Designing a Rewriting Induction Prover with an Increased Capability of Non-Orientable Theorems

Takahito Aoto

RIEC, Tohoku University, Japan
aoto@nue.riec.tohoku.ac.jp

Abstract. Rewriting induction (Reddy, 1990) is an automated proof method for inductive theorems of term rewriting systems. Reasoning by the rewriting induction is based on the noetherian induction on some reduction order and the original rewriting induction is not capable of proving theorems which are not orientable by that reduction order. To deal with such theorems, Bouhoula (1995) as well as Dershowitz & Reddy (1993) used the ordered rewriting. However, even using ordered rewriting, the weak capability of non-orientable theorems is considered one of the weakness of rewriting induction approach compared to other automated methods for proving inductive theorems. We present a refined system of rewriting induction with an increased capability of non-orientable theorems and a capability of disproving incorrect conjectures. Soundness for proving/disproving are shown and effectiveness of our system is demonstrated through some examples.

1 Introduction

Properties of programs are often proved by induction on data structures such as natural numbers or lists. Such properties are called *inductive properties* of programs. Inductive properties are indispensable in formal treatments of programs such as program verification and program transformation. For such applications, automated reasoning on inductive properties is crucial.

Comparing to the high degree of automation on automated proving of theorems, automated proving of *inductive* theorems still is considered as a very challenging problem [15]. Currently best known successful approaches to automated proving of inductive theorems are *explicit* induction methods with sophisticated heuristics [12, 17, 23] or with powerful decision procedures [27]. On the other hand, in the field of term rewriting, *implicit* induction methods for equational theories that automatically perform inductive reasoning based on implicit induction principles have been investigated for many years [5–10, 14, 16, 18, 20, 21, 24–26, 29, 31]. Although it is not among the best known successful approaches, some extensions are known to be competitive [5, 7].

Rewriting induction[1] proposed by Reddy [26] is one of such inductive theorem proving methods. Contrasted to *inductionless induction* [16, 18, 21, 25, 31], in which some kind of Church-Rosser property is needed, the basis of rewriting induction is a noetherian induction. The theorem prover SPIKE [6, 9, 10] is the best known successful induction prover based on a variant of rewriting induction *test set induction*.

Many refinements have been introduced for the rewriting induction to increase its power and efficiency of automated theorem proving. The underlying rewriting mechanism has been replaced by ordered rewriting in [13] so that rewriting induction can also handle non-orientable equations; not only ordered rewriting but also relaxed rewriting is used in the SPIKE theorem prover to get more flexible expansion and simplification rules. Modulo rewriting is also used to deal with non-orientable theorems [1]. Another refinement is to use simplification by conjectures (i.e. equations to prove) [2, 6, 9–11, 13] and a mechanism for disproving incorrect conjectures [6, 9]. A capability of theories with non-free constructors is investigated in [7, 8, 19].

Another direction for extension is to make the framework more general. The SPIKE theorem prover can handle not only equational theories but conditional ones; moreover, inductive theorems can be given not only in equations but also in clauses. Further generalization is given in [11] whose underlying logical theory is replaced with an abstract first-order deductive relation. Stratulat [29] strengthens such an abstraction further by a general abstract inference system that can be used to prove general inductive properties of any first-order deductive relation. Equations with regular constraints can be also treated in [7, 8]. Needless to say, such extensions also benefit from the enhancement of the proving power on the basic rewriting induction system.

It is well-known that an introduction of suitable lemmas often prevents automated inductive theorem proving from divergence. Thus the techniques to introduce suitable lemmas automatically in the process of proving have been investigated [22, 28, 33, 34]. Divergence critic [34] is an automated lemma discovery method for rewriting induction which finds lemmas from a divergent sequence of proofs. The SPIKE theorem prover contains a lemma discovery tool based on the divergence critic. Urso & Kounalis [33] gave a lemma discovery method *Sound Generalization* for monomorphic term rewriting systems which is sound, that is, does not generate incorrect lemmas from correct conjectures. A part of divergence critic is extend to sound one by Shimazu et.al. [28].

In this paper, we present a refined inference system of rewriting induction with an increased capability of non-orientable theorems. We also present how the system is combined with rules for adding sound lemmas and disproving incorrect conjectures. Soundness of the presented systems is shown and effectiveness is demonstrated through examples. A part of our inference system is implemented and we report a preliminary experiment.

[1] Originally, it is called "*term rewriting induction*". The terminology "*rewriting induction*" is introduced in [24].

The rest of the paper is organized as follows. After fixing basic notations (Section 2), we review rewriting induction (Section 3). In Section 4, we present our basic system of rewriting induction. Then, we incorporate a rule for adding sound lemmas (Section 5) and rules for disproving incorrect conjectures (Section 6). Section 7 introduces a modification of systems which turns out to be useful by our preliminary experiment. In Section 8, we compare our system with other rewriting induction provers. Section 9 concludes.

2 Preliminaries

We introduce notations for term rewriting used in this paper. (For details, see [3].) The sets of function symbols and variables are denoted by $\mathcal{F}$ and V, respectively. The set of terms over $\mathcal{F}, V$ is denoted by $\mathrm{T}(\mathcal{F}, V)$. We use $\equiv$ to denote the syntactical equality. We write $u \trianglelefteq t$ if u is a subterm of t. The *root symbol* of a term t is denoted by $\mathrm{root}(t)$ and the set of variables in a term t by $V(t)$.

Let $\Box$ be a constant not occurring in $\mathcal{F}$. A *context* is an element in $\mathrm{T}(\mathcal{F} \cup \{\Box\}, V)$. The constant $\Box$ is called a *hole*. If a context C has n holes in it, we denote by $C[t_1, \ldots, t_n]$ a term obtained by replacing holes with $t_1, \ldots, t_n$ from left to right. A mapping σ from V to $\mathrm{T}(\mathcal{F}, V)$ is called a *substitution*; we identify σ and its homomorphic extension. $\sigma(t)$ is also written as $t\sigma$, which is called an *instance* of the term t. The *domain* of σ is defined by $\mathrm{dom}(\sigma) = \{x \in V \mid x\sigma \not\equiv x\}$. We denote by $\mathrm{mgu}(s, t)$ the *most general unifier* of terms s, t.

A pair $\langle l, r\rangle$ of terms l, r satisfying conditions (1) $\mathrm{root}(l) \in \mathcal{F}$ and (2) $V(r) \subseteq V(l)$ is said to be a *rewrite rule*. A rewrite rule $\langle l, r\rangle$ is denoted by $l \to r$. A *term rewriting system (TRS)* is a set of rewrite rules. Let $\mathcal{R}$ be a TRS. If there exist a context C, a substitution σ, and a rewrite rule $l \to r \in \mathcal{R}$ such that $s \equiv C[l\sigma]$ and $t \equiv C[r\sigma]$, we write $s \to_{\mathcal{R}} t$. We call $s \to_{\mathcal{R}} t$ a *rewrite step*. $\to_{\mathcal{R}}$ forms a relation on $\mathrm{T}(\mathcal{F}, V)$, called the *rewrite relation* of $\mathcal{R}$. A term t is said to be *normal* when there exists no s such that $t \to_{\mathcal{R}} s$. An *equation* $l \doteq r$ is a pair $\langle l, r\rangle$ of terms. When we write $l \doteq r$, however, we do not distinguish $\langle l, r\rangle$ and $\langle r, l\rangle$. The rewrite relation of a set E of equations is defined as $s \leftrightarrow_E t$ if there exist a context C, a substitution σ and an equation $l \doteq r \in E$ satisfying $s \equiv C[l\sigma]$ and $t \equiv C[r\sigma]$. The reflexive closure and reflexive transitive closure of $\to_{\mathcal{R}}$ ($\leftrightarrow_E$) are denoted by $\xrightarrow{=}_{\mathcal{R}}$ (resp. $\xleftrightarrow{=}_E$) and $\xrightarrow{*}_{\mathcal{R}}$ (resp. $\xleftrightarrow{*}_E$). The symmetric closure of $\to_{\mathcal{R}}$ is denoted by $\leftrightarrow_{\mathcal{R}}$. The modulo rewriting relation is defined like this: $\leftrightarrow_{\mathcal{R}/E} = \xleftrightarrow{*}_E \circ \to_{\mathcal{R}} \circ \xleftrightarrow{*}_E$, where $\circ$ denotes the composition of relations. When $s \equiv C[s_1, \ldots, s_n]$, $t \equiv C[t_1, \ldots, t_n]$, and $s_i \to_{\mathcal{R}} t_i$ for all $i = 1, \ldots, n$, we write $s \Vdash_{\mathcal{R}} t$.

The set of *defined function symbols* is given by $\mathcal{D}_{\mathcal{R}} = \{\mathrm{root}(l) \mid l \to r \in \mathcal{R}\}$ and the set of *constructor symbols* by $\mathcal{C}_{\mathcal{R}} = \mathcal{F} \setminus \mathcal{D}_{\mathcal{R}}$. The set of defined symbols appearing in a term t is denoted by $\mathcal{D}_{\mathcal{R}}(t)$. When $\mathcal{R}$ is obvious from its context, we omit the subscript $\mathcal{R}$ from $\mathcal{D}_{\mathcal{R}}, \mathcal{C}_{\mathcal{R}}$. Terms in $\mathrm{T}(\mathcal{C}, V)$ are said to be *constructor terms*; a substitution σ such that $x\sigma \in \mathrm{T}(\mathcal{C}, V)$ for any $x \in \mathrm{dom}(\sigma)$ is called a *constructor substitution*. A term of the form $f(c_1, \ldots, c_n)$ for some $f \in \mathcal{D}$ and

$c_1, \ldots, c_n \in \mathrm{T}(\mathcal{C}, V)$ is said to be *basic*. The set $\{u \trianglelefteq s \mid \exists f \in \mathcal{D}.\ \exists c_1, \ldots, c_n \in \mathrm{T}(\mathcal{C}, V).\ u \equiv f(c_1, \ldots, c_n)\}$ of basic subterms of s is written as $\mathcal{B}(s)$.

A term t is said to be *ground* if $V(t) = \emptyset$. The set of ground terms is denoted by $\mathrm{T}(\mathcal{F})$. If $t\sigma \in \mathrm{T}(\mathcal{F})$, $t\sigma$ is called a *ground instance* of t. The ground instance of a rewrite rule, an equation, etc. is defined similarly. A *ground substitution* is a substitution σ_g such that $x\sigma_g \in \mathrm{T}(\mathcal{F})$ for any $x \in \mathrm{dom}(\sigma_g)$. A TRS $\mathcal{R}$ is said to be *quasi-reducible* if no ground basic term is normal. Without loss of generality, we assume that $t\sigma_g$ is ground (i.e. $V(t) \subseteq \mathrm{dom}(\sigma_g)$) when we speak of an instance $t\sigma_g$ of t by a ground substitution σ_g; and so for ground instances of rewrite rules, equations, etc. An *inductive theorem* of a TRS $\mathcal{R}$ is an equation that is valid on $\mathrm{T}(\mathcal{F})$, i.e. $s \doteq t$ is an inductive theorem if $s\sigma_g \overset{*}{\leftrightarrow}_{\mathcal{R}} t\sigma_g$ holds for any ground instance $s\sigma_g \doteq t\sigma_g$. We write $\mathcal{R} \vdash_{ind} E$ then all equations in E are inductive theorems of $\mathcal{R}$.

A relation R on $\mathrm{T}(\mathcal{F}, V)$ is said to be *closed under substitutions* if $s\ R\ t$ implies $s\sigma\ R\ t\sigma$ for any substitution σ; *closed under contexts* if $s\ R\ t$ implies $C[s]\ R\ C[t]$ for any context C. A *reduction order* (*reduction quasi-order*) is a well-founded partial order (resp. quasi-order) on $\mathrm{T}(\mathcal{F}, V)$ that is closed under substitutions and contexts. For a quasi-order $\succsim$, we let $\approx = \succsim \cap \precsim$ and $\succ = \succsim \setminus \precsim$. A quasi-order $\succsim$ on $\mathrm{T}(\mathcal{F}, V)$ is said to be *ground-total* if $s_g \succsim t_g$ or $s_g \precsim t_g$ for any $s_g, t_g \in \mathrm{T}(\mathcal{F})$.

3 Rewriting Induction

Rewriting induction proposed by Reddy [26] is a method to prove inductive theorems automatically. This section reviews the rewriting induction.

Let $\mathcal{R}$ be a TRS and $>$ a reduction order. We list inference rules of rewriting induction in **Fig.1**. In the figure, the relation $\uplus$ expresses the disjoint union and the ternary operation Expd is defined as:

$$\mathrm{Expd}_u(s, t) = \{C[r]\sigma \doteq t\sigma \mid s \equiv C[u], \sigma = \mathrm{mgu}(u, l), l \to r \in \mathcal{R}, l{:}\mathrm{basic}\}$$

A rewriting induction procedure starts from a pair $\langle E_0, \emptyset \rangle$ where E_0 is the set of conjectures to prove. It successively applies the inference rules to a pair $\langle E, H \rangle$. Intuitively, E is a set of equations to be proved and H is a set of induction hypotheses and theorems already proved. If a derivation eventually reaches the form $\langle \emptyset, H' \rangle$ then $\mathcal{R} \vdash_{ind} E_0$. On the other hand, when none of the rules are applicable for $\langle E, H \rangle$ with $E \neq \emptyset$, the procedure reports "failure" and the procedure may also run forever ("divergence")—in these cases, rewriting induction fails to prove $\mathcal{R} \vdash_{ind} E_0$. We use $\rightsquigarrow$ to denote one step application of an inference rule possibly with a superscript indicating which inference rule is used. $\overset{*}{\rightsquigarrow}$ is the reflexive transitive closure of $\rightsquigarrow$.

*Example 1 (*RI*).* Let

$$\mathcal{R} = \left\{ \begin{array}{ll} \mathsf{plus}(\mathsf{0}, y) & \to y \\ \mathsf{plus}(\mathsf{s}(x), y) & \to \mathsf{s}(\mathsf{plus}(x, y)) \end{array} \right\},$$

Expand

$$\frac{\langle E \uplus \{s \doteq t\},\ H\rangle}{\langle E \cup \mathrm{Expd}_u(s,t),\ H \cup \{s \to t\}\rangle}\ u \in \mathcal{B}(s), s > t$$

Simplify

$$\frac{\langle E \uplus \{s \doteq t\},\ H\rangle}{\langle E \cup \{s' \doteq t\},\ H\rangle}\ s \to_{\mathcal{R}\cup H} s'$$

Delete

$$\frac{\langle E \uplus \{s \doteq s\},\ H\rangle}{\langle E,\ H\rangle}$$

Fig. 1. Inference rules of RI

$E = \{\mathsf{plus}(\mathsf{plus}(x,y),z) \doteq \mathsf{plus}(x,\mathsf{plus}(y,z))\}$, $>$ a lexicographic path order based on the precedence $\mathsf{plus} > \mathsf{s} > \mathsf{0}$. The following is a successful derivation of RI:

$$\begin{aligned}
&\left\langle \left\{ \mathsf{plus}(\mathsf{plus}(x,y),z) \doteq \mathsf{plus}(x,\mathsf{plus}(y,z)) \right\}, \left\{ \right\} \right\rangle \\
\rightsquigarrow^{e}_{\mathrm{RI}} &\left\langle \begin{Bmatrix} \mathsf{plus}(y_0,z) \doteq \mathsf{plus}(\mathsf{0},\mathsf{plus}(y_0,z)) \\ \mathsf{plus}(\mathsf{s}(\mathsf{plus}(x_1,y_1)),z) \doteq \mathsf{plus}(\mathsf{s}(x_1),\mathsf{plus}(y_1,z)) \end{Bmatrix}, \right. \\
&\qquad \left. \left\{ \mathsf{plus}(\mathsf{plus}(x,y),z) \to \mathsf{plus}(x,\mathsf{plus}(y,z)) \right\} \right\rangle \\
\overset{*}{\rightsquigarrow}^{s}_{\mathrm{RI}} &\left\langle \begin{Bmatrix} \mathsf{plus}(y_0,z) \doteq \mathsf{plus}(y_0,z) \\ \mathsf{s}(\mathsf{plus}(x_1,\mathsf{plus}(y_1,z))) \doteq \mathsf{s}(\mathsf{plus}(x_1,\mathsf{plus}(y_1,z))) \end{Bmatrix}, \right. \\
&\qquad \left. \left\{ \mathsf{plus}(\mathsf{plus}(x,y),z) \to \mathsf{plus}(x,\mathsf{plus}(y,z)) \right\} \right\rangle \\
\overset{*}{\rightsquigarrow}^{d}_{\mathrm{RI}} &\left\langle \left\{ \right\}, \left\{ \mathsf{plus}(\mathsf{plus}(x,y),z) \to \mathsf{plus}(x,\mathsf{plus}(y,z)) \right\} \right\rangle
\end{aligned}$$

4 Proving Non-Orientable Theorems

In this section, we present the basic system BRI of our rewriting induction extended with a capability of proving non-orientable theorems.

The inference rules of BRI are presented in **Fig.2**. The system is based on a TRS $\mathcal{R}$ and a reduction quasi-order $\succsim$. Elements of H are rewriting rules, and those of K are equations $l \doteq r$ such that $l \not\succ r$ nor $r \not\succ l$. $K^{\succ}$ and $K^{\approx}$ are instantiations of equations in K whose sides are orientable or equivalent (c.f. [4]):

$$K^{\succ} = \{l\sigma \to r\sigma \mid l \doteq r \in K, l\sigma \succ r\sigma\}$$
$$K^{\approx} = \{l\sigma = r\sigma \mid l \doteq r \in K, l\sigma \approx r\sigma\}$$

Expd2 is an operation introduced in [1] that expands not only the larger side of an equation but both sides of the equation:

$$\mathrm{Expd2}_{u,v}(s,t) = \bigcup \left\{ \mathrm{Expd}_{v\sigma}(t\sigma, s') \mid s' \doteq t\sigma \in \mathrm{Expd}_u(s,t) \right\}$$

Example 2 (Expd2). Let $\mathcal{R}$ be as in *Example 1*, $s \equiv \mathsf{plus}(x,\mathsf{plus}(y,z))$, and $t \equiv \mathsf{plus}(y,\mathsf{plus}(x,z))$. Then $u \equiv \mathsf{plus}(y,z)$ is the only basic subterm of s and $v \equiv \mathsf{plus}(x,z)$ is the only basic subterm of t. We have

$$\mathrm{Expd2}_{u,v}(s,t) = \left\{ \begin{array}{l} \mathsf{plus}(\mathsf{0},z) \doteq \mathsf{plus}(\mathsf{0},z), \\ \mathsf{plus}(\mathsf{s}(x_1),z) \doteq \mathsf{plus}(\mathsf{0},\mathsf{s}(\mathsf{plus}(x_1,z))), \\ \mathsf{plus}(\mathsf{s}(x_2),\mathsf{s}(\mathsf{plus}(y_2,z))) \doteq \mathsf{plus}(\mathsf{s}(y_2),\mathsf{s}(\mathsf{plus}(x_2,z))) \end{array} \right\}.$$

Expand

$$\frac{\langle E \uplus \{s \doteq t\},\ H,\ K\rangle}{\langle E \cup \mathrm{Expd}_u(s,t),\ H \cup \{s \to t\},\ K\rangle}\ u \in \mathcal{B}(s), s \succ t$$

Expand2

$$\frac{\langle E \uplus \{s \doteq t\},\ H,\ K\rangle}{\langle E \cup \mathrm{Expd2}_{u,v}(s,t),\ H,\ K \cup \{s \doteq t\}\rangle}\ u \in \mathcal{B}(s), v \in \mathcal{B}(t), s \not\succ t \wedge t \not\succ s$$

Simplify

$$\frac{\langle E \uplus \{s \doteq t\},\ H,\ K\rangle}{\langle E \cup \{s' \doteq t\},\ H,\ K\rangle}\ s \to_{(\mathcal{R} \cup H \cup K^{\succ})/K \approx} s'$$

Simplify-C

$$\frac{\langle E \uplus \{s \doteq t\},\ H,\ K\rangle}{\langle E \cup \{s' \doteq t\},\ H,\ K\rangle}\ s \overset{*}{\leftrightarrow}_{K \approx} \circ \Vdash_E \circ \overset{*}{\leftrightarrow}_{K \approx} s', s \succsim s' \vee t \succsim s'$$

Delete

$$\frac{\langle E \uplus \{s \doteq t\},\ H,\ K\rangle}{\langle E,\ H,\ K\rangle}\ s \overset{*}{\leftrightarrow}_{K \approx} \circ \Vdash_K \circ \overset{*}{\leftrightarrow}_{K \approx} t$$

Fig. 2. Inference rules of BRI

The system BRI is an extension of eRI of [1] and cRI of [2]. Inference rules *Expand2* and *Simplify-C* are extended (instead, we will impose the ground-totality of $\succsim$ as we will see) from the corresponding rules of eRI and cRI. The biggest difference is that, the system includes not only a capability of conjectures with equivalent sides but also that of conjectures with incomparable sides.

Theorem 1 (soundness of BRI). *Let $\mathcal{R}$ be a quasi-reducible TRS, E a set of equations, $\succsim$ a ground-total reduction quasi-order satisfying $\mathcal{R} \subseteq \succ$. If $\langle E, \emptyset, \emptyset\rangle \overset{*}{\rightsquigarrow}_{\mathsf{BRI}} \langle \emptyset, H, K\rangle$ for some H, K, then $\mathcal{R} \vdash_{ind} E$.*

Proof. Proved based on a method similar to the proofs of [1, 2]. Abstract principle is designed based on the ground-totality. One also needs the commutativity of rewrite steps at parallel positions to show the property of $s_g \Vdash_E t_g$. □

*Example 3 (*BRI*).* Let $\mathcal{R}$ be as in *Example 1* and

$$E = \left\{ \mathsf{plus}(x, \mathsf{plus}(y, z)) \doteq \mathsf{plus}(y, \mathsf{plus}(x, z)) \right\}$$

Let $\succsim$ be a multiset path order based on the precedence $\mathsf{plus} \succ \mathsf{s} \succ \mathsf{0}$. The following is a successful derivation of BRI:

$$\left\langle \left\{ \mathsf{plus}(x, \mathsf{plus}(y, z)) \doteq \mathsf{plus}(y, \mathsf{plus}(x, z)) \right\}, \{\}, \{\} \right\rangle$$

$$\rightsquigarrow^{e2} \left\langle \begin{Bmatrix} \mathsf{plus}(\mathsf{0}, z) \doteq \mathsf{plus}(\mathsf{0}, z), \\ \mathsf{plus}(\mathsf{s}(x_1), z) \doteq \mathsf{plus}(\mathsf{0}, \mathsf{s}(\mathsf{plus}(x_1, z))), \\ \mathsf{plus}(\mathsf{s}(x_2), \mathsf{s}(\mathsf{plus}(y_2, z))) \doteq \mathsf{plus}(\mathsf{s}(y_2), \mathsf{s}(\mathsf{plus}(x_2, z))) \end{Bmatrix}, \right.$$
$$\left. \{\}, \left\{ \mathsf{plus}(x, \mathsf{plus}(y, z)) \doteq \mathsf{plus}(y, \mathsf{plus}(x, z)) \right\} \right\rangle$$

$$\overset{*}{\rightsquigarrow}_s \overset{*}{\rightsquigarrow}_d \left\langle \begin{array}{l} \{\, \mathsf{s}(\mathsf{plus}(x_2, \mathsf{s}(\mathsf{plus}(y_2, z)))) \doteq \mathsf{s}(\mathsf{plus}(y_2, \mathsf{s}(\mathsf{plus}(x_2, z)))) \,\} \\ \{\}, \{\, \mathsf{plus}(x, \mathsf{plus}(y, z)) \doteq \mathsf{plus}(y, \mathsf{plus}(x, z)) \,\} \end{array} \right\rangle$$

$$\rightsquigarrow_{e2} \left\langle \begin{array}{l} \left\{ \begin{array}{l} \mathsf{s}(\mathsf{plus}(0, \mathsf{s}(z))) \doteq \mathsf{s}(\mathsf{plus}(0, \mathsf{s}(z))) \\ \mathsf{s}(\mathsf{plus}(\mathsf{s}(x_3), \mathsf{s}(z))) \doteq \mathsf{s}(\mathsf{plus}(0, \mathsf{s}(\mathsf{s}(\mathsf{plus}(x_3, z))))) \\ \mathsf{s}(\mathsf{plus}(\mathsf{s}(x_3), \mathsf{s}(\mathsf{s}(\mathsf{plus}(y_3, z))))) \doteq \mathsf{s}(\mathsf{plus}(\mathsf{s}(y_3), \mathsf{s}(\mathsf{s}(\mathsf{plus}(x_3, z))))) \end{array} \right\} \\ \{\}, \left\{ \begin{array}{l} \mathsf{s}(\mathsf{plus}(x_2, \mathsf{s}(\mathsf{plus}(y_2, z)))) \doteq \mathsf{s}(\mathsf{plus}(y_2, \mathsf{s}(\mathsf{plus}(x_2, z)))) \\ \mathsf{plus}(x, \mathsf{plus}(y, z)) \doteq \mathsf{plus}(y, \mathsf{plus}(x, z)) \end{array} \right\} \end{array} \right\rangle$$

$$\overset{*}{\rightsquigarrow}_s \overset{*}{\rightsquigarrow}_d \left\langle \begin{array}{l} \left\{ \begin{array}{l} \mathsf{s}(\mathsf{s}(\mathsf{plus}(x_3, \mathsf{s}(z)))) \doteq \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{plus}(x_3, z)))) \\ \mathsf{s}(\mathsf{s}(\mathsf{plus}(x_3, \mathsf{s}(\mathsf{s}(\mathsf{plus}(y_3, z)))))) \doteq \mathsf{s}(\mathsf{s}(\mathsf{plus}(y_3, \mathsf{s}(\mathsf{s}(\mathsf{plus}(x_3, z))))))) \end{array} \right\} \\ \{\}, \left\{ \begin{array}{l} \mathsf{s}(\mathsf{plus}(x_2, \mathsf{s}(\mathsf{plus}(y_2, z)))) \doteq \mathsf{s}(\mathsf{plus}(y_2, \mathsf{s}(\mathsf{plus}(x_2, z)))) \\ \mathsf{plus}(x, \mathsf{plus}(y, z)) \doteq \mathsf{plus}(y, \mathsf{plus}(x, z)) \end{array} \right\} \end{array} \right\rangle$$

$$\rightsquigarrow_{e1} \left\langle \begin{array}{l} \left\{ \begin{array}{l} \mathsf{s}(\mathsf{s}(\mathsf{s}(z))) \doteq \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{plus}(0, z)))) \\ \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{plus}(x_4, \mathsf{s}(z))))) \doteq \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{plus}(\mathsf{s}(x_4), z)))) \\ \mathsf{s}(\mathsf{s}(\mathsf{plus}(x_3, \mathsf{s}(\mathsf{s}(\mathsf{plus}(y_3, z)))))) \doteq \mathsf{s}(\mathsf{s}(\mathsf{plus}(y_3, \mathsf{s}(\mathsf{s}(\mathsf{plus}(x_3, z))))))) \end{array} \right\} \\ \{\, \mathsf{s}(\mathsf{s}(\mathsf{plus}(x_3, \mathsf{s}(z)))) \rightarrow \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{plus}(x_3, z)))) \,\} \\ \left\{ \begin{array}{l} \mathsf{s}(\mathsf{plus}(x_2, \mathsf{s}(\mathsf{plus}(y_2, z)))) \doteq \mathsf{s}(\mathsf{plus}(y_2, \mathsf{s}(\mathsf{plus}(x_2, z)))) \\ \mathsf{plus}(x, \mathsf{plus}(y, z)) \doteq \mathsf{plus}(y, \mathsf{plus}(x, z)) \end{array} \right\} \end{array} \right\rangle$$

$$\overset{*}{\rightsquigarrow}_s \overset{*}{\rightsquigarrow}_d \left\langle \begin{array}{l} \{\}, \{\, \mathsf{s}(\mathsf{s}(\mathsf{plus}(x_3, \mathsf{s}(z)))) \rightarrow \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{plus}(x_3, z)))) \,\} \\ \left\{ \begin{array}{l} \mathsf{s}(\mathsf{plus}(x_2, \mathsf{s}(\mathsf{plus}(y_2, z)))) \doteq \mathsf{s}(\mathsf{plus}(y_2, \mathsf{s}(\mathsf{plus}(x_2, z)))) \\ \mathsf{plus}(x, \mathsf{plus}(y, z)) \doteq \mathsf{plus}(y, \mathsf{plus}(x, z)) \end{array} \right\} \end{array} \right\rangle$$

We note that the system eRI [1] is not capable of this proof, because the conjecture $\mathsf{plus}(x, \mathsf{plus}(y, z)) \doteq \mathsf{plus}(y, \mathsf{plus}(x, z))$ has incomparable sides.

5 Adding Sound Lemmas

It is well-known that an introduction of suitable lemmas often prevents automated inductive theorem proving from divergence. Thus it is very helpful for the success of derivations to add a suitable lemma automatically in the process of proving.

Divergence critic [34] is an automated lemma discovery method for rewriting induction which finds lemmas from a divergent sequence of proofs. The SPIKE theorem prover contains a lemma discovery tool based on the divergence critic. However, the divergence critic may introduce a lemma that is not an inductive theorem. This fact complicates the design of a rewriting induction prover with an automated introduction of lemmas.

Another approach is to add only lemmas that are guaranteed to be inductive theorems (when the initial conjectures are inductive theorems). Urso & Kounalis [33] gave a lemma discovery method called *Sound Generalization* for monomorphic TRSs which is sound, that is, does not generate incorrect lemmas from inductive theorems. A part of divergence critic is extend to sound one by Shimazu et.al. [28].

We incorporate an inference rule for adding sound lemmas (**Fig.3**). We denote by BRIL the obtained system.

$$\textit{Lemma}\quad \frac{\langle E,\ H,\ K\rangle}{\langle E \cup L,\ H,\ K\rangle}\ \mathcal{R} \vdash_{ind} E \Longrightarrow \mathcal{R} \vdash_{ind} L$$

Fig. 3. Additional inference rules of BRIL

Theorem 2 (soundness of BRIL). *Let $\mathcal{R}$ be a quasi-reducible TRS, E a set of equations, $\succsim$ a ground-total reduction quasi-order satisfying $\mathcal{R} \subseteq \succ$. If $\langle E, \emptyset, \emptyset\rangle \overset{*}{\rightsquigarrow}_{\mathsf{BRIL}} \langle \emptyset, H, K\rangle$ for some H, K, then $\mathcal{R} \vdash_{ind} E$.*

Proof. The case for the application of *Lemma* inference rule is safely incorporated into the proof of **Theorem 1** without modifying the abstract principle. □

*Example 4 (*BRIL*).* Let $\mathcal{R}$ and E be as follows:

$$\mathcal{R} = \left\{ \begin{array}{ll} \mathsf{plus}(\mathsf{0}, y) & \to y \\ \mathsf{plus}(\mathsf{s}(x), y) & \to \mathsf{s}(\mathsf{plus}(x, y)) \\ \mathsf{times}(\mathsf{0}, y) & \to \mathsf{0} \\ \mathsf{times}(\mathsf{s}(x), y) & \to \mathsf{plus}(y, \mathsf{times}(x, y)) \end{array} \right\}$$

$$E = \left\{ \mathsf{times}(x, \mathsf{s}(y)) \doteq \mathsf{plus}(x, \mathsf{times}(x, y)) \right\}$$

Then $\mathcal{R}$ is monomorphic. Let $\succsim$ be a multiset path order based on the precedence $\mathsf{plus} \succ \mathsf{s} \succ \mathsf{0}$. The following is a successful derivation of BRIL using the *Lemma* inference with the sound generalization [33].

$$\left\langle \left\{ \mathsf{times}(x, \mathsf{s}(y)) \doteq \mathsf{plus}(x, \mathsf{times}(x, y)) \right\}, \{\}, \{\} \right\rangle$$

$$\rightsquigarrow^{e} \quad \left\langle \begin{array}{l} \left\{ \begin{array}{l} \mathsf{0} \doteq \mathsf{plus}(\mathsf{0}, \mathsf{times}(\mathsf{0}, y)) \\ \mathsf{plus}(\mathsf{s}(y), \mathsf{times}(x_1, \mathsf{s}(y))) \doteq \mathsf{plus}(\mathsf{s}(x_1), \mathsf{times}(\mathsf{s}(x_1), y)) \end{array} \right\} \\ \left\{ \mathsf{times}(x, \mathsf{s}(y)) \to \mathsf{plus}(x, \mathsf{times}(x, y)) \right\}, \{\} \end{array} \right\rangle$$

$$\overset{*}{\rightsquigarrow}^{s}\ \overset{*}{\rightsquigarrow}^{d} \left\langle \begin{array}{l} \left\{ \mathsf{plus}(y, \mathsf{plus}(x_1, \mathsf{times}(x_1, y))) \doteq \mathsf{plus}(x_1, \mathsf{plus}(y, \mathsf{times}(x_1, y))) \right\} \\ \left\{ \mathsf{times}(x, \mathsf{s}(y)) \to \mathsf{plus}(x, \mathsf{times}(x, y)) \right\}, \{\} \end{array} \right\rangle$$

$$\rightsquigarrow^{l} \rightsquigarrow^{sc} \left\langle \begin{array}{l} \left\{ \begin{array}{l} \mathsf{plus}(x_1, \mathsf{plus}(y, \mathsf{times}(x_1, y))) \doteq \mathsf{plus}(x_1, \mathsf{plus}(y, \mathsf{times}(x_1, y))) \\ \mathsf{plus}(y, \mathsf{plus}(x_1, z))) \doteq \mathsf{plus}(x_1, \mathsf{plus}(y, z)) \end{array} \right\} \\ \left\{ \mathsf{times}(x, \mathsf{s}(y)) \to \mathsf{plus}(x, \mathsf{times}(x, y)) \right\}, \{\} \end{array} \right\rangle$$

$$\overset{*}{\rightsquigarrow} \quad \left\langle \begin{array}{l} \{\}, \left\{ \begin{array}{l} \mathsf{s}(\mathsf{s}(\mathsf{plus}(x_3, \mathsf{s}(z)))) \to \mathsf{s}(\mathsf{s}(\mathsf{plus}(x_3, z)))) \\ \mathsf{times}(x, \mathsf{s}(y)) \to \mathsf{plus}(x, \mathsf{times}(x, y)) \end{array} \right\} \\ \left\{ \begin{array}{l} \mathsf{s}(\mathsf{plus}(x_2, \mathsf{s}(\mathsf{plus}(y_2, z)))) \doteq \mathsf{s}(\mathsf{plus}(y_2, \mathsf{s}(\mathsf{plus}(x_2, z)))) \\ \mathsf{plus}(y, \mathsf{plus}(x_1, z)) \doteq \mathsf{plus}(x_1, \mathsf{plus}(y, z)) \end{array} \right\} \end{array} \right\rangle$$

In the derivation, the generalization from the equation $\mathsf{plus}(y, \mathsf{plus}(x_1, \mathsf{times}(x_1, y))) \doteq \mathsf{plus}(x_1, \mathsf{plus}(y, \mathsf{times}(x_1, y)))$ to $\mathsf{plus}(y, \mathsf{plus}(x_1, z))) \doteq \mathsf{plus}(x_1, \mathsf{plus}(y, z))$ is obtained by the sound generalization.

6 Disproving Incorrect Conjectures

Rewriting induction with inference rules for disproving incorrect conjectures has been introduced in [6, 9]. Usefulness of a mechanism for detecting incorrect conjectures is clear.

Our system BRIL is extended to the system BRILD with inference rules for disproving incorrect conjectures as in **Fig.4**.

Decompose

$$\frac{\langle E \uplus \{f(s_1,\ldots,s_n) \doteq f(t_1,\ldots,t_n)\},\ H,\ K\rangle}{\langle E \cup \{s_i \doteq t_i \mid 1 \le i \le n\},\ H,\ K\rangle}\ f \in \mathcal{C}$$

Disproof1

$$\frac{\langle E \uplus \{s \doteq x\},\ H,\ K\rangle}{\bot}\ x \in V \setminus V(s)$$

Disproof2

$$\frac{\langle E \uplus \{f(s_1,\ldots,s_n) \doteq x\},\ H,\ K\rangle}{\bot}\ f \in \mathcal{C}, x \in V$$

Disproof3

$$\frac{\langle E \uplus \{f(s_1,\ldots,s_m) \doteq g(t_1,\ldots,t_n)\},\ H,\ K\rangle}{\bot}\ f \neq g, f, g \in \mathcal{C}$$

Fig. 4. Additional inference rules of BRILD

The next lemma follows easily from the definition.

Lemma 1. *Let $\mathcal{R}$ be a quasi-reducible TRS. Then (1) $\mathcal{R} \vdash_{ind} s \doteq t$ implies $\mathcal{R} \vdash_{ind} \mathrm{Expd}_u(s,t)$ for any $u \in \mathcal{B}(s)$. (2) $\mathcal{R} \vdash_{ind} s \doteq t$ implies $\mathcal{R} \vdash_{ind} \mathrm{Expd2}_{u,v}(s,t)$ for any $u \in \mathcal{B}(s)$ and $v \in \mathcal{B}(t)$.*

Theorem 3 (soundness of BRILD). *Let $\mathcal{R}$ be a quasi-reducible confluent TRS, E a set of equations, $\succsim$ a ground-total reduction quasi-order satisfying $\mathcal{R} \subseteq \succ$. (1) $\langle E, \emptyset, \emptyset\rangle \overset{*}{\rightsquigarrow}_{\mathsf{BRILD}} \langle \emptyset, H, K\rangle$ for some H, K, then $\mathcal{R} \vdash_{ind} E$. (2) $\langle E, \emptyset, \emptyset\rangle \overset{*}{\rightsquigarrow}_{\mathsf{BRILD}} \bot$ then $\mathcal{R} \not\vdash_{ind} E$.*

Proof. (1) Since $\mathcal{R}$ is confluent, $\mathcal{R} \vdash_{ind} f(s_1,\ldots,s_n) \doteq f(t_1,\ldots,t_n)$ iff $\mathcal{R} \vdash_{ind} s_i \doteq t_i$ for all $1 \le i \le n$. Thus, any successful derivation can be modified in such a way that all inferences of *Decompose* rule are replaced by those of *Lemma* rule and *Simplify-C* rule. Then the claim follows from **Theorem 2**. (2) It can be shown by induction on the length of $\langle E, H, K\rangle \overset{*}{\rightsquigarrow}_{\mathsf{BRILD}} \bot$ that $\mathcal{R} \vdash_{ind} H \cup K$ implies $\mathcal{R} \not\vdash_{ind} E$, using **Lemma 1**. □

7 Expanding Quasi-Basic Subterms

In this section, we present a small modification of our system which turned out to be useful for proving some additional theorems in our preliminary experiment.

Definition 1 (quasi-basic). *A term u is quasi-basic w.r.t. a TRS $\mathcal{R}$ if (1)* $\mathrm{root}(u) \in \mathcal{D}$ *and (2) for any $l \to r \in \mathcal{R}$ such that l is basic,* $\mathrm{cap}(u)$ *is unifiable with l then there exists $\sigma = \mathrm{mgu}(\mathrm{cap}(u), l)$ such that σ does not instantiate any variable in $V(\mathrm{cap}(u)) \setminus V(u)$. Here,* $\mathrm{cap}(f(u_1, \ldots, u_n))$ *($f \in \mathcal{D}$) is a term $f(\tilde{u}_1, \ldots, \tilde{u}_n)$ where $\tilde{u}_i$ is obtained from u_i by replacing maximal subterms with defined root symbol by fresh variables. The set of quasi-basic subterms of s is denoted by $\mathcal{QB}(s)$.*

Example 5 (quasi-basic). Quasi-basic subterms of $\mathsf{plus}(x, \mathsf{plus}(y, z))$ (w.r.t. the TRS $\mathcal{R}$ in *Example 1*) are $\mathsf{plus}(y, z)$ and $\mathsf{plus}(x, \mathsf{plus}(y, z))$.

We now extend the notions of $\mathrm{Expd}_u(s,t)$ and $\mathrm{Expd2}_{u,v}(s,t)$ for basic subterms u, v to those for quasi-basic subterms u, v. For $u \in \mathcal{QB}(s)$ and $v \in \mathcal{QB}(t)$, $\mathrm{Expd}_u(s,t)$ and $\mathrm{Expd2}_{u,v}(s,t)$ are defined like this:

$$\begin{aligned}\mathrm{Expd}_u(s,t) &= \{C[r]\sigma \doteq t\sigma \mid s \equiv C[u], \sigma = \mathrm{mgu}(u,l), l \to r \in \mathcal{R}, l{:}\mathrm{basic}\}\\ \mathrm{Expd2}_{u,v}(s,t) &= \bigcup \left\{\mathrm{Expd}_{v\sigma}(t\sigma, s') \mid s' \doteq t\sigma \in \mathrm{Expd}_u(s,t)\right\}.\end{aligned}$$

This definition generalizes those for basic terms. From the definition of quasi-basic term it follows that if v is a quasi-basic term and l is a basic lhs of a rewrite rule unifiable with v and $\sigma = \mathrm{mgu}(v, l)$ then $x\sigma \in \mathrm{T}(\mathcal{C}, V)$ for any $x \in \mathrm{dom}(\sigma) \cap V(v)$. Then the well-definedness of $\mathrm{Expd2}_{u,v}(s,t)$ follows from the next lemma.

Lemma 2. *Let $v \in \mathcal{QB}(t)$ and σ a constructor substitution. Then $v\sigma \in \mathcal{QB}(t\sigma)$.*

Since $\mathcal{B}(s) \subseteq \mathcal{QB}(s)$ for any term s, the following rules are more flexible than *Expand* and *Expand2* rules and sometimes more useful to prove theorems.

Expand′

$$\frac{\langle E \uplus \{s \doteq t\},\ H,\ K\rangle}{\langle E \cup \mathrm{Expd}_u(s,t),\ H \cup \{s \to t\},\ K\rangle}\ u \in \mathcal{QB}(s), s \succ t$$

Expand2′

$$\frac{\langle E \uplus \{s \doteq t\},\ H,\ K\rangle}{\langle E \cup \mathrm{Expd2}_{u,v}(s,t),\ H,\ K \cup \{s \doteq t\}\rangle}\ \begin{array}{l} u \in \mathcal{QB}(s), v \in \mathcal{QB}(t),\\ s \not\succ t \wedge t \not\succ s\end{array}$$

The systems obtain by replacing *Expand* and *Expand2* rules in BRI/BRIL/BRILD by *Expand′* and *Expand2′* rules are denoted by BRI′/BRIL′/BRILD′. The soundness of these systems are obtained by using the following lemma.

Lemma 3. *Let $\mathcal{R}$ be a quasi-reducible TRS and u a quasi-basic term. For any ground constructor substitution σ_g, $u\sigma_g$ is reducible.*

8 Related Works

In this section, we compare our inference system and other closely related rewriting induction provers SPIKE and NICE.

SPIKE[2] is a well-known rewriting induction (*test set induction*) prover. The scope of SPIKE is much broader than ours—it can handle not only equational theories but conditional ones, conjectures can be given not only in equations but also in clauses. A disproving mechanism is also included in the system. The mechanisms of SPIKE for proving non-orientable theorem includes the ordered rewriting and relaxed rewriting [6, 9, 10]. Apart from the original version, recently the new version[3] has became available. The new version of SPIKE is based on 'Descente Infinie' induction [5, 29] and an extended mechanism for dealing with non-orientable theorems has been incorporated [30]. We denote by SPIKE/B and SPIKE/S for the original version and new version of SPIKE respectively.

The most notable difference between SPIKE and BRILD′ is the capability of non-orientable theorems by *Expand2* rule. The inference rule of simplification by conjecture (*Simplify-C*) is essentially from [2] and we refer [2] for the comparison to simplification rules of SPIKE. The inference rule for disproving incorrect conjectures of SPIKE is as follows ([9, 10]):

$$\frac{\langle E \cup \{C\}, H\rangle}{\bot}\ C : \text{quasi-inconsistent}$$

We refer [10] for the definition of quasi-inconsistency (which is too complex to present here). The *Decompose* inference rule is from [10] and the *Disproof1-3* inference rules are from [28]. Our inference rules for disproving incorrect conjectures are simplified by making use of the fact that the underlying TRSs are limited to quasi-reducible TRSs.

NICE[4] is a rewriting induction prover that incorporates two extensions for monomorphic TRSs—namely, term partition techniques [32] and a sound generalization technique [33]. The proofs by rewriting induction with these two new mechanisms run independently. We denote by $\mathsf{NICE_P}$ and $\mathsf{NICE_G}$ the proof with the term partition and the proof with the sound generalization. We also note that NICE is capable of conditional theories. NICE does not have a special mechanism for proving non-orientable theorems but the underlying rewriting is performed by the ordered rewriting. It does not incorpolates mechanisms for disproving incorrect conjectures nor simplification by conjectures.

In **Tab. 1** we list the result of an experiment performed by SPIKE, NICE, and our preliminary implementation of BRILD′. Our implementation incorporates the sound generalization same as $\mathsf{NICE_G}$ to add lemmas automatically. A check in each column indicates the success of proof; all successful proofs of BRILD′ are performed at most 7 (expansion) steps. Some additional conjectures needed to prove $\mathsf{times}(x, y) \doteq \mathsf{times}(y, x)$ and $\mathsf{sum}(\mathsf{app}(xs, ys)) \doteq \mathsf{sum}(\mathsf{app}(ys, xs))$ in the system iRI [1] turn out to be unnecessary in BRILD′. The system BRILD′ can prove non-orientable inductive theorems which have not been capable in other rewriting induction provers. We have also tested equational examples from Dream Cor-

[2] `http://www.loria.fr/equipes/cassis/softwares/spike/`
[3] `http://lita.sciences.univ-metz.fr/~stratula/`
[4] `http://www-sop.inria.fr/coprin/urso/`

(many-sorted) conjectures	SPIKE/B	$\mathrm{NICE_P}$	$\mathrm{NICE_G}$	SPIKE/S	BRILD′
$\mathsf{plus}(x,y) \doteq \mathsf{plus}(y,x)$	$\surd$	$\times$	$\surd$	$\surd$	$\surd$
$\mathsf{plus}(x,\mathsf{plus}(y,z)) \doteq \mathsf{plus}(y,\mathsf{plus}(x,z))$	$\surd$	$\times$	$\surd$	$\surd$	$\surd$
$\mathsf{times}(x,\mathsf{s}(y)) \doteq \mathsf{plus}(x,\mathsf{times}(x,y))$	$\times$	$\times$	$\times$	$\times$	$\surd$
$\mathsf{times}(x,\mathsf{plus}(y,z)) \doteq \mathsf{plus}(\mathsf{times}(x,y),\mathsf{times}(x,z))$	$\times$	$\times$	$\times$	$\times$	$\times$
$\mathsf{plus}(x,y) \doteq \mathsf{plus}(y,x)$, $\mathsf{times}(x,\mathsf{plus}(y,z)) \doteq \mathsf{plus}(\mathsf{times}(x,y),\mathsf{times}(x,z))$	$\times$	$\times$	$\times$	$\times$	$\surd$
$\mathsf{times}(x,y) \doteq \mathsf{times}(y,x)$	$\times$	$\times$	$\times$	$\times$	$\surd$
$\mathsf{sum}(\mathsf{app}(xs,ys)) \doteq \mathsf{sum}(\mathsf{app}(ys,xs))$	$\times$	$\times$	$\times$	$\times$	$\surd$

Table 1. Comparison of BRILD′ and rewriting induction provers

pus[5]; our implementation proves 51 out of 69 examples with 1 sec. timeout for each, while the RI proves 33 examples. In the Appendix, we present a proof of $\{\mathsf{plus}(x,y) \doteq \mathsf{plus}(y,x),\ \mathsf{times}(x,\mathsf{plus}(y,z)) \doteq \mathsf{plus}(\mathsf{times}(x,y),\mathsf{times}(x,z))\}$ scripted from the output of our preliminary implementation. It can be seen that the proof uses the *Expand2* rule many times.

9 Conclusion

We have presented a rewriting induction system BRILD′ with an increased capability of proving non-orientable theorems and that of disproving incorrect theorems. The system is intended to amenable for automation and a part of the system is implemented. Soundness of the system for proving and disproving are shown. It was demonstrated through some examples that our inference system enables simple rewriting induction proofs for some theorems which have not been provable in known rewriting induction provers. A comparison with other rewriting induction provers shows that our approach is useful to enlarge the scope of inductive theorems which can be proved automatically based on rewriting induction. Further implementation and experiments remain as our future work.

Acknowledgments

Thanks are due to anonymous referees for comments and suggestions. This work was partially supported by a grant from JSPS, No. 20500002.

References

1. T. Aoto. Dealing with non-orientable equations in rewriting induction. In *Proc. of the 17th International Conference on Rewriting Techniques and Applications*, volume 4098 of *LNCS*, pages 242–256. Springer-Verlag, 2006.

[5] `http://dream.inf.ed.ac.uk/dc/lib.html`

2. T. Aoto. Soundness of rewriting induction based on an abstract principle. *IPSJ Transactions on Programming*, 49(SIG 1 (PRO 35)):28–38, 2008.
3. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
4. L. Bachmair, N. Dershowitz, and D. A. Plaisted. Completion without failure. In *Resolution of Equations in Algebraic Structures*, volume 2, pages 1–30. Academic Press, 1989.
5. G. Barthe and S. Stratulat. Validation of the JavaCard Platform with implicit induction techniques. In *Proc. of the 14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *LNCS*, pages 337–351. Springer-Verlag, 2003.
6. A. Bouhoula. Automated theorem proving by test set induction. *Journal of Symbolic Computation*, 23:47–77, 1997.
7. A. Bouhoula and F. Jacquemard. Automated induction with constrained tree automata. In *Proc. of the 4th International Joint Conference on Automated Reasoning*. Springer-Verlag, to appear.
8. A. Bouhoula and J.-P. Jouannaud. Automata-driven automated induction. *Information and Computation*, 169(1):1–22, 2001.
9. A. Bouhoula, E. Kounalis, and M. Rusinowitch. Automated mathematical induction. *Journal of Logic and Computation*, 5(5):631–668, 1995.
10. A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14:189–235, 1995.
11. R. Bronsard, U. S. Reddy, and R. W. Hasker. Induction using term orders. *Journal of Automated Reasoning*, 16:3–37, 1996.
12. A. Bundy, D. Basin, D. Hutter, and A. Ireland. *Rippling: Meta-Level Guidance for Mathematical Reasoning*. Cambridge University Press, 2005.
13. N. Dershowitz and U. S. Reddy. Deductive and inductive synthesis of equational programs. *Journal of Symbolic Computation*, 15:467–494, 1993.
14. S. Falke and D. Kapur. Inductive decidability using implicit induction. In *Proc. of the 13th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 4246 of *LNAI*, pages 45–59. Springer-Verlag, 2006.
15. B. Gramlich. Strategic issues, problems and challenges in inductive theorem proving. *Electronic Notes in Theoretical Computer Science*, 125:5–43, 2005.
16. G. Huet and J.-M. Hullot. Proof by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25(2):239–266, 1982.
17. D. Hutter and C. Sengler. INKA: The next generation. In *Proc. of the 13th International Conference on Automated Deduction*, pages 288–292, 1996.
18. J.-P. Jouannaud and E. Kounalis. Automatic proofs by induction in theories without constructors. *Information and Computation*, 82:1–33, 1989.
19. D. Kapur. Constructors can be partial, too. In *Automated reasoning and its applications: essays in honor of Larry Wos*, pages 177–210. MIT Press, 1997.
20. D. Kapur, J. Giesl, and M. Subramaniam. Induction and decision procedures. *Revista de la real academia de ciencas (RACSAM) Serie A: Matematicas*, 98(1):154–180, 2004.
21. D. Kapur, P. Narendran, and H. Zhang. Automating inductionless induction using test sets. *Journal of Symbolic Computation*, 11(1–2):81–111, 1991.
22. D. Kapur and M. Subramaniam. Lemma discovery in automating induction. In *Proc. of the 13th International Conference on Automated Deduction*, volume 1104 of *LNCS*, pages 538–552. Springer-Verlag, 1996.
23. M. Kaufmann, P. Manolios, and J. S. Moore. *Computer-Aided Reasoning: ACL2 Case Studies*. Kluwer Academic Publishers, 2000.

24. H. Koike and Y. Toyama. Inductionless induction and rewriting induction. *Computer Software*, 17(6):1–12, 2000. In Japanese.
25. D. R. Musser. On proving inductive properties of abstract data types. In *Proc. of the 7th Annual ACM Symposium on Principles of Programming Languages*, pages 154–162. ACM Press, 1980.
26. U. S. Reddy. Term rewriting induction. In *Proc. of the 10th International Conference on Automated Deduction*, volume 449 of *LNAI*, pages 162–177. Springer-Verlag, 1990.
27. N. Shankar, S. Owre, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS Prover Guide*. Computer Science Laboratory, SRI International, Menlo Park, CA, Sept. 1999.
28. S. Shimazu, T. Aoto, and Y. Toyama. Automated lemma generation for rewriting induction with disproof. In *The 8th JSSST Workshop on Programming and Programming Languages*, pages 75–89, 2006. In Japanese.
29. S. Stratulat. A general framework to build contextual cover set induction provers. *Journal of Symbolic Computation*, 32:403–445, 2001.
30. S. Stratulat. Combining rewriting with explicit induction to reason on non-orientable equalities. In *Proc. of the 19th International Conference on Rewriting Techniques and Applications*, LNCS, to appear.
31. Y. Toyama. How to prove equivalence of term rewriting systems without induction. *Theoretical Computer Science*, 90(2):369–390, 1991.
32. P. Urso and E. Kounalis. Term partition for mathematical induction. In *Proc. of the 14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *LNCS*, pages 352–366, 2003.
33. P. Urso and E. Kounalis. Sound generalizations in mathematical induction. *Theoretical Computer Science*, 323:443–471, 2004.
34. T. Walsh. A divergence critic for inductive proof. *Journal of Artificial Intelligence Research*, 4:209–235, 1996.

A A Proof for $\{+(x,y) \doteq +(y,x), *(x,+(y,z)) \doteq +(*(x,y),*(x,z))\}$

```
SPECIFICATION:
   [ Nat ]
   [ * : Nat*Nat=>Nat,
     + : Nat*Nat=>Nat,
     s : Nat=>Nat,
     0 : Nat ]
   [ +(0,y) -> y,
     +(s(x),y) -> s(+(x,y)),
     *(0,j) -> 0,
     *(s(i),j) -> +(j,*(i,j)) ]
   (Monomorphic)
      reflective positions: s/1
      downward positions: +/2
      upward positions: +/1
      down-contextual positions:
      up-contextual positions: */1
   [ +(y,x) = +(x,y),
     *(x,+(y,z)) = +(*(x,y),*(x,z)) ]
Start the rewriting induction with
   [ +(y,x) = +(x,y),
     *(x,+(y,z)) = +(*(x,y),*(x,z)) ]
   [ +(0,y) -> y,
     +(s(x),y) -> s(+(x,y)),
```

```
     *(0,j) -> 0,
     *(s(i),j) -> +(j,*(i,j)) ]
Expand2 the equation +(y,x) = +(x,y) at e(L) and e(R).
   [ 0 = 0,
     s(+(x2,0)) = s(x2),
     s(x) = s(+(x,0)),
     s(+(x2,s(x))) = s(+(x,s(x2))) ]
Simplify & Delete.
es:
   [ *(x,+(y,z)) = +(*(x,y),*(x,z)) ]
hs:
   [ ]
ks:
   [ +(y,x) = +(x,y) ]
Expand the L of the equation *(x,+(y,z)) = +(*(x,y),*(x,z)) at e
   [ 0 = +(*(0,y),*(0,z)),
     +(+(y,z),*(i4,+(y,z))) = +(*(s(i4),y),*(s(i4),z)) ]
Try SG to +(+(y,z),+(*(i4,y),*(i4,z))) = +(+(y,*(i4,y)),+(z,*(i4,z)))
Adding New Lemma (SG): +(+(x5,z),+(*(i4,y),x6)) = +(+(x5,*(i4,y)),+(z,x6))
Simplify & Delete.
es:
   [ +(+(x5,z),+(*(i4,y),x6)) = +(+(x5,*(i4,y)),+(z,x6)) ]
hs:
   [ *(x,+(y,z)) -> +(*(x,y),*(x,z)) ]
ks:
   [ +(y,x) = +(x,y) ]
Expand2 the equation +(+(x5,z),+(*(i4,y),x6)) = +(+(x5,*(i4,y)),+(z,x6)) at 1(L) and 1(R).
   [ +(y7,+(*(i4,y),x6)) = +(*(i4,y),+(y7,x6)),
     +(s(+(x10,y10)),+(*(i4,y),x6)) = +(s(+(x10,*(i4,y))),+(y10,x6)) ]
Simplify & Delete.
es:
   [ +(y7,+(*(i4,y),x6)) = +(*(i4,y),+(y7,x6)) ]
hs:
   [ *(x,+(y,z)) -> +(*(x,y),*(x,z)) ]
ks:
   [ +(+(x5,z),+(*(i4,y),x6)) = +(+(x5,*(i4,y)),+(z,x6)),
     +(y,x) = +(x,y) ]
Expand2 the equation +(y7,+(*(i4,y),x6)) = +(*(i4,y),+(y7,x6)) at e(L) and 1(R).
   [ +(*(0,j4),x) = +(0,+(0,x)),
     s(+(x6,+(*(0,j4),x))) = +(0,+(s(x6),x)),
     +(*(s(i5),j5),x) = +(+(j5,*(i5,j5)),+(0,x)),
     s(+(x7,+(*(s(i5),j5),x))) = +(+(j5,*(i5,j5)),+(s(x7),x)) ]
Simplify & Delete.
es:
   [ +(x7,+(+(j5,*(i5,j5)),x)) = +(+(x,x7),+(j5,*(i5,j5))) ]
hs:
   [ *(x,+(y,z)) -> +(*(x,y),*(x,z)) ]
ks:
   [ +(y7,+(*(i4,y),x6)) = +(*(i4,y),+(y7,x6)),
     +(+(x5,z),+(*(i4,y),x6)) = +(+(x5,*(i4,y)),+(z,x6)),
     +(y,x) = +(x,y) ]
Expand2 the equation +(x7,+(+(j5,*(i5,j5)),x)) = +(+(x,x7),+(j5,*(i5,j5)))
   at e(L) and 1(R).
   [ +(+(j,*(i,j)),0) = +(0,+(j,*(i,j))),
     s(+(x5,+(+(j,*(i,j)),0))) = +(s(x5),+(j,*(i,j))),
     +(+(j,*(i,j)),s(x4)) = +(s(+(x4,0)),+(j,*(i,j))),
     s(+(x6,+(+(j,*(i,j)),s(x4)))) = +(s(+(x4,s(x6))),+(j,*(i,j))) ]
Simplify & Delete.
es:
   [ ]
hs:
   [ *(x,+(y,z)) -> +(*(x,y),*(x,z)) ]
ks:
   [ +(x7,+(+(j5,*(i5,j5)),x)) = +(+(x,x7),+(j5,*(i5,j5))),
     +(y7,+(*(i4,y),x6)) = +(*(i4,y),+(y7,x6)),
     +(+(x5,z),+(*(i4,y),x6)) = +(+(x5,*(i4,y)),+(z,x6)),
     +(y,x) = +(x,y) ]
Success
```