use $\uplus$ for the disjoint union. We consider a partition of function symbols $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$; function symbols in $\mathcal{D}$ are called *defined symbols*, and those in $\mathcal{C}$ are *constructors*. Terms in $T(\mathcal{C}, \mathcal{V})$ are called *constructor terms*, and those in $T(\mathcal{F})$ are *ground constructor terms*. We use v, v' , etc. for ground constructor terms.

A *substitution* is a sort-preserving mapping $\theta : \mathcal{V} \rightarrow T(\mathcal{F}, \mathcal{V})$ such that $\text{dom}(\theta) = \{x \in \mathcal{V} \mid \theta(x) \neq x\}$ is finite. The term obtained by applying a substitution θ to a term t is written as $t\theta$. A substitution θ is *ground* if $x\theta \in T(\mathcal{F})$

for all $x \in \text{dom}(\theta)$. Throughout the paper, θ_g, θ'_g , etc. denote ground substitutions, and whenever an expression like $t\theta_g$ is used, we assume $\mathcal{V}(t\theta_g) = \emptyset$ (i.e. $\mathcal{V}(t) \subseteq \text{dom}(\theta_g)$). A *context* $C[]$ is a term with a sorted hole, and $C[t]$ denotes the term obtained by filling the hole with a term t of the same sort.

An *equation* is a pair of terms s and t of the same sort, written as $s \doteq t$. Equations $s \doteq t$ and $t \doteq s$ are identified. A *rewrite rule*, written as $l \rightarrow r$, is an ordered pair of terms l and r of the same sort such that $l \notin \mathcal{V}$, $\mathcal{V}(l) \supseteq \mathcal{V}(r)$ and $\text{root}(l) \in \mathcal{D}$. We assume that variables in rewrite rules and equations are renamed when necessary. A rewrite rule $l \rightarrow r$ is *left-linear* if l is linear. A rewrite rule $f(l_1, \dots, l_n) \rightarrow r$ is a *constructor rule* if each l_i ($i \in \{1, \dots, n\}$) is a constructor term. A (many-sorted) *term rewriting system* (TRS, for short) is a finite set of rewrite rules. A TRS $\mathcal{R}$ is *left-linear* if all its rewrite rules are left-linear, and it is a *constructor TRS* if all its rewrite rules are constructor rules.

Let $\mathcal{R}$ be a TRS. The *rewrite relation* $\rightarrow_{\mathcal{R}}$ is the binary relation on $T(\mathcal{F}, \mathcal{V})$ defined by $s \rightarrow_{\mathcal{R}} t$ iff $s \equiv C[l\theta]$ and $t \equiv C[r\theta]$ for some $l \rightarrow r \in \mathcal{R}$, a context $C[]$ and a substitution θ . The inverse of $\rightarrow_{\mathcal{R}}$ is denoted by $\leftarrow_{\mathcal{R}}$. The reflexive transitive closure of $\rightarrow_{\mathcal{R}}$ ($\leftarrow_{\mathcal{R}}$) is denoted by $\rightarrow^*_{\mathcal{R}}$ ($\leftarrow^*_{\mathcal{R}}$, respectively), and the reflexive transitive symmetric closure of $\rightarrow_{\mathcal{R}}$ is denoted by $\leftrightarrow^*_{\mathcal{R}}$. A term t is in *normal form* (with respect to $\rightarrow_{\mathcal{R}}$) if $t \rightarrow_{\mathcal{R}} u$ for no term u . $\text{NF}(\mathcal{R})$ denotes the set of terms in normal form. For a set $\mathcal{E}$ of equations, $\leftrightarrow_{\mathcal{E}}$ is defined by $s \leftrightarrow_{\mathcal{E}} t$ iff $s \equiv C[l\theta]$ and $t \equiv C[r\theta]$ or $s \equiv C[r\theta]$ and $t \equiv C[l\theta]$ for some $l \doteq r \in \mathcal{E}$, a context $C[]$ and a substitution θ . The reflexive transitive closure of $\rightarrow_{\mathcal{R}} \cup \leftarrow_{\mathcal{R}} \cup \leftrightarrow_{\mathcal{E}}$ is denoted by $\leftrightarrow^*_{\mathcal{R} \cup \mathcal{E}}$.

A TRS $\mathcal{R}$ is *confluent* if for every $s, s', s'' \in T(\mathcal{F}, \mathcal{V})$, $s \rightarrow^*_{\mathcal{R}} s'$ and $s \rightarrow^*_{\mathcal{R}} s''$ imply that there exists $t \in T(\mathcal{F}, \mathcal{V})$ such that $s' \rightarrow^*_{\mathcal{R}} t$ and $s'' \rightarrow^*_{\mathcal{R}} t$. A TRS $\mathcal{R}$ is *terminating* or *strongly normalising* if there exists no infinite rewrite sequence $s_1 \rightarrow_{\mathcal{R}} s_2 \rightarrow_{\mathcal{R}} s_3 \rightarrow_{\mathcal{R}} \dots$. We use $\text{CR}(\mathcal{R})$ and $\text{SN}(\mathcal{R})$ to denote that $\mathcal{R}$ is confluent and strongly normalising, respectively.

Since our definition of rewrite rule $l \rightarrow r$ requires the condition $\text{root}(l) \in \mathcal{D}$, we have the following.

Proposition 2.1. *For every constructor term $t \in T(\mathcal{C}, \mathcal{V})$, t is in normal form.*

Proposition 2.2. *Let $\mathcal{R}$ be a confluent TRS. Then, if $s, t \in \text{NF}(\mathcal{R})$ and $s \leftrightarrow^*_{\mathcal{R}} t$ then $s \equiv t$.*

In the rest of the paper, we often use the above propositions without any reference.

Next we introduce a key notion in this paper.

Definition 2.3 (Local sufficient completeness). Consider a signature with a set $\mathcal{F} (= \mathcal{D} \uplus \mathcal{C})$ of function symbols. Let $\mathcal{R}$ be a TRS.

1. A ground term $t \in T(\mathcal{F})$ is *sufficiently complete under $\mathcal{R}$* if there exists $v \in T(\mathcal{C})$ such that $t \rightarrow^*_{\mathcal{R}} v$.

2. A term $t \in T(\mathcal{F}, \mathcal{V})$ is *sufficiently complete under $\mathcal{R}$* if for every ground substitution θ_g , $t\theta_g$ is sufficiently complete under $\mathcal{R}$.
3. A set $\mathcal{A} \subseteq T(\mathcal{F}, \mathcal{V})$ is *sufficiently complete under $\mathcal{R}$* , or $\mathcal{R}$ is *sufficiently complete on $\mathcal{A}$* , if every term $t \in \mathcal{A}$ is sufficiently complete under $\mathcal{R}$.

We use $SC(\mathcal{R}, t)$ and $SC(\mathcal{R}, \mathcal{A})$ to denote that t and $\mathcal{A}$ are sufficiently complete under $\mathcal{R}$, respectively. We also use $SC(t)$ and $SC(\mathcal{A})$ when $\mathcal{R}$ is understood from the context. A standard notion of sufficient completeness is the case where $\mathcal{A} = T(\mathcal{F})$ in the above definition.

A TRS $\mathcal{R}$ is said to be *quasi-reducible* if $f(v_1, \dots, v_n) \notin NF(\mathcal{R})$ for any $f \in \mathcal{D}$ with $arity(f) = n$ and any $v_1, \dots, v_n \in T(\mathcal{C})$. The next proposition provides a criterion for sufficient completeness on $T(\mathcal{F})$. (Closely related statements can be found in [16, 18].)

Proposition 2.4. *Let $\mathcal{R}$ be a strongly normalising TRS. Then, $\mathcal{R}$ is sufficiently complete on $T(\mathcal{F})$ if and only if $\mathcal{R}$ is quasi-reducible.*

Proof. Suppose that $\mathcal{R}$ is strongly normalising. For the only if part, let $\mathcal{R}$ be sufficiently complete on $T(\mathcal{F})$, and let $t \equiv f(v_1, \dots, v_n)$ for $f \in \mathcal{D}$ and $v_1, \dots, v_n \in T(\mathcal{C})$. Then $t \xrightarrow{*}_{\mathcal{R}} u$ for some $u \in T(\mathcal{C})$. Since $t \notin T(\mathcal{C})$, we have $t \not\equiv u$. Hence $t \notin NF(\mathcal{R})$.

For the if part, let $\mathcal{R}$ be quasi-reducible. We show by Noetherian induction on $\rightarrow_{\mathcal{R}}$ that $\mathcal{R}$ is sufficiently complete on $T(\mathcal{F})$. Let $t \in T(\mathcal{F})$. If $t \in T(\mathcal{C})$ then $(\exists u \in T(\mathcal{C}). t \xrightarrow{*}_{\mathcal{R}} u)$ holds. Suppose $t \notin T(\mathcal{C})$. Then there exists $f \in \mathcal{F}(t) \cap \mathcal{D}$. In particular, there exists a subterm s of t such that $s \equiv f(v_1, \dots, v_n)$ for $f \in \mathcal{D}$ and $v_1, \dots, v_n \in T(\mathcal{C})$. Then by quasi-reducibility of $\mathcal{R}$, $s \notin NF(\mathcal{R})$, and hence $t \equiv C[s] \rightarrow_{\mathcal{R}} t'$ for some t' . Then by induction hypothesis, $t' \xrightarrow{*}_{\mathcal{R}} u$ for some $u \in T(\mathcal{C})$. Thus, $(\exists u \in T(\mathcal{C}). t \xrightarrow{*}_{\mathcal{R}} u)$ holds. $\square$

Since quasi-reducibility for TRSs is decidable [10], sufficient completeness on $T(\mathcal{F})$ is decidable for strongly normalising TRSs.

3 Inductive Theorem Proving in TRSs

In this section, we present a method for proving inductive theorems of TRSs, and give several examples to explain how it works. The conditions for applying the method concern local sufficient completeness and confluence, and we propose methods for checking them automatically.

Let $\mathcal{R}$ be a TRS. An equation $s \doteq t$ is an *inductive theorem of $\mathcal{R}$* if $s\theta_g \xrightarrow{*}_{\mathcal{R}} t\theta_g$ for every ground substitution θ_g . We write $\mathcal{R} \models_{ind} \mathcal{E}$ for a set $\mathcal{E}$ of equations if every $s \doteq t \in \mathcal{E}$ is an inductive theorem of $\mathcal{R}$.

Example 3.1. Consider a signature with $\mathcal{S} = \{\text{Nat}\}$ and $\mathcal{F} = \{+ : \text{Nat} \times \text{Nat} \rightarrow \text{Nat}, s : \text{Nat} \rightarrow \text{Nat}, 0 : \text{Nat}\}$. Let $\mathcal{R}_{add}$ be the following TRS for the addition of natural

numbers:

$$\mathcal{R}_{add} \begin{cases} 0 + y & \rightarrow y \\ s(x) + y & \rightarrow s(x + y) \end{cases}$$

We consider the following equation:

$$x + 0 \doteq x$$

It is seen that $x + 0 \xrightarrow{*}_{\mathcal{R}_{add}} x$ does not hold, since both $x + 0$ and x are in normal form with respect to $\rightarrow_{\mathcal{R}_{add}}$. However, the equation is an inductive theorem of $\mathcal{R}_{add}$, i.e., $(x + 0)\theta_g \xrightarrow{*}_{\mathcal{R}_{add}} x\theta_g$ for every ground substitution θ_g . (For a proof of this claim, see Example 3.3.) This shows that an inductive theorem is not necessarily a theorem in equational logic.

The next theorem differs from previous work on inductionless induction methods in that, for an equation $s \doteq t$ to be an inductive theorem, it requires sufficient completeness only for s and t instead of for all ground terms.

Theorem 3.2. *Consider a signature with a set $\mathcal{F} (= \mathcal{D} \uplus \mathcal{C})$ of function symbols. Suppose that a TRS $\mathcal{R}$ and an equation $s \doteq t$ satisfy the following conditions:*

1. *s and t are sufficiently complete under $\mathcal{R}$.*
2. *$s \rightarrow t$ is a rewrite rule.*
3. *The TRS $\mathcal{R} \cup \{s \rightarrow t\}$ is confluent.*

Then $s \doteq t$ is an inductive theorem of $\mathcal{R}$.

Proof. Let $\mathcal{R}$ be a TRS and let $s, t \in T^{\alpha}(\mathcal{F}, \mathcal{V})$. Suppose that they satisfy the conditions 1, 2 and 3. To show the claim, let θ_g be any ground substitution. Then by the condition 1, there exist $v, v' \in T(\mathcal{C})$ such that $s\theta_g \xrightarrow{*}_{\mathcal{R}} v$ and $t\theta_g \xrightarrow{*}_{\mathcal{R}} v'$. Since $s\theta_g \rightarrow_{\mathcal{R} \cup \{s \rightarrow t\}} t\theta_g$ holds, we have $v \xleftarrow{*}_{\mathcal{R}} s\theta_g \xrightarrow{*}_{\mathcal{R} \cup \{s \rightarrow t\}} v'$. By the condition 3, the TRS $\mathcal{R} \cup \{s \rightarrow t\}$ is confluent, and hence we have $v \equiv v'$ (by Proposition 2.2). Now we have $s\theta_g \xrightarrow{*}_{\mathcal{R}} v \equiv v' \xleftarrow{*}_{\mathcal{R}} t\theta_g$, which implies $s\theta_g \xrightarrow{*}_{\mathcal{R}} t\theta_g$. Thus we conclude that $s \doteq t$ is an inductive theorem of $\mathcal{R}$.

$$\begin{array}{ccc} s\theta_g & \xrightarrow[\{s \rightarrow t\}]{} & t\theta_g \\ * \downarrow R & & * \downarrow R \\ v & \equiv & v' \\ & \text{by CR}(\mathcal{R} \cup \{s \rightarrow t\}) & \end{array} \quad \square$$

In what follows, we give several examples to show that Theorem 3.2 is useful for proving inductive theorems of various TRSs. Some of these examples are used to ensure the correctness of equivalence transformations of TRSs in Section 4.

Example 3.3. Consider the TRS $\mathcal{R}_{add}$ and the equation $x + 0 \doteq x$ in Example 3.1. We show that $x + 0 \doteq x$ is an inductive theorem of $\mathcal{R}_{add}$ by applying Theorem 3.2.

Consider $x + 0 \rightarrow x$, which is clearly a rewrite rule (so the condition 2 is satisfied). To verify the condition 3, i.e., the TRS $\mathcal{R}_{add} \cup \{x + 0 \rightarrow x\}$ is confluent, we rely on the system ACP, an automated confluence prover which originates in [4] and then have been developed rapidly.

The input file for ACP, which represents the TRS $\mathcal{R}_{add} \cup \{x + 0 \rightarrow x\}$, is as follows:

```
(VAR x y)

(RULES
  plus(zero,y) -> y
  plus(s(x),y) -> s(plus(x,y))
  plus(x,zero) -> x
)
```

Then the output of ACP is like this:

```
YES
Rewrite Rules:
  [ plus(zero,?y) -> ?y,
    plus(s(?x),?y) -> s(plus(?x,?y)),
    plus(?x,zero) -> ?x ]
Apply Direct Methods...
Inner CPs:
  [ ]
Outer CPs:
  [ zero = zero,
    s(plus(?x_1,zero)) = s(?x_1) ]
Overlay, check Innermost Termination...
Innermost Terminating (hence Terminating), WCR
Knuth & Bendix
Direct Methods: CR

Combined result: CR
add.trs: Success(CR)
(4 msec.)
```

which means that the TRS $\mathcal{R}_{add} \cup \{x + 0 \rightarrow x\}$ can be proved confluent with appropriate methods. Also, as indicated in the above output, the TRS is terminating (strongly normalising).¹ It is easily seen that $\mathcal{R}_{add}$ is quasi-reducible, so by Proposition 2.4, $\mathcal{R}_{add}$ is sufficiently complete on $T(\mathcal{F})$. Hence the condition 1 in Theorem 3.2 is satisfied. Thus by Theorem 3.2, we see that $x + 0 \doteq x$ is an inductive theorem of $\mathcal{R}_{add}$.

Example 3.4. Consider a signature with $\mathcal{S} = \{\text{Nat}\}$ and $\mathcal{F} = \{\text{sum} : \text{Nat} \rightarrow \text{Nat}, + : \text{Nat} \times \text{Nat} \rightarrow \text{Nat}, s : \text{Nat} \rightarrow \text{Nat}, 0 : \text{Nat}\}$. Let $\mathcal{R}_{sum}$ be the following TRS for the summation function $\text{sum}(n) = n + \dots + 1 + 0$:

$$\mathcal{R}_{sum} \left\{ \begin{array}{ll} \text{sum}(0) & \rightarrow 0 \\ \text{sum}(s(x)) & \rightarrow s(x) + \text{sum}(x) \\ x + 0 & \rightarrow x \\ x + s(y) & \rightarrow s(x + y) \end{array} \right.$$

We show that the following equation is an inductive theorem of $\mathcal{R}_{sum}$ by applying Theorem 3.2:

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

The output of ACP for an input file representing the TRS $\mathcal{R}_{sum} \cup \{x + (y + z) \rightarrow (x + y) + z\}$ is shown in Figure 2. Thus the TRS is confluent. Also, as indicated in the output, the

TRS is terminating (strongly normalising). It is easily seen that $\mathcal{R}_{sum}$ is quasi-reducible, so by Proposition 2.4, $\mathcal{R}_{sum}$ is sufficiently complete on $T(\mathcal{F})$, which implies that $x + (y + z)$ and $(x + y) + z$ are sufficiently complete under $\mathcal{R}_{sum}$. Hence by Theorem 3.2, the equation $x + (y + z) \doteq (x + y) + z$ is an inductive theorem of $\mathcal{R}_{sum}$.

The next two examples demonstrate that our method for proving inductive theorems works for TRSs under which not all terms are sufficiently complete. Example 3.5 shows that the conjecture in Figure 1 is indeed an inductive theorem of the TRS corresponding to the Haskell program.

Example 3.5. Consider a signature with $\mathcal{S} = \{\text{Nat}, \text{List}\}$ and $\mathcal{F} = \{\text{replicate} : \text{Nat} \rightarrow \text{List}, \text{take} : \text{Nat} \times \text{List} \rightarrow \text{List}, \text{ones} : \text{List}, : : \text{Nat} \times \text{List} \rightarrow \text{List}, [] : \text{List}, s : \text{Nat} \rightarrow \text{Nat}, 0 : \text{Nat}\}$. Let $\mathcal{R}_{repl}$ be the following TRS:

$$\mathcal{R}_{repl} \left\{ \begin{array}{ll} \text{replicate}(0) & \rightarrow [] \\ \text{replicate}(s(n)) & \rightarrow s(0) : \text{replicate}(n) \\ \text{take}(0, xs) & \rightarrow [] \\ \text{take}(s(n), []) & \rightarrow [] \\ \text{take}(s(n), x : xs) & \rightarrow x : \text{take}(n, xs) \\ \text{ones} & \rightarrow s(0) : \text{ones} \end{array} \right.$$

We show that the following equation is an inductive theorem of $\mathcal{R}_{repl}$ by applying Theorem 3.2:

$$\text{replicate}(n) \doteq \text{take}(n, \text{ones})$$

The output of ACP for an input file representing the TRS $\mathcal{R}_{repl} \cup \{\text{replicate}(n) \rightarrow \text{take}(n, \text{ones})\}$ shows success in checking confluence. Thus the TRS is confluent. Also, we see that $\text{replicate}(n)$ and $\text{take}(n, \text{ones})$ are sufficiently complete under $\mathcal{R}_{repl}$, using the procedure given in Section 5 (see Example 5.7). Hence by Theorem 3.2, the equation $\text{replicate}(n) \doteq \text{take}(n, \text{ones})$ is an inductive theorem of $\mathcal{R}_{repl}$.

Example 3.6. Consider a signature with $\mathcal{S} = \{\text{Nat}, \text{Stream}\}$ and $\mathcal{F} = \{\text{sumstr} : \text{Nat} \times \text{Stream} \rightarrow \text{Nat}, \text{hd} : \text{Stream} \rightarrow \text{Nat}, \text{tl} : \text{Stream} \rightarrow \text{Stream}, \text{succ} : \text{Stream} \rightarrow \text{Stream}, \text{inc} : \text{Stream}, : : \text{Nat} \times \text{Stream} \rightarrow \text{Stream}, + : \text{Nat} \times \text{Nat} \rightarrow \text{Nat}, s : \text{Nat} \rightarrow \text{Nat}, 0 : \text{Nat}\}$. Let $\mathcal{R}_{sumstr}$ be the following TRS where $\text{sumstr}(n, a)$ computes the summation of the first n elements of a stream a of natural numbers:

$$\mathcal{R}_{sumstr} \left\{ \begin{array}{ll} \text{sumstr}(0, a) & \rightarrow 0 \\ \text{sumstr}(s(n), a) & \rightarrow \text{hd}(a) + \text{sumstr}(n, \text{tl}(a)) \\ \text{hd}(x : a) & \rightarrow x \\ \text{tl}(x : a) & \rightarrow a \\ \text{inc} & \rightarrow 0 : \text{succ}(\text{inc}) \\ \text{succ}(x : a) & \rightarrow s(x) : \text{succ}(a) \\ 0 + y & \rightarrow y \\ s(x) + y & \rightarrow s(x + y) \end{array} \right.$$

We show that the following equations are inductive theorems of $\mathcal{R}_{sumstr}$ by applying Theorem 3.2:

$$\mathcal{E} \left\{ \begin{array}{ll} x + y & \doteq y + x \\ (x + y) + z & \doteq x + (y + z) \end{array} \right.$$

¹ ACP proves confluence and termination for unsorted TRSs, but those properties hold also for many-sorted TRSs that are obtained from the unsorted TRSs by sorting in appropriate ways.

```

YES
Rewrite Rules:
  [ sum(zero) -> zero,
    sum(s(?x)) -> plus(s(?x), sum(?x)),
    plus(?x, zero) -> ?x,
    plus(?x, s(?y)) -> s(plus(?x, ?y)),
    plus(?x, plus(?y, ?z)) -> plus(plus(?x, ?y), ?z) ]
Apply Direct Methods...
Inner CPs:
  [ plus(?x_3, ?x_1) = plus(plus(?x_3, ?x_1), zero),
    plus(?x_3, s(plus(?x_2, ?y_2))) = plus(plus(?x_3, ?x_2), s(?y_2)),
    plus(?x_1, plus(plus(?x, ?y), ?z)) = plus(plus(?x_1, ?x), plus(?y, ?z)) ]
Outer CPs:
  [ ]
not Overlay, check Termination...
Terminating, WCR
Knuth & Bendix
Direct Methods: CR

Combined result: CR
sumnat.trs: Success(CR)
(8 msec.)

```

Figure 2. The output of ACP for $\mathcal{R}_{sum} \cup \{x + (y + z) \rightarrow (x + y) + z\}$ in Example 3.4.

The outputs of ACP for input files representing the TRSs $\mathcal{R}_{sumstr} \cup \{x + y \rightarrow y + x\}$ and $\mathcal{R}_{sumstr} \cup \{(x + y) + z \rightarrow x + (y + z)\}$ both show success in checking confluence. Thus those TRSs are confluent. Also, it is seen that all terms in $T^{\text{Nat}}(\mathcal{F})$ are sufficiently complete under $\mathcal{R}_{sumstr}$, as shown in Section 5 (see Example 5.9). Hence by Theorem 3.2, the equations $x + y \doteq y + x$ and $(x + y) + z \doteq x + (y + z)$ are inductive theorems of $\mathcal{R}_{sumstr}$.

4 Equivalence Transformation of TRSs

In this section, we discuss a particular kind of program transformation, called equivalence transformation of TRSs, which is introduced in [24] and also studied in [9] with some modifications. The transformation consists of inference rules that are enough to simulate the well-known program transformation by Burstall and Darlington [8]. We follow the style of transformation in [9], where inductive theorem proving as in the previous section plays an important role.

First we give an example of equivalence transformation, which transforms the TRS $\mathcal{R}_{sum}$ in Example 3.4 into the tail-recursive form. This example is taken from [24, Example 7.1], but we reformulate the transformation in the style of [9].

Example 4.1. Consider the TRS $\mathcal{R}_{sum}$ in Example 3.4.

$$\mathcal{R}_{sum} \left\{ \begin{array}{ll} \text{sum}(0) & \rightarrow 0 \\ \text{sum}(s(x)) & \rightarrow s(x) + \text{sum}(x) \\ x + 0 & \rightarrow x \\ x + s(y) & \rightarrow s(x + y) \end{array} \right.$$

By the equivalence transformation we introduce in this section, $\mathcal{R}_{sum}$ can be transformed into the following TRS $\mathcal{R}'_{sum}$:

$$\mathcal{R}'_{sum} \left\{ \begin{array}{ll} \text{sum}(0) & \rightarrow 0 \\ \text{sum}(s(x)) & \rightarrow \text{sum1}(x, s(x)) \\ \text{sum1}(0, y) & \rightarrow y \\ \text{sum1}(s(x), y) & \rightarrow \text{sum1}(x, y + s(x)) \\ x + 0 & \rightarrow x \\ x + s(y) & \rightarrow s(x + y) \end{array} \right.$$

with the help of the following singleton set of equations:

$$\mathcal{E} \left\{ x + (y + z) \doteq (x + y) + z \right.$$

Note that $\mathcal{R}_{sum} \models_{ind} \mathcal{E}$ holds (see Example 3.4), which plays an important role in the equivalence transformation (see Example 4.9).

Before giving the definition of the equivalence transformation, we make a remark on discussing equivalence of TRSs over two different signatures. In the case of the above example, we have $\text{sum1}(0, s(0)) \rightarrow_{\mathcal{R}'_{sum}} s(0)$, but $\text{sum1}(0, s(0))$ is in normal form with respect to $\rightarrow_{\mathcal{R}_{sum}}$. So we cannot simply discuss equivalence of TRSs by their rewrite relations but need to take the signatures into account.

Given a signature with a set $\mathcal{F}$ of function symbols, we speak of a TRS $\mathcal{R}$ (or a set $\mathcal{E}$ of equations) *over* $\mathcal{F}$ when all its rewrite rules (resp. equations) consist of terms in $T(\mathcal{F}, \mathcal{V})$.

Now the equivalence transformation of TRSs, which is essentially the same as the one in [9], is defined as follows.²

² The only difference from the equivalent transformation in [9] is that we set $\mathcal{F}_{k+1} = \mathcal{F}_k$ in the Addition rule while in [9], $\mathcal{F}_{k+1}$ is any set of function

Definition 4.2 (Equivalence transformation). Let $\mathcal{R}_0$ be a left-linear constructor TRS over $\mathcal{F}_0$ and $\mathcal{E}$ be a set of equations over $\mathcal{F}_0$. An *equivalence transformation sequence under $\mathcal{E}$* is a sequence $\mathcal{R}_0, \dots, \mathcal{R}_n$ of TRSs (over $\mathcal{F}_0, \dots, \mathcal{F}_n$, respectively) such that $\mathcal{R}_{k+1}$ is obtained from $\mathcal{R}_k$ by applying one of the following inference rules:

(I) *Introduction*

$$\mathcal{R}_{k+1} = \mathcal{R}_k \cup \{f(x_1, \dots, x_m) \rightarrow r\}$$

provided that $f \notin \mathcal{F}_k$ and $r \in T(\mathcal{F}_k, \{x_1, \dots, x_m\})$ where $x_1, \dots, x_m$ are mutually distinct variables. We set $\mathcal{F}_{k+1} = \mathcal{F}_k \cup \{f\}$.

(A) *Addition*

$$\mathcal{R}_{k+1} = \mathcal{R}_k \cup \{l \rightarrow r\}$$

provided that $l \xrightarrow{\mathcal{R}_k \cup \mathcal{E}} r$ and $l \rightarrow r$ is a rewrite rule. We set $\mathcal{F}_{k+1} = \mathcal{F}_k$.

(E) *Elimination*

$$\mathcal{R}_{k+1} = \mathcal{R}_k \setminus \{l \rightarrow r\}$$

We define $\mathcal{F}_{k+1}$ as any set of function symbols such that $\mathcal{F}_{k+1} \subseteq \mathcal{F}_k$ and $\mathcal{R}_{k+1}$ is a TRS over $\mathcal{F}_{k+1}$.

We write $\mathcal{R}_k \Rightarrow \mathcal{R}_{k+1}$ when $\mathcal{R}_{k+1}$ is obtained from $\mathcal{R}_k$ by applying one of the above rules. The reflexive transitive closure of $\Rightarrow$ is denoted by $\Rightarrow^*$. We indicate the rule of $\Rightarrow$ by $\Rightarrow_I, \Rightarrow_A$, or $\Rightarrow_E$. Finally, we say that there exists an *equivalence transformation from $\mathcal{R}$ to $\mathcal{R}'$ under $\mathcal{E}$* if there exists an equivalence transformation sequence $\mathcal{R} \xRightarrow{I} \mathcal{R}_1 \xRightarrow{A} \dots \xRightarrow{E} \mathcal{R}'$ under $\mathcal{E}$.

In Theorem 4.7 below, we prove the correctness of the above equivalence transformation under some conditions. Before that, we show some lemmas on properties of the inference rules. First we show a lemma on the Introduction rule.

Lemma 4.3. *Suppose that there exists an equivalence transformation sequence $\mathcal{R}_0, \dots, \mathcal{R}_i$ (over $\mathcal{F}_0, \dots, \mathcal{F}_i$, respectively) such that $\mathcal{R}_0 \xRightarrow{I} \mathcal{R}_1 \xRightarrow{I} \dots \xRightarrow{I} \mathcal{R}_i$. For each $k \in \{0, \dots, i-1\}$, the following hold.*

1. If $s \rightarrow_{\mathcal{R}_k} t$, then $s \rightarrow_{\mathcal{R}_{k+1}} t$.
2. For every $t \in T(\mathcal{F}_{k+1})$, there exists $t' \in T(\mathcal{F}_k)$ such that $t \xrightarrow{\mathcal{R}_{k+1}} t'$.
3. If $\text{CR}(\mathcal{R}_k)$, then $\text{CR}(\mathcal{R}_{k+1})$.

Proof. 1. By the definition of the Introduction rule, we have $\mathcal{R}_k \subseteq \mathcal{R}_{k+1}$. Hence the claim follows.
2. By the definition of the Introduction rule, we can reduce the new function symbol $f(\notin \mathcal{F}_k)$ in t one by one using the rule $f(x_1, \dots, x_m) \rightarrow r$.

symbols such that $\mathcal{F}_{k+1} \subseteq \mathcal{F}_k$ and $\mathcal{R}_{k+1}$ is a TRS over $\mathcal{F}_{k+1}$. Since the rewrite rule $l \rightarrow r$ added in the Addition rule can always be considered over $\mathcal{F}_k$, this difference is not essential.

3. By the commutativity of mutually orthogonal TRSs, originally shown in [23]. See also [5, Section 9.3.2]. $\square$

Using induction on k in the above lemma, we have the following.

Corollary 4.4. *Suppose that there exists an equivalence transformation sequence $\mathcal{R}_0 \xRightarrow{I} \mathcal{R}_i$ where $\mathcal{R}_0$ and $\mathcal{R}_i$ are over $\mathcal{F}_0$ and $\mathcal{F}_i$, respectively. Then the following hold.*

1. If $s \rightarrow_{\mathcal{R}_0} t$, then $s \rightarrow_{\mathcal{R}_i} t$.
2. For every $t \in T(\mathcal{F}_i)$, there exists $t' \in T(\mathcal{F}_0)$ such that $t \xrightarrow{\mathcal{R}_i} t'$.
3. If $\text{CR}(\mathcal{R}_0)$, then $\text{CR}(\mathcal{R}_i)$.

Next we show a lemma on the Addition rule. In this lemma, we suppose $\mathcal{R}_0 \models_{\text{ind}} \mathcal{E}$, but it works only for terms in the signature with the set $\mathcal{F}_0$ of function symbols. So, to apply those inductive theorems in $\mathcal{E}$, we need to find appropriate ground substitutions whose ranges are included in $T(\mathcal{F}_0)$.

Lemma 4.5. *Suppose that there exists an equivalence transformation sequence $\mathcal{R}_0 \xRightarrow{I} \mathcal{R}_i \xRightarrow{A} \mathcal{R}_{i+1} \xRightarrow{A} \dots \xRightarrow{A} \mathcal{R}_a$ under $\mathcal{E}$ where $\mathcal{R}_0$ and $\mathcal{E}$ are over $\mathcal{F}_0$, and $\mathcal{R}_i, \mathcal{R}_{i+1}, \dots, \mathcal{R}_a$ are over $\mathcal{F}_i$. Suppose also that $\mathcal{R}_0 \models_{\text{ind}} \mathcal{E}$. Then, for all $k \in \{i, \dots, a-1\}$ and all $s, t \in T(\mathcal{F}_i)$, if $s \rightarrow_{\mathcal{R}_{k+1}} t$ then $s \xrightarrow{\mathcal{R}_k} t$.*

Proof. Let $s \rightarrow_{\mathcal{R}_{k+1}} t$. By the definition of the Addition rule, we have $s \rightarrow_{\mathcal{R}_k} t$ or $s \rightarrow_{\{l \rightarrow r\}} t$ where $l \rightarrow r$ is the rewrite rule added in the step from $\mathcal{R}_k$ to $\mathcal{R}_{k+1}$. If $s \rightarrow_{\mathcal{R}_k} t$ then the claim clearly holds. So suppose $s \rightarrow_{\{l \rightarrow r\}} t$. Then there exist a context $C[\]$ and a ground substitution θ_g such that $s \equiv C[l\theta_g]$ and $t \equiv C[r\theta_g]$. Now, to prove the lemma, it suffices to show that $l\theta_g \xrightarrow{\mathcal{R}_k} r\theta_g$. By the definition of the Addition rule, we have $l \xrightarrow{\mathcal{R}_k \cup \mathcal{E}} r$. Hence we have $l\theta_g \xrightarrow{\mathcal{R}_k \cup \mathcal{E}} r\theta_g$. Now, consider applications of $\leftrightarrow_{\mathcal{E}}$ in the sequence $l\theta_g \xrightarrow{\mathcal{R}_k \cup \mathcal{E}} r\theta_g$, and our aim is to show that each such application $s' \leftrightarrow_{\mathcal{E}} t'$ can be replaced by $s' \xrightarrow{\mathcal{R}_k} t'$.

Suppose one such application is of the form $C'[l'\theta'_g] \leftrightarrow_{\mathcal{E}} C'[r'\theta'_g]$ where $l' \doteq r' \in \mathcal{E}$, $C'[\]$ is some context, and θ'_g is some ground substitution such that $\theta'_g(x) \in T(\mathcal{F}_i)$ for all $x \in \text{dom}(\theta'_g)$. Here by Corollary 4.4(2), for each $x \in \text{dom}(\theta'_g)$, there exists $u \in T(\mathcal{F}_0)$ such that $\theta'_g(x) \xrightarrow{\mathcal{R}_i} u$. Hence we can find a ground substitution θ''_g such that $\theta'_g(x) \xrightarrow{\mathcal{R}_i} \theta''_g(x) \in T(\mathcal{F}_0)$ for all $x \in \text{dom}(\theta'_g) = \text{dom}(\theta''_g)$. From this we have $C'[l'\theta'_g] \xrightarrow{\mathcal{R}_i} C'[l'\theta''_g]$ and $C'[r'\theta'_g] \xrightarrow{\mathcal{R}_i} C'[r'\theta''_g]$. By assumption, we have $\mathcal{R}_0 \models_{\text{ind}} \mathcal{E}$, and hence $l'\theta''_g \xrightarrow{\mathcal{R}_0} r'\theta''_g$. Since $\mathcal{R}_0 \subseteq \mathcal{R}_i \subseteq \mathcal{R}_k$, we have $l'\theta''_g \xrightarrow{\mathcal{R}_k} r'\theta''_g$, and thus $C'[l'\theta'_g] \xrightarrow{\mathcal{R}_k} C'[l'\theta''_g] \xrightarrow{\mathcal{R}_k} C'[r'\theta''_g] \xrightarrow{\mathcal{R}_k} C'[r'\theta'_g]$. $\square$

Applying the above lemma repeatedly, we have the following.

Corollary 4.6. *Suppose that there exists an equivalence transformation sequence $\mathcal{R}_0 \xRightarrow{I} \mathcal{R}_i \xRightarrow{A} \mathcal{R}_a$ under $\mathcal{E}$ where $\mathcal{R}_0$ and $\mathcal{E}$ are over $\mathcal{F}_0$, and $\mathcal{R}_i$ and $\mathcal{R}_a$ are over $\mathcal{F}_i$. Suppose also that $\mathcal{R}_0 \models_{\text{ind}} \mathcal{E}$. Then, for all $s, t \in T(\mathcal{F}_i)$, if $s \rightarrow_{\mathcal{R}_a} t$ then $s \xrightarrow{\mathcal{R}_i} t$.*

Now we prove the correctness of the equivalence transformation under some conditions. We first give the correctness that states that each individual ground term is evaluated to the same ground constructor term under the TRSs before and after the transformation.

Theorem 4.7. *Let $\mathcal{R}$ be a left-linear constructor TRS, and suppose that there exists an equivalence transformation from $\mathcal{R}$ to $\mathcal{R}'$ under $\mathcal{E}$ where $\mathcal{R}$ and $\mathcal{E}$ are over $\mathcal{F} (= \mathcal{D} \uplus \mathcal{C})$, and $\mathcal{R}'$ is over $\mathcal{F}' (= \mathcal{D}' \uplus \mathcal{C})$. If $\mathcal{R} \models_{ind} \mathcal{E}$ and $CR(\mathcal{R})$, then for every ground term $t \in T(\mathcal{F} \cap \mathcal{F}')$ that is sufficiently complete under both $\mathcal{R}$ and $\mathcal{R}'$, and for every ground constructor term $v \in T(\mathcal{C})$, $t \xrightarrow{*}_{\mathcal{R}} v$ if and only if $t \xrightarrow{*}_{\mathcal{R}'} v$.*

Proof. Suppose the equivalence transformation sequence from $\mathcal{R}$ to $\mathcal{R}'$ is like this: $\mathcal{R} = \mathcal{R}_0 \xrightarrow{I} \mathcal{R}_1 \xrightarrow{A} \mathcal{R}_a \xrightarrow{E} \mathcal{R}'$. Let $\mathcal{R} \models_{ind} \mathcal{E}$ and $CR(\mathcal{R})$. Let $t \in T(\mathcal{F} \cap \mathcal{F}')$ be a ground term that is sufficiently complete under both $\mathcal{R}$ and $\mathcal{R}'$, and let $v \in T(\mathcal{C})$.

First we show the if part. Suppose $t \xrightarrow{*}_{\mathcal{R}'} v$. By the definition of the Elimination rule, we have $\mathcal{R}' \subseteq \mathcal{R}_a$, and so $t \xrightarrow{*}_{\mathcal{R}_a} v$. Then by Corollary 4.6, we have $t \xrightarrow{*}_{\mathcal{R}_i} v$. On the other hand, t is sufficiently complete under $\mathcal{R}$, so there exists $v' \in T(\mathcal{C})$ such that $t \xrightarrow{*}_{\mathcal{R}} v'$. Since $\mathcal{R} \subseteq \mathcal{R}_i$, we have $t \xrightarrow{*}_{\mathcal{R}_i} v'$. Now we have $v \xrightarrow{*}_{\mathcal{R}_i} t \xrightarrow{*}_{\mathcal{R}_i} v'$, which implies $v \equiv v'$, since we have $CR(\mathcal{R}_i)$ by $CR(\mathcal{R})$ and Corollary 4.4(3). Thus we obtain $t \xrightarrow{*}_{\mathcal{R}} v$.

For the only if part, suppose $t \xrightarrow{*}_{\mathcal{R}} v$. Since t is sufficiently complete under $\mathcal{R}'$, there exists $v' \in T(\mathcal{C})$ such that $t \xrightarrow{*}_{\mathcal{R}'} v'$. Then we have $t \xrightarrow{*}_{\mathcal{R}} v'$ by the if part, which we have shown above. Hence we have $v \equiv v'$ by $CR(\mathcal{R})$. Thus we obtain $t \xrightarrow{*}_{\mathcal{R}'} v$. $\square$

From the above theorem, we obtain the following corollary which states that each ground term in any subset that is sufficiently complete under the TRSs is evaluated to the same ground constructor term before and after the transformation.

Corollary 4.8. *Let $\mathcal{R}$ be a left-linear constructor TRS, and suppose that there exists an equivalence transformation from $\mathcal{R}$ to $\mathcal{R}'$ under $\mathcal{E}$ where $\mathcal{R}$ and $\mathcal{E}$ are over $\mathcal{F} (= \mathcal{D} \uplus \mathcal{C})$, and $\mathcal{R}'$ is over $\mathcal{F}' (= \mathcal{D}' \uplus \mathcal{C})$. Suppose also the following conditions:*

1. Both $\mathcal{R}$ and $\mathcal{R}'$ are sufficiently complete on a set $\mathcal{A} \subseteq T(\mathcal{F} \cap \mathcal{F}')$.
2. $\mathcal{R} \models_{ind} \mathcal{E}$.
3. $CR(\mathcal{R})$.

Then, for every ground term $t \in \mathcal{A}$ and every ground constructor term $v \in T(\mathcal{C})$, $t \xrightarrow{}_{\mathcal{R}} v$ if and only if $t \xrightarrow{*}_{\mathcal{R}'} v$.*

The correctness of the equivalent transformation given in [9, Theorem 1] can be derived from a very particular case (where $\mathcal{A} = T(\mathcal{F} \cap \mathcal{F}')$) of the above corollary.

In what follows, we give examples of equivalence transformations and show the correctness by Corollary 4.8.

Example 4.9. Consider the transformation in Example 4.1. The TRS $\mathcal{R}_{sum}$ is a left-linear constructor TRS. Here we show that $\mathcal{R}_{sum}$ can be transformed into the TRS $\mathcal{R}'_{sum}$ by an equivalence transformation in Definition 4.2 under the following set of equations:

$$\mathcal{E} \left\{ \begin{array}{l} x + (y + z) \doteq (x + y) + z \end{array} \right.$$

We have already shown that $\mathcal{R}_{sum} \models_{ind} \mathcal{E}$ in Example 3.4.

1. Let $\mathcal{R}_1 = \mathcal{R}_{sum} \cup \{\text{sum1}(x, y) \rightarrow y + \text{sum}(x)\}$. Clearly, $\mathcal{R}_{sum} \xRightarrow{I} \mathcal{R}_1$.
2. Let $\mathcal{R}_2 = \mathcal{R}_1 \cup \{\text{sum}(s(x)) \rightarrow \text{sum1}(x, s(x))\}$. Here, we have

$$\begin{array}{lcl} \text{sum}(s(x)) & \rightarrow_{\mathcal{R}_1} & s(x) + \text{sum}(x) \\ & \leftarrow_{\mathcal{R}_1} & \text{sum1}(x, s(x)) \end{array}$$

Thus, $\mathcal{R}_1 \xRightarrow{A} \mathcal{R}_2$.

3. Let $\mathcal{R}_3 = \mathcal{R}_2 \cup \{\text{sum1}(0, y) \rightarrow y\}$. Here, we have

$$\begin{array}{lcl} \text{sum1}(0, y) & \rightarrow_{\mathcal{R}_2} & y + \text{sum}(0) \\ & \rightarrow_{\mathcal{R}_2} & y + 0 \\ & \rightarrow_{\mathcal{R}_2} & y \end{array}$$

Thus, $\mathcal{R}_2 \xRightarrow{A} \mathcal{R}_3$.

4. Let $\mathcal{R}_4 = \mathcal{R}_3 \cup \{\text{sum1}(s(x), y) \rightarrow \text{sum1}(x, y + s(x))\}$. Here, we have

$$\begin{array}{lcl} \text{sum1}(s(x), y) & \rightarrow_{\mathcal{R}_3} & y + \text{sum}(s(x)) \\ & \rightarrow_{\mathcal{R}_3} & y + (s(x) + \text{sum}(x)) \\ & \leftarrow_{\mathcal{E}} & (y + s(x)) + \text{sum}(x) \\ & \leftarrow_{\mathcal{R}_3} & \text{sum1}(x, y + s(x)) \end{array}$$

Thus, $\mathcal{R}_3 \xRightarrow{A} \mathcal{R}_4$.

5. We apply the Elimination rule twice to $\mathcal{R}_4$ and obtain $\mathcal{R}_4 \xRightarrow{E} \mathcal{R}'_{sum}$.

Thus, we have $\mathcal{R}_{sum} \xRightarrow{I} \mathcal{R}_1 \xRightarrow{A} \mathcal{R}_2 \xRightarrow{A} \mathcal{R}_3 \xRightarrow{A} \mathcal{R}_4 \xRightarrow{E} \mathcal{R}'_{sum}$, which is an equivalence transformation from $\mathcal{R}_{sum}$ to $\mathcal{R}'_{sum}$ under $\mathcal{E}$.

Since $\mathcal{R}_{sum}$ is an orthogonal TRS, we have $CR(\mathcal{R}_{sum})$ by a well-known result [20]. Also, $\mathcal{R}_{sum}$ is sufficiently complete on $T(\mathcal{F}) (= T(\mathcal{F} \cap \mathcal{F}'))$ as seen in Example 3.4, and similarly $\mathcal{R}'_{sum}$ is sufficiently complete on $T(\mathcal{F}')$ since it is strongly normalising and quasi-reducible. Thus we obtain the correctness of the equivalence transformation in the style of Corollary 4.8 where $\mathcal{A} = T(\mathcal{F})$.

The next example demonstrates that Corollary 4.8 is effective in showing the correctness of equivalence transformation of TRSs that are sufficiently complete not on the set of all terms but on the set of terms of a particular sort.

Example 4.10. Consider the TRS $\mathcal{R}_{sumstr}$ in Example 3.6. It is a left-linear constructor TRS. Here we show that $\mathcal{R}_{sumstr}$

can be transformed into the following TRS $\mathcal{R}'_{sumstr}$:

$$\mathcal{R}'_{sumstr} \left\{ \begin{array}{ll} \text{sumstr}(n, a) & \rightarrow \text{sumstr1}(n, a, 0) \\ \text{sumstr1}(0, a, y) & \rightarrow y \\ \text{sumstr1}(s(n), a, y) & \rightarrow \text{sumstr1}(n, \text{tl}(a), \text{hd}(a) + y) \\ \text{hd}(x : a) & \rightarrow x \\ \text{tl}(x : a) & \rightarrow a \\ \text{inc} & \rightarrow 0 : \text{succ}(\text{inc}) \\ \text{succ}(x : a) & \rightarrow s(x) : \text{succ}(a) \\ 0 + y & \rightarrow y \\ s(x) + y & \rightarrow s(x + y) \end{array} \right.$$

by an equivalence transformation in Definition 4.2 under the following set of equations:

$$\mathcal{E} \left\{ \begin{array}{ll} x + y & \doteq y + x \\ (x + y) + z & \doteq x + (y + z) \end{array} \right.$$

We have already shown that $\mathcal{R}_{sumstr} \models_{ind} \mathcal{E}$ in Example 3.6.

1. Let $\mathcal{R}_1 = \mathcal{R}_{sumstr} \cup \{\text{sumstr1}(n, a, y) \rightarrow y + \text{sumstr}(n, a)\}$. Clearly, $\mathcal{R}_{sumstr} \xrightarrow{I} \mathcal{R}_1$.
2. Let $\mathcal{R}_2 = \mathcal{R}_1 \cup \{\text{sumstr}(n, a) \rightarrow \text{sumstr1}(n, a, 0)\}$. Here, we have

$$\begin{array}{ll} \text{sumstr}(n, a) & \leftarrow_{\mathcal{R}_1} 0 + \text{sumstr}(n, a) \\ & \leftarrow_{\mathcal{R}_1} \text{sumstr1}(n, a, 0) \end{array}$$

Thus, $\mathcal{R}_1 \xrightarrow{A} \mathcal{R}_2$.

3. Let $\mathcal{R}_3 = \mathcal{R}_2 \cup \{\text{sumstr1}(0, a, y) \rightarrow y\}$. Here, we have

$$\begin{array}{ll} \text{sumstr1}(0, a, y) & \xrightarrow{\mathcal{R}_2} y + \text{sumstr}(0, a) \\ & \xrightarrow{\mathcal{R}_2} y + 0 \\ & \leftrightarrow_{\mathcal{E}} 0 + y \\ & \xrightarrow{\mathcal{R}_2} y \end{array}$$

Thus, $\mathcal{R}_2 \xrightarrow{A} \mathcal{R}_3$.

4. Let $\mathcal{R}_4 = \mathcal{R}_3 \cup \{\text{sumstr1}(s(n), a, y) \rightarrow \text{sumstr1}(n, \text{tl}(a), \text{hd}(a) + y)\}$.

Here, we have

$$\begin{array}{ll} \text{sumstr1}(s(n), a, y) & \\ \xrightarrow{\mathcal{R}_3} & y + \text{sumstr}(s(n), a) \\ \xrightarrow{\mathcal{R}_3} & y + (\text{hd}(a) + \text{sumstr}(n, \text{tl}(a))) \\ \leftrightarrow_{\mathcal{E}} & (y + \text{hd}(a)) + \text{sumstr}(n, \text{tl}(a)) \\ \leftrightarrow_{\mathcal{E}} & (\text{hd}(a) + y) + \text{sumstr}(n, \text{tl}(a)) \\ \xrightarrow{\mathcal{R}_3} & \text{sumstr1}(n, \text{tl}(a), \text{hd}(a) + y) \end{array}$$

Thus, $\mathcal{R}_3 \xrightarrow{A} \mathcal{R}_4$.

5. We apply the Elimination rule three times to $\mathcal{R}_4$ and obtain $\mathcal{R}_4 \xrightarrow{E} \mathcal{R}'_{sumstr}$.

Thus, we have $\mathcal{R}_{sumstr} \xrightarrow{I} \mathcal{R}_1 \xrightarrow{A} \mathcal{R}_2 \xrightarrow{A} \mathcal{R}_3 \xrightarrow{A} \mathcal{R}_4 \xrightarrow{E} \mathcal{R}'_{sumstr}$, which is an equivalence transformation from $\mathcal{R}_{sumstr}$ to $\mathcal{R}'_{sumstr}$ under $\mathcal{E}$.

Since $\mathcal{R}_{sumstr}$ is an orthogonal TRS, we have $\text{CR}(\mathcal{R}_{sumstr})$. Also, both $\mathcal{R}_{sumstr}$ and $\mathcal{R}'_{sumstr}$ are sufficiently complete on $\text{T}^{\text{Nat}}(\mathcal{F}) (\subseteq \text{T}(\mathcal{F} \cap \mathcal{F}'))$ as shown in the next section (see Examples 5.9 and 5.10). Hence we obtain the correctness of

the equivalence transformation in the style of Corollary 4.8 where $\mathcal{A} = \text{T}^{\text{Nat}}(\mathcal{F})$.

5 Proving Local Sufficient Completeness

In this section, we present a method for proving local sufficient completeness for (possibly non-terminating) TRSs. We will introduce a derivation system, which act on a set of *guarded terms* (Definition 5.3). Each guarded term is accompanied with a set of *annotated terms*, which, together with a partial order and the notion of *skeleton*, are used to check availability of induction hypothesis in derivation rules.

First we introduce some notations used in this section. A (strict) *partial order* is a binary relation that is irreflexive and transitive. A partial order $>$ on terms is *stable* if it is closed under substitution, i.e. $s > t$ implies $s\theta > t\theta$ for every substitution θ ; it is *well-founded* if there exists no infinite descending chain $t_0 > t_1 > \dots$; it has *the subterm property* if $t > u$ holds for every term t and its proper subterm u . Another ordering on terms used in our derivation system is the *subsumption ordering* $\leq$, which is given by: $s \leq t$ iff $s\theta = t$ for some substitution θ .

Next we introduce notions of annotation and skeleton which are heavily used in our derivation system.

Definition 5.1 (Annotated term, Skeleton).

1. Let $\underline{\mathcal{F}} = \{f : \alpha_1 \times \dots \times \alpha_n \rightarrow \alpha \mid f : \alpha_1 \times \dots \times \alpha_n \rightarrow \alpha \in \underline{\mathcal{F}}\}$ be a set of new function symbols; more precisely, for each $f \in \mathcal{F}$, prepare a new function symbol $\underline{f}$ having the same sort declaration as f . The underline of f is called the *annotation* for f . The set $\text{AT}^\alpha(\mathcal{F} \cup \underline{\mathcal{F}}, \mathcal{V})$ of *annotated terms* of sort α is defined inductively as follows:
 - a. If $t \in \text{T}^\alpha(\mathcal{F}, \mathcal{V})$ then $t \in \text{AT}^\alpha(\mathcal{F} \cup \underline{\mathcal{F}}, \mathcal{V})$;
 - b. If $\underline{f} \in \underline{\mathcal{F}}$ is equipped with its sort declaration $\underline{f} : \alpha_1 \times \dots \times \alpha_n \rightarrow \alpha$ and $t_i \in \text{AT}^{\alpha_i}(\mathcal{F} \cup \underline{\mathcal{F}}, \mathcal{V})$ for all $i \in \{1, \dots, n\}$, then $\underline{f}(t_1, \dots, t_n) \in \text{AT}^\alpha(\mathcal{F} \cup \underline{\mathcal{F}}, \mathcal{V})$. We set $\text{AT}(\mathcal{F} \cup \underline{\mathcal{F}}, \mathcal{V}) = \bigcup_{\alpha \in \mathcal{S}} \text{AT}^\alpha(\mathcal{F} \cup \underline{\mathcal{F}}, \mathcal{V})$.
2. We denote by $\underline{t}$ the annotated term obtained from $t \in \text{T}(\mathcal{F}, \mathcal{V})$ by putting annotation for each function symbol occurrence in t .
3. For each annotated term $t \in \text{AT}(\mathcal{F} \cup \underline{\mathcal{F}}, \mathcal{V})$, its *skeleton* $\text{skel}(t) \in \text{T}(\mathcal{F}, \mathcal{V})$ is defined as follows: when $t \in \text{T}^\alpha(\mathcal{F}, \mathcal{V})$, $\text{skel}(t) = x (\in \text{T}^\alpha(\mathcal{F}, \mathcal{V}))$; otherwise, $\text{skel}(\underline{f}(t_1, \dots, t_n)) = \underline{f}(\text{skel}(t_1), \dots, \text{skel}(t_n))$. Here, we assume that a fresh variable x is taken for each time when a variable is required.

Clearly, we have $\underline{t} \in \text{T}(\underline{\mathcal{F}}, \mathcal{V})$ for each term t , and skeletons of annotated terms are linear terms without any annotation.

Example 5.2. Let $\mathcal{F} = \{0, s, +\}$. Then $\underline{\mathcal{F}} = \{0, \underline{s}, \underline{+}\}$. Then, $t = s(x) \underline{+} 0$ is an annotated term. Its skeleton $\text{skel}(t) = x_1 + x_2$. For $t' = s(x) + 0$, we have $\underline{t'} = \underline{s}(x) \underline{+} 0$.

<i>Start</i>	<i>Abstract</i>
$\frac{\Gamma \cup \{t H\}}{\Gamma \cup \{t (\{t\} \cup H)\}}$	$\frac{\Gamma \cup \{(C[t]) H\}}{\Gamma \cup \{(C[x]) (\{\tilde{h}_t(x)\} \cup H)\}} \quad x : \text{fresh}, \exists u \in H. \text{skel}(u) \leq t < u$
<i>Expand</i>	
$\frac{\Gamma \cup \{t H\}}{\Gamma \cup \{t\sigma_i H\sigma_i\}_i} \quad x^\alpha \in \mathcal{V}(t), \vec{x} : \text{a sequence of fresh variables}$	$\{\sigma_i\}_i = \{\{x \mapsto f(\vec{x})^\alpha\} \mid f \in \mathcal{F}, f \in \mathcal{D} \Rightarrow \forall u^\alpha \in H \setminus \mathcal{V}. \text{skel}(u) \not\leq f(\vec{x}) \vee x \notin \mathcal{V}(u)\}$
<i>Simplify</i>	<i>Decompose</i>
$\frac{\Gamma \cup \{t H\}}{\Gamma \cup \{s H\}} \quad t \rightarrow_{\mathcal{R}} s$	$\frac{\Gamma \cup \{f(t_1, \dots, t_n) H\}}{\Gamma \cup \{t_1 H, \dots, t_n H\}} \quad f \in C$
<i>Lemma</i>	<i>Delete</i>
$\frac{\Gamma \cup \{t H\}}{\Gamma \cup \{t (\{s\} \cup H), s H\}}$	$\frac{\Gamma \cup \{t H\}}{\Gamma} \quad \exists u \in H. \text{skel}(u) \leq t < u$

Figure 3. Derivation rules for proving local sufficient completeness

Our derivation system acts on a set of guarded terms, defined as follows:

Definition 5.3 (Guarded term). A *guarded term*, denoted by $t|H$, consists of a linear term t and a set H of linear annotated terms.

Now we are ready to present the notion of derivation of our system for proving local sufficient completeness.

Definition 5.4 (Derivation for proving local sufficient completeness). Let $\mathcal{R}$ be a TRS, $>$ a well-founded stable partial order with the subterm property over the set $T(\mathcal{F}, \mathcal{V})$.

1. The derivation rules of our system are listed in Figure 3. It derives from a set of guarded terms (given at upper side) to a set of guarded terms (given at lower side) if the side condition is satisfied.
2. The comparison by $>$ for terms involving annotations is performed after erasing all annotations.
3. We use new unary function symbols $\tilde{h}_t$ for each linear term t ; we extend the order $>$ as $\tilde{h}_t(s) > u$ iff $s > u$ or $s = u$, and the notion of skeleton as $\text{skel}(\tilde{h}_t(u)) = t$.
4. For sets Γ, Γ' of guarded terms, we write $\Gamma \rightsquigarrow \Gamma'$ if Γ' is derived from Γ by some derivation rule. The reflexive transitive closure of $\rightsquigarrow$ is written as $\rightsquigarrow^*$.

Lemma 5.5. Suppose $\Gamma \rightsquigarrow^* \emptyset$. Then, for every guarded term $t|H \in \Gamma$ and every ground substitution σ_g , $\text{SC}(t\sigma_g)$ holds if the following condition holds:

$$\forall u \in H. \forall w_g \in T(\mathcal{F}). \text{skel}(u) \leq w_g < u\sigma_g \Rightarrow \text{SC}(w_g)$$

Proof. We show the claim by induction on the length of $\Gamma \rightsquigarrow^* \emptyset$. The base step is trivial. Suppose $\Gamma \rightsquigarrow \Gamma' \rightsquigarrow^* \emptyset$. Note by the induction hypothesis, $\forall u \in H'. \forall w_g \in T(\mathcal{F}). \text{skel}(u) \leq w_g < u\sigma_g \Rightarrow \text{SC}(w_g)$ implies $\text{SC}(t'\sigma_g)$ for every $t'|H' \in \Gamma'$ and σ_g . We divide cases by the rule applied in the step $\Gamma \rightsquigarrow \Gamma'$.

(*Start*) Then $\Gamma = \Sigma \cup \{t'|H'\}$ and $\Gamma' = \Sigma \cup \{t'|(\{t'\} \cup H')\}$. The case $t|H \in \Sigma$ follows directly from the induction hypothesis. Thus, it remains to show the case $t|H = t'|H'$. We show $\text{SC}(t\sigma_g)$ for every ground substitution σ_g . Suppose $\text{SC}(t\sigma_g)$ does not hold for some σ_g . Then, because $>$ is well-founded, there is such $t\sigma_g$ minimal w.r.t. $>$, i.e. $\text{SC}(t\sigma_g)$ does not hold and $\text{SC}(t\rho_g)$ for every $t\rho_g$ such that $t\rho_g < t\sigma_g$. Suppose w_g satisfies $\text{skel}(t) \leq w_g < t\sigma_g$. By $\text{skel}(t) \leq w_g$, we know $w_g = t\rho_g$ for some ρ_g , and hence it follows $\text{SC}(w_g)$. Thus, if there exists $u \in \{t\} \cup H$ such that $\text{skel}(u) \leq w_g < u\sigma_g$, then $\text{SC}(w_g)$. Hence, from the induction hypothesis, $\text{SC}(t\sigma_g)$ should follow. This contradicts our assumption that $\text{SC}(t\sigma_g)$ does not hold. All in all, we conclude $\text{SC}(t\sigma_g)$ for every σ_g .

(*Abstract*) Then $\Gamma = \Sigma \cup \{C[t']|H'\}$ and $\Gamma' = \Sigma \cup \{C[x]|(\{\tilde{h}_{t'}(x)\} \cup H')\}$, where x is a fresh variable and $\text{skel}(u) \leq t' < u$ for some $u \in H'$. The case $t|H \in \Sigma$ follows directly from the induction hypothesis. Thus, it remains to show the case $t|H = C[t']|H'$. Let σ_g be a ground substitution such that $(\alpha) : \forall u \in H. \forall w_g \in T(\mathcal{F}). \text{skel}(u) \leq w_g < u\sigma_g \Rightarrow \text{SC}(w_g)$. We have to show $\text{SC}(C[t']\sigma_g)$. We know from the induction hypothesis that, for every σ'_g , if $\forall u \in (\{\tilde{h}_t(x)\} \cup H). \forall w_g \in T(\mathcal{F}). \text{skel}(u) \leq w_g < u\sigma'_g \Rightarrow \text{SC}(w_g)$, then $\text{SC}(C[x]\sigma'_g)$. Take $\sigma'_g = \sigma_g \cup \{x \mapsto t\sigma_g\}$. Then $C[x]\sigma'_g = C[t]\sigma_g$, and thus it remains to show $\forall u \in (\{\tilde{h}_t(x)\} \cup H). \forall w_g \in T(\mathcal{F}). \text{skel}(u) \leq w_g < u\sigma'_g \Rightarrow \text{SC}(w_g)$. Suppose $u \in H$ and take w_g such that $\text{skel}(u) \leq w_g < u\sigma'_g$. As $u \in H$, $x \notin \mathcal{V}(u)$ by our assumption that x is fresh. Thus, $u\sigma_g = u\sigma'_g$. Hence we have $\text{skel}(u) \leq w_g < u\sigma_g$, and thus $\text{SC}(w_g)$ by our assumption (α) . Otherwise, $u = \tilde{h}_t(x)$. Take w_g such that $\text{skel}(u) = t \leq w_g < u\sigma'_g$. By $w_g < u\sigma'_g = \tilde{h}_t(\sigma'_g(x))$ and our definition of the order $<$, we have $w_g \leq \sigma'_g(x) = t\sigma_g$. By the condition of the rule, there exists $u' \in H$ such that $\text{skel}(u') \leq t < u'$. From $t < u'$, we have $t\sigma_g < u'\sigma_g$, and thus $w_g \leq \sigma'_g(x) = t\sigma_g < u'\sigma_g$. We also have $\text{skel}(u') \leq t \leq w_g$. Thus $\text{skel}(u') \leq w_g < u'\sigma_g$ for

some $u' \in H$. Hence, $SC(w_g)$ follows from our assumption (α) .

(Expand) Then $\Gamma = \Sigma \cup \{t'|H'\}$ and $\Gamma' = \Sigma \cup \{t'\sigma_i|H'\sigma_i\}_i$, where $x \in \mathcal{V}(t)$, $\vec{x}$ is a sequence of fresh variables, and $\{\sigma_i\}_i = \{\{x \mapsto f(\vec{x})\} \mid f \in C \vee (\forall u \in H' \setminus \mathcal{V}. \text{skel}(u) \not\leq f(\vec{x}) \vee x \notin \mathcal{V}(u))\}$. The case $t|H \in \Sigma$ follows directly from the induction hypothesis. Thus, it remains to show the case $t|H = t'|H'$. Let σ_g be a ground substitution such that $(\beta): \forall u \in H. \forall w_g \in T(\mathcal{F}). \text{skel}(u) \leq w_g < u\sigma_g \Rightarrow SC(w_g)$. We have to show $SC(t\sigma_g)$. For this, we distinguish two cases.

1. Suppose that there exists an index i such that $t\sigma_g = (t\sigma_i)\sigma'_g$ for some σ'_g . Then, by $t\sigma_i \in H\sigma_i \in \Gamma'$, we know from the induction hypothesis that if $\forall u \in H\sigma_i. \forall w_g \in T(\mathcal{F}). \text{skel}(u) \leq w_g < u\sigma'_g \Rightarrow SC(w_g)$, then $SC((t\sigma_i)\sigma'_g)$. Since $t\sigma_g = (t\sigma_i)\sigma'_g$, it remains to show $\forall u \in H\sigma_i. \forall w_g \in T(\mathcal{F}). \text{skel}(u) \leq w_g < u\sigma'_g \Rightarrow SC(w_g)$. Suppose $u \in H\sigma_i$ and $\text{skel}(u) \leq w_g < u\sigma'_g$. Then, there exists $\hat{u} \in H$ such that $u = \hat{u}\sigma_i$, and we have $\text{skel}(\hat{u}) = \text{skel}(u) \leq w_g < u\sigma'_g = (\hat{u}\sigma_i)\sigma'_g = \hat{u}\sigma'_g$. Thus, $SC(w_g)$ follows from our assumption (β) .
2. Otherwise. Then we have $t\sigma_g = (t\theta)\sigma'_g$ for some $\theta = \{x \mapsto f(\vec{x})\}$ for some $f \in \mathcal{D}$ such that $\text{skel}(u) \leq f(\vec{x}) \wedge x \in \mathcal{V}(u)$ for some $u \in H \setminus \mathcal{V}$. Then we have $\text{skel}(u) \leq f(\vec{x}) \leq (x\theta)\sigma'_g = x\sigma_g$. Furthermore, by $x \in \mathcal{V}(u)$ and $u \notin \mathcal{V}$, we have $x\sigma_g < u\sigma_g$ by the subterm property of $>$. Hence, $\text{skel}(u) \leq x\sigma_g < u\sigma_g$. Thus, by our assumption (β) , we obtain $SC(x\sigma_g)$. Then by the definition, we know there exists a constructor ground term $h(\vec{w}_g)$ such that $x\sigma_g \xrightarrow{*}_{\mathcal{R}} h(\vec{w}_g)$. From $h \in C$, there is an index i such that $\sigma_i = \{x \mapsto h(\vec{x})\}$. As $\vec{x}$ is fresh, we may assume $x\sigma_g \xrightarrow{*}_{\mathcal{R}} (x\sigma_i)\sigma'_g$. Thus $t\sigma_g \xrightarrow{*}_{\mathcal{R}} (t\sigma_i)\sigma'_g$ for some i and σ'_g . From the previous case, we know $SC((t\sigma_i)\sigma'_g)$ and hence $SC(t\sigma_g)$ follows.

(Simplify) Then $\Gamma = \Sigma \cup \{t'|H'\}$ and $\Gamma' = \Sigma \cup \{s'|H'\}$, where $t' \rightarrow_{\mathcal{R}} s'$. The case $t|H \in \Sigma$ follows directly from the induction hypothesis. Thus, it remains to show the case $t|H = t'|H'$. Let σ_g be a ground substitution such that $\forall u \in H. \forall w_g \in T(\mathcal{F}). \text{skel}(u) \leq w_g < u\sigma_g \Rightarrow SC(w_g)$. Then, by induction hypothesis, we have $SC(s'\sigma_g)$. Since $t'\sigma_g \rightarrow_{\mathcal{R}} s'\sigma_g$, $SC(t'\sigma_g)$ clearly holds.

(Decompose) Then $\Gamma = \Sigma \cup \{f(t_1, \dots, t_n)|H'\}$ and $\Gamma' = \Sigma \cup \{t_1|H', \dots, t_n|H'\}$, where $f \in C$. The case $t|H \in \Sigma$ follows directly from the induction hypothesis. Thus, it remains to show the case $t|H = f(t_1, \dots, t_n)|H'$. Let σ_g be a ground substitution such that $\forall u \in H. \forall w_g \in T(\mathcal{F}). \text{skel}(u) \leq w_g < u\sigma_g \Rightarrow SC(w_g)$. Then, by induction hypothesis, we have $SC(t_i\sigma_g)$ for all $i = 1, \dots, n$. Thus, since $f \in C$, we have $SC(f(t_1, \dots, t_n)\sigma_g)$.

(Lemma) Then $\Gamma = \Sigma \cup \{t'|H'\}$ and $\Gamma' = \Sigma \cup \{t'|(\{s\} \cup H'), s|H'\}$, where x is a fresh variable. The case $t|H \in \Sigma$ follows directly from the induction hypothesis. Thus, it remains

to show the case $t|H = t'|H'$. Let σ_g be a ground substitution such that $\forall u \in H. \forall w_g \in T(\mathcal{F}). \text{skel}(u) \leq w_g < u\sigma_g \Rightarrow SC(w_g)$. Then, by the induction hypothesis, we have $SC(s\sigma_g)$. Note that, for every $\rho_g \leq \sigma_g$, $\text{skel}(u) \leq w_g \leq u\rho_g$ implies $\text{skel}(u) \leq w_g \leq u\sigma_g$ for all $u \in H$ and w_g . Hence, we have $SC(s\rho_g)$. Suppose $\text{skel}(s) = s \leq w_g < s\sigma_g$. Then $w_g = s\rho_g$ for some ρ_g , where one may w.l.o.g. assume $\rho_g \leq \sigma_g$. Thus, we have $SC(w_g)$. Hence, it holds that $\forall u \in \{s\} \cup H. \forall w_g \in T(\mathcal{F}). \text{skel}(u) \leq w_g < u\sigma_g \Rightarrow SC(w_g)$. Hence, we obtain $SC(t\sigma_g)$.

(Delete) Then $\Gamma = \Gamma' \cup \{t'|H'\}$, where $\text{skel}(u) \leq t' < u$ for some $u \in H'$. The case $t|H \in \Gamma'$ follows directly from the induction hypothesis. Thus, it remains to show the case $t|H = t'|H'$. Let σ_g be a ground substitution such that $(\gamma): \forall u \in H. \forall w_g \in T(\mathcal{F}). \text{skel}(u) \leq w_g < u\sigma_g \Rightarrow SC(w_g)$. By the condition of the rule, $\text{skel}(u) \leq t \wedge u < t$ for some $u \in H$. Thus, $\text{skel}(u) \leq t \leq t\sigma_g < u\sigma_g$. Hence $SC(t\sigma_g)$ by our assumption (γ) . $\square$

The next theorem immediately follows from Lemma 5.5.

Theorem 5.6. Let $\mathcal{R}$ be a TRS and t a linear term. If $\{t|\emptyset\} \xrightarrow{*} \emptyset$ for some well-founded stable order $>$ with the subterm property, then $SC(\mathcal{R}, t)$.

A well-founded stable order $>$ with the subterm property is a particular case of a *simplification order*, and many methods for constructing such an order are known, e.g. lexicographic path order, multiset path order, Knuth-Bendix order, etc. (cf. [5, 22])

In what follows, we give some examples of derivations for proving local sufficient completeness. The first two derivations illustrate the basic use of the inference rules except *Abstract* and *Lemma*.

Example 5.7. Let $\mathcal{R}_{\text{repl}}$ be the TRS given in Example 3.5. We show $SC(\mathcal{R}_{\text{repl}}, \text{replicate}(n))$ and $SC(\mathcal{R}_{\text{repl}}, \text{take}(n, \text{ones}))$. For this, take $>$ as the lexicographic path ordering based on some precedence (e.g. $\text{replicate} > \text{take} > : > s > [] > ()$). In Figure 4, we present a derivation of $\{\text{replicate}(n)|\emptyset\} \xrightarrow{*} \emptyset$. The final *Delete* step follows by

$$\begin{aligned} \text{skel}(\text{replicate}(s(n))) &= \text{replicate}(n_1) \\ &\leq \text{replicate}(n) \\ &< \text{replicate}(s(n)). \end{aligned}$$

In Figure 5, we also present a derivation of $\{\text{take}(n, \text{ones})|\emptyset\} \xrightarrow{*} \emptyset$. Thus, we conclude that $\text{replicate}(n)$ and $\text{take}(n, \text{ones})$ are sufficiently complete under $\mathcal{R}_{\text{repl}}$.

The following is a more challenging example of a derivation, which illustrates the use of all the inference rules.

Example 5.8. Consider a signature with $S = \{\text{Nat}, \text{List}\}$ and $\mathcal{F} = \{\text{prod} : \text{List} \rightarrow \text{Nat}, \text{zeros} : \text{List}, : : \text{Nat} \times \text{List} \rightarrow \text{List}, [] : \text{List}, + : \text{Nat} \times \text{Nat} \rightarrow \text{Nat}, \times : \text{Nat} \times$

$\leadsto_{Start}$	$\{ \text{replicate}(n) \emptyset \}$
$\leadsto_{Expand}$	$\{ \text{replicate}(n) \{ \text{replicate}(n) \} \}$
$\leadsto_{Simplify}^*$	$\{ \text{replicate}(0) \{ \text{replicate}(0) \}, \text{replicate}(s(n)) \{ \text{replicate}(s(n)) \} \}$
$\leadsto_{Decompose}^*$	$\{ [] \{ \text{replicate}(0) \}, (s(0) : \text{replicate}(n)) \{ \text{replicate}(s(n)) \} \}$
$\leadsto_{Delete}$	$\{ \text{replicate}(n) \{ \text{replicate}(s(n)) \} \}$
	$\}$

Figure 4. A derivation for $SC(\mathcal{R}_{repl}, \text{replicate}(n))$

$\leadsto_{Start}$	$\{ \text{take}(n, \text{ones}) \emptyset \}$
$\leadsto_{Expand}$	$\{ \text{take}(n, \text{ones}) \{ \text{take}(n, \text{ones}) \} \}$
$\leadsto_{Simplify}^*$	$\{ \text{take}(0, \text{ones}) \{ \text{take}(0, \text{ones}) \}, \text{take}(s(n), \text{ones}) \{ \text{take}(s(n), \text{ones}) \} \}$
$\leadsto_{Decompose}^*$	$\{ [] \{ \text{take}(0, \text{ones}) \}, (s(0) : \text{take}(n, \text{ones})) \{ \text{take}(s(n), \text{ones}) \} \}$
$\leadsto_{Delete}$	$\{ \text{take}(n, \text{ones}) \{ \text{take}(s(n), \text{ones}) \} \}$
	$\}$

Figure 5. A derivation for $SC(\mathcal{R}_{repl}, \text{take}(n, \text{ones}))$

$\text{Nat} \rightarrow \text{Nat}$, $s : \text{Nat} \rightarrow \text{Nat}$, $0 : \text{Nat}$. Let $\mathcal{R}_{prodlist}$ be the following TRS for the product of the elements of a list of natural numbers:

$$\mathcal{R}_{prodlist} \left\{ \begin{array}{ll} 0 + y & \rightarrow y \\ s(x) + y & \rightarrow s(x + y) \\ 0 \times y & \rightarrow 0 \\ s(x) \times y & \rightarrow (x \times y) + y \\ \text{prod}([]) & \rightarrow s(0) \\ \text{prod}(x : ys) & \rightarrow x \times \text{prod}(ys) \\ \text{zeros} & \rightarrow 0 : \text{zeros} \end{array} \right.$$

Then $SC(\mathcal{R}_{prodlist}, \text{prod}(xs))$ holds. In Figure 6, we present a derivation of $\{\text{prod}(xs) | \emptyset\} \xrightarrow{*} \emptyset$, where any lexicographic path ordering based on some precedence can be taken for $>$.

Unfortunately, our current derivation system is not sufficient for proving the statements of local sufficient completeness mentioned in Examples 3.6 and 4.10. Here we prove them without using the derivation system.

Example 5.9. Let $\mathcal{R}_{sumstr}$ be the TRS given in Example 3.6. We show $SC(\mathcal{R}_{sumstr}, x^{\text{Nat}})$, which means that $\mathcal{R}_{sumstr}$ is sufficiently complete on $T^{\text{Nat}}(\mathcal{F})$. It does not seem possible to obtain a derivation of $\{x^{\text{Nat}} | \emptyset\} \xrightarrow{*} \emptyset$. However, one can prove the following claim: For every ground term $t \in T(\mathcal{F})$, (1) if $t \in T^{\text{Nat}}(\mathcal{F})$ then there exists $u \in T^{\text{Nat}}(C)$ such that $t \xrightarrow{*} u$ and (2) if $t \in T^{\text{Stream}}(\mathcal{F})$ then for each $n \geq 0$, there exist $u_1, \dots, u_n \in T^{\text{Nat}}(C)$ and $s \in T^{\text{Stream}}(\mathcal{F})$ such that $t \xrightarrow{*} u_1 : u_2 : \dots : u_n : s$.

(Proof of the claim) The proof proceeds by induction on t .

1. Case $t = 0$. Then, the claim trivially holds by $0 \in T^{\text{Nat}}(C)$.

2. Case $t = s(t')$. Then, by induction hypothesis, $t' \xrightarrow{*} u'$ for some $u' \in T^{\text{Nat}}(C)$. Thus, the claim follows from $s(u') \in T^{\text{Nat}}(C)$.
3. Case $t = t_1 + t_2$. Then, by induction hypothesis, $t_i \xrightarrow{*} u_i$ for some $u_i \in T^{\text{Nat}}(C)$ ($i \in \{1, 2\}$). Then, $u_1 = s^{n_1}(0)$ and $u_2 = s^{n_2}(0)$ for some $n_1, n_2 \geq 0$. Thus, $t = t_1 + t_2 \xrightarrow{*} s^{n_1}(0) + s^{n_2}(0) \xrightarrow{*} s^{n_1+n_2}(0) \in T^{\text{Nat}}(C)$.
4. Case $t = \text{inc}$. The claim follows, because we have $t \xrightarrow{*} 0 : s(0) : \dots : s^n(0) : \text{succ}^{n+1}(\text{inc})$, for each $n \geq 0$.
5. Case $t = \text{tl}(t')$. Then, by induction hypothesis, we have $t' \xrightarrow{*} u_1 : u_2 : \dots : u_{n+1} : s$ for some $u_1, \dots, u_{n+1} \in T^{\text{Nat}}(C)$ and $s \in T^{\text{Stream}}(\mathcal{F})$. Then, we obtain $\text{tl}(t') \xrightarrow{*} \text{tl}(u_1 : u_2 : \dots : u_{n+1} : s) \rightarrow u_2 : \dots : u_{n+1} : s$, and thus, the claim follows.
6. Case $t = \text{succ}(t')$. By induction hypothesis, we have $t' \xrightarrow{*} u_1 : \dots : u_n : s$ for some $u_1, \dots, u_n \in T^{\text{Nat}}(C)$ and $s \in T^{\text{Stream}}(\mathcal{F})$. Thus, $\text{succ}(t') \xrightarrow{*} \text{succ}(u_1 : \dots : u_n : s) \rightarrow s(u_1) : \dots : s(u_n) : \text{succ}(s)$, and hence, the claim follows.
7. Case $t = t_1 : t_2$. By induction hypothesis, we have $t_1 \xrightarrow{*} u_1$ for some $u_1 \in T^{\text{Nat}}(C)$, and $t_2 \xrightarrow{*} u_2 : \dots : u_{n+1} : s$ for some $u_2, \dots, u_{n+1} \in T^{\text{Nat}}(C)$ and $s \in T^{\text{Stream}}(\mathcal{F})$. Then $t_1 : t_2 \xrightarrow{*} u_1 : u_2 : \dots : u_{n+1} : s$, and thus, the claim follows.
8. Case $t = \text{hd}(t')$. Then, by induction hypothesis, $t' \xrightarrow{*} u : s$ for some $u \in T^{\text{Nat}}(C)$ and $s \in T^{\text{Stream}}(\mathcal{F})$. Then $\text{hd}(t') \xrightarrow{*} \text{hd}(u : s) \rightarrow u$, and thus, the claim follows.
9. Case $t = \text{sumstr}(t_1, t_2)$. Then, by induction hypothesis, we have $t_1 \xrightarrow{*} s^n(0)$ for some $n \geq 0$. Again by

Figure 6. A derivation for $\text{SC}(\mathcal{R}_{\text{prodlist}}, \text{prod}(xs))$

induction hypothesis, $t_2 \xrightarrow{*} u_1 : \dots : u_n : s$ where $u_i = s^{k_i}(0)$ for each $1 \leq i \leq n$, and $s \in T^{\text{Stream}}(\mathcal{F})$. Then $\text{sumstr}(t_1, t_2) \xrightarrow{*} \text{sumstr}(s^n(0), u_1 : \dots : u_n : s) \xrightarrow{*} u_1 + (u_2 + (\dots + (u_n + 0) \dots)) \xrightarrow{*} s^k(0)$, where $k = \sum_{1 \leq i \leq n} k_i$. Thus, the claim follows.

(End of the proof of the claim)

From the part (1) of the claim, we conclude $\text{SC}(\mathcal{R}_{\text{sumstr}}, x^{\text{Nat}})$.

Example 5.10. We now prove $\text{SC}(\mathcal{R}'_{\text{sumstr}}, x^{\text{Nat}})$ for Example 4.10. The proof is very similar to the one in Example 5.9, and we prove the same claim for $\mathcal{R}'_{\text{sumstr}}$. The proofs of cases other than $t = \text{sumstr}(t_1, t_2)$ and $t = \text{sumstr}(t_1, t_2, t_3)$ are the same. Thus, we consider only these two cases.

1. Case $t = \text{sumstr}(t_1, t_2)$. Then, by induction hypothesis, we have $t_1 \xrightarrow{*} s^n(0)$ for some $n \geq 0$. Again by induction hypothesis, $t_2 \xrightarrow{*} u_1 : \dots : u_n : s$ where $u_i = s^{k_i}(0)$ for each $1 \leq i \leq n$, and $s \in T^{\text{Stream}}(\mathcal{F})$. Then, we obtain $\text{sumstr}(t_1, t_2) \xrightarrow{*} \text{sumstr}(s^n(0), u_1 : \dots : u_n : s) \rightarrow$

$\text{sumstr1}(s^n(0), u_1 : \dots : u_n : s, 0) \xrightarrow{*} u_n + (u_{n-1} + (\dots + (u_1 + 0) \dots)) \xrightarrow{*} s^k(0)$, where $k = \sum_{1 \leq i \leq n} k_i$. Thus, the claim follows.

2. Case $t = \text{sumstr}(t_1, t_2, t_3)$. Then, by induction hypothesis, we have $t_1 \xrightarrow{*} s^n(0)$ and $t_3 \xrightarrow{*} s^m(0)$ for some $n, m \geq 0$. Again by induction hypothesis, $t_2 \xrightarrow{*} u_1 : \dots : u_n : s$ where $u_i = s^{k_i}(0)$ for each $1 \leq i \leq n$, and $s \in T^{\text{Stream}}(\mathcal{F})$. Then, we have $\text{sumstr}(t_1, t_2, t_3) \xrightarrow{*} \text{sumstr1}(s^n(0), u_1 : \dots : u_n : s, s^m(0)) \xrightarrow{*} u_n + (u_{n-1} + (\dots + (u_1 + s^m(0)) \dots)) \xrightarrow{*} s^k(0)$, where $k = (\sum_{1 \leq i \leq n} k_i) + m$. Thus, the claim follows.

Thus, we conclude $\text{SC}(\mathcal{R}'_{\text{sumstr}}, x^{\text{Nat}})$.

6 Conclusion

We have presented a framework for proving inductive theorems of first-order equational theories, using techniques from the field of term rewriting. We have introduced the

notion of local sufficient completeness, which is useful in proving inductive theorems of TRSs that are not necessarily sufficiently complete for all terms, as well as in proving the correctness of equivalence transformation of such TRSs. We have also presented a derivation system to check the condition of local sufficient completeness and proved the correctness of the system. To check another condition in our method for proving inductive theorems, we have made use of an automated confluence prover.

We are planning to make experiments to examine to what extent our method works. In doing so, we are to cope with cases where confluence does not hold but the equation in question is still an inductive theorem. If only terminating TRSs were treated, completion could be used in the same way as traditional inductionless induction methods do. However, in the cases where non-terminating TRSs are involved, some new investigations are necessary.

The equivalent transformation in [9] is extended to that with templates where all TRSs that match those templates can be transformed. In the present paper, we have not used such templates but transformed individual TRSs one by one. Incorporating our framework into such transformation with templates is left as future work. Another direction of future work is to provide a framework of proving inductive properties of programs in languages with polymorphic types and higher-order functions.

Acknowledgments

We are grateful to the anonymous referees for valuable comments. The first author thanks the Japan Society for the Promotion of Science (JSPS), Core-to-Core Program (A. Advanced Research Networks) for supporting this research. The research was also supported by JSPS KAKENHI Grant Numbers 17K00005, 18K11158 and 19K11891.

References

- [1] T. Aoto. 2008. Sound lemma generation for proving inductive validity of equations. In *Proc. of 28th FSTTCS (LIPIcs)*, Vol. 2. Schloss Dagstuhl, 13–24.
- [2] T. Aoto, M. Hamana, N. Hirokawa, A. Middeldorp, J. Nagele, N. Nishida, K. Shintani, and H. Zankl. 2018. Confluence competition 2018. In *Proc. of 3rd FSCD (LIPIcs)*, Vol. 108. Schloss Dagstuhl, 32:1–32:5.
- [3] T. Aoto and S. Stratulat. 2014. Decision procedures for proving inductive theorems without induction. In *Proc. of 16th PPDP*. ACM Press, 237–248.
- [4] T. Aoto, J. Yoshida, and Y. Toyama. 2009. Proving confluence of term rewriting system automatically. In *Proc. of 20th RTA (LNCS)*, Vol. 5595. Springer-Verlag, 93–102.
- [5] F. Baader and T. Nipkow. 1998. *Term Rewriting and All That*. Cambridge University Press.
- [6] A. Bouhoula. 2009. Simultaneous checking of completeness and ground confluence for algebraic specifications. *ACM Transactions on Computational Logic* 10, 2 (2009), 20:1–33.
- [7] A. Bouhoula, E. Kounalis, and M. Rusinowitch. 1995. Automated mathematical induction. *Journal of Logic and Computation* 5, 5 (1995), 631–668.
- [8] R. M. Burstall and J. Darlington. 1977. A transformation system for developing recursive programs. *Journal of the Association for Computing Machinery* 24, 1 (1977), 44–67.
- [9] Y. Chiba, T. Aoto, and Y. Toyama. 2005. Program transformation by templates based on term rewriting. In *Proc. of 7th PPDP*. ACM Press, 59–69.
- [10] H. Comon and F. Jacquemard. 2003. Ground reducibility is EXPTIME-complete. *Information and Computation* 187 (2003), 123–153.
- [11] S. Falke and D. Kapur. 2012. Rewriting induction + linear arithmetic = decision procedure. In *Proc. of 6th IJCAR (LNAI)*, Vol. 7364. 241–255.
- [12] G. Huet and J.-M. Hullot. 1982. Proof by induction in equational theories with constructors. *J. Comput. System Sci.* 25, 2 (1982), 239–266.
- [13] J.-P. Jouannaud and E. Kounalis. 1989. Automatic proofs by induction in theories without constructors. *Information and Computation* 82 (1989), 1–33.
- [14] D. Kapur, P. Narendran, and H. Zhang. 1987. On sufficient-completeness and related properties of term rewriting systems. *Acta Informatica* 24, 4 (1987), 395–415.
- [15] D. Kapur, P. Narendran, and H. Zhang. 1991. Automating inductionless induction using test sets. *Journal of Symbolic Computation* 11, 1–2 (1991), 81–111.
- [16] A. Lazrek, P. Lescanne, and J. J. Thiel. 1990. Tools for proving inductive equalities, relative completeness, and ω -completeness. *Information and Computation* 84 (1990), 47–70.
- [17] D. R. Musser. 1980. On proving inductive properties of abstract data types. In *Proc. of 7th POPL*. ACM Press, 154–162.
- [18] T. Nipkow and G. Weikum. 1983. A decidability result about sufficient-completeness of axiomatically specified abstract data types. In *Proc. of 6th GI-Conference on Theoretical Computer Science (LNCS)*, Vol. 145. Springer-Verlag, 257–268.
- [19] U. S. Reddy. 1990. Term rewriting induction. In *Proc. of CADE-10 (LNAI)*, Vol. 449. Springer-Verlag, 162–177.
- [20] B. Rosen. 1973. Tree-manipulating systems and Church-Rosser theorems. *Journal of the Association for Computing Machinery* 20, 1 (1973), 160–187.
- [21] S. Stratulat. 2001. A general framework to build contextual cover set induction provers. *Journal of Symbolic Computation* 32 (2001), 403–445.
- [22] Terese. 2003. *Term Rewriting Systems*. Cambridge University Press.
- [23] Y. Toyama. 1988. Commutativity of term rewriting systems. In *Programming of Future Generation Computers II*, K. Fuchi and L. Kott (Eds.). North-Holland, 393–407.
- [24] Y. Toyama. 1991. How to prove equivalence of term rewriting systems without induction. *Theoretical Computer Science* 90, 2 (1991), 369–390.