On Usefulness of Syntactically Complex Lemmas in Theory Exploration for Inductive Theorems

Haruhiko Sato and Masahito Kurihara

Abstract—**Theory exploration has been investigated as the lemma generation methods which play important role in automation of theorem provers. In our previous research on exploration of equational inductive theorems, we have found that in some cases it is important to find syntactically complex theorems for deriving other theorems which is hard to directly prove by induction. In this paper we discuss on the details of such cases through an example.**

Index Terms—**inductive theorem proving, term rewriting systems, theory exploration**

I. Introduction

With the increasing importance of software systems, the need of *formal verification* is growing to ensure correctness rigorously. In formal verification, typically we have to solve a huge number of *theorem proving* problems. Since the cost of interactive proof by human is expensive, the improvement of automated theorem provers is one of the most important subjects for popularization of formal verification.

In automated theorem proving, it is known to be difficult to prove the type of theorems called *inductive theorems*, the class of theorems requiring mathematical (well-founded) induction to prove, because such theorems may essentially require auxiliary lemmas to prove and in general it is difficult to find automatically such an appropriate intermediate lemma from the original theorem. One of the approaches to finding such lemmas is the top-down way based on the observation of the failed proof attempts such as divergence critic [1], and sound generalization [2], [3]. An another approach for lemma generation is *theory exploration* [4], [5], [6] or lemma discovery. In theory exploration we try to discover as many valid equations (lemmas or theorems) as possible from scratch, instead of trying to find useful lemmas for a specific theorem. The constructed theory, a set of valid equations, is expected to be used as collection of lemmas in the proofs of the original goal or other more difficult theorems.

To enlarge the scope of provable theorems in the exploration, in [7] we proposed an approach of applying the rewriting induction technique [8], [9], a powerful induction scheme based on the theory of term rewriting systems (TRS), to prove conjectures instead of simple structural induction. In the experiments based on the proposed methods, we have found that in some cases syntactically simple theorems which is hard to prove by induction is derived from other syntactically complex theorems which is easy to prove. In this paper, through an example, we discuss the detail of the importance and difficulties of finding such complex theorems in theory exploration for inductive theorems.

This work was supported by JSPS KAKENHI Grant Numbers 16K16032 and 16K00090.

H. Sato is with the Department of Engineering, Hokkai-Gakuen University, Japan. e-mail: h-sato@hgu.jp

M. Kurihara is with the Graduate School of Information Science and Technology, Hokkaido University, Japan. e-mail: kurihara@ist.hokudai.ac.jp

This paper is organized as follows. In Section II we review the basic notions for term rewriting and rewriting induction. In Section III, we show the detail of a case of theory exploration where syntactically complex lemmas play important role. In Section IV we discuss on approaches for finding the syntactically complex but important lemmas and concludes in Section V.

II. Preliminaries

In this section, we briefly review basic notions and concepts for term rewriting systems [10], [11], rewriting induction [8], [9], and divergence critic [1].

A. Term Rewriting Systems

A *signature* $\mathcal{F}$ is a set of function symbols, where each $f \in \mathcal{F}$ is associated with a non-negative integer n, the arity of f. Let $\mathcal{V}$ be a set of variables such that $\mathcal{F} \cap \mathcal{V} = \emptyset$. The set $\mathcal{T}(\mathcal{F}, \mathcal{V})$ of all $\mathcal{F}$-terms over $\mathcal{V}$ is inductively defined as follows: $\mathcal{V} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$ and if $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and $f \in \mathcal{F}$ then $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, where n is the arity of f. We write $s \equiv t$ when the terms s and t are identical. A term s is a *subterm* of t, if either $s \equiv t$ or $t \equiv f(t_1, \ldots, t_n)$ and s is a subterm of some t_i. A *substitution* is a function $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $\mathcal{F}(x) \neq x$ for only finitely many xs. Any substitution σ can be extended to a mapping $\sigma : \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathcal{T}(\mathcal{F}, \mathcal{V})$ by defining $\sigma(f(s_1, \ldots, s_n)) = f(\sigma(s_1), \ldots, \sigma(s_n))$. Application $\sigma(s)$ of σ to s is also written as $s\sigma$. A term t is an *instance* of a term s if there exists a substitution σ such that $s\sigma \equiv t$. A term is a *ground term* if it contains no variables. A term t is a *ground instance* of s if t is a ground term and is an instance of s. A *ground-substitution* σ_g is a substitution whose range is ground term, i.e. $\sigma_g : \mathcal{V} \to \mathcal{T}(\mathcal{F})$. A substitution σ is *more general* than a substitution σ' if there is a substitution δ such that $\sigma' = \sigma\delta$. For two terms s and t, if there is a substitution σ such that $s\sigma \equiv t\sigma$, σ is a *unifier* of s and t. We denote the most general unifier of s and t by $mgu(s, t)$. Let $\Box_i$ be a new symbol which does not occur in $\mathcal{F} \cup \mathcal{V}$ for any $i \geq 0$. A *context*, denoted by C, is a term $t \in T(\Sigma, V \cup \{\Box_1, \Box_2, \ldots\})$ with some occurrences of $\Box_1, \Box_2, \ldots$. $C[s_1, s_2, \ldots]$ denotes the term obtained by replacing $\Box_i$ in C with s_i.

A rewrite rule $l \to r$ is an ordered pair of terms such that l is not a variable and every variable contained in r is also in l. A *term rewriting system (TRS)*, denoted by $\mathcal{R}$, is a set of rewrite rules. The *reduction relation* $\to_{\mathcal{R}} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V}) \times \mathcal{T}(\mathcal{F}, \mathcal{V})$ is defined by $s \to_{\mathcal{R}} t$ iff there exists a rule $l \to r \in \mathcal{R}$, a context C, and a substitution σ such that $s \equiv C[l\sigma]$ and $C[r\sigma] \equiv t$. A term s is *reducible* if $s \to_{\mathcal{R}} t$ for some t; otherwise, s is a *normal form*. A TRS $\mathcal{R}$ is *terminating* if

DELETE	$\langle \mathcal{E} \uplus \{s = s\}, \mathcal{H}\rangle \vdash \langle \mathcal{E}, \mathcal{H}\rangle$
SIMPLIFY	$\langle \mathcal{E} \uplus \{s = t\}, \mathcal{H}\rangle \vdash \langle \mathcal{E} \cup \{s' = t\}, \mathcal{H}\rangle$
	if $s \rightarrow_{\mathcal{R}\cup\mathcal{H}} s'$
EXPAND	$\langle \mathcal{E} \uplus \{s = t\}, \mathcal{H}\rangle \vdash$
	$\langle \mathcal{E} \cup \mathrm{Expd}_p(s,t), \mathcal{H} \cup \{s \rightarrow t\}\rangle$
	if $s\|_p$ is basic and
	$\mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t\}$ is terminating
POSTULATE	$\langle \mathcal{E}, \mathcal{H}\rangle \vdash \langle \mathcal{E} \cup \{s = t\}, \mathcal{H}\rangle$

Figure 1. Inferences rules of rewriting induction

there is no infinite rewrite sequence $s_0 \rightarrow_{\mathcal{R}} s_1 \rightarrow_{\mathcal{R}} \cdots$. A TRS is *ground-reducible* if every ground term is reducible with the TRS. The *root symbol* of a term $s \equiv f(s_1, \ldots, s_n)$ is f and denoted by $root(s)$. The set of all *defined symbols* of $\mathcal{R}$ is defined by $\mathcal{D}_{\mathcal{R}} = \{root(l) \mid l \rightarrow r \in \mathcal{R}\}$. The set of all *constructor symbols* of $\mathcal{R}$ is defined by $C_{\mathcal{R}} = \Sigma \setminus D_{\mathcal{R}}$. A term consisting of only constructor symbols and variables is a *constructor term*. The set of all defined symbols in a term t is denoted by $\mathcal{D}(t)$. The relation $\leftrightarrow_{\mathcal{R}}$ is the symmetric closure of the rewrite relation $\rightarrow_{\mathcal{R}}$. The transitive closure of a relation R is denoted by R^+. The reflexive and transitive closure of a relation R is denoted by R^*.

B. Rewriting Induction

Rewriting induction [8], [9] can be formulated as an inference system on the pair $\langle \mathcal{E}, \mathcal{H}\rangle$ of a set of conjecture equations $\mathcal{E}$ and a set of inductive hypothesis rules $\mathcal{H}$, shown in Figure 1, where the function Expd for a position p and terms s and t is defined as $\mathrm{Expd}_p(s,t) = \{s[r]_p\sigma = t\sigma \mid l \rightarrow r \in \mathcal{R}, \sigma = mgu(s|_p, l), l : \text{basic}\}$.

Definition 1 (inductive theorem)**.** *An equation $s = t$ is called* inductive theorem *of $\mathcal{R}$, denoted by $\mathcal{R} \vdash_{ind} s = t$, if for any ground substitution σ_g, $s\sigma_g \leftrightarrow^*_{\mathcal{R}} t\sigma_g$ holds.*

Theorem 1 (correctness of rewriting induction [8], [9])**.** *Given a ground reducible and terminating TRS $\mathcal{R}$, if there exists a derivation sequence $\langle \mathcal{E}_0, \mathcal{H}_0\rangle \vdash \langle \mathcal{E}_1, \mathcal{H}_1\rangle \vdash \cdots \vdash \langle \mathcal{E}_n, \mathcal{H}_n\rangle$ where all rule in $\mathcal{H}_0$ are inductive theorems of $\mathcal{R}$ and $\mathcal{E}_n$ is empty set, then $\mathcal{E}_0 \cup \mathcal{H}_n$ are inductive theorems in $\mathcal{R}$.*

In addition to the standard inference rules of rewriting induction in Figure 1, in the context of theory exploration we also use the following rule SIMP-POSTULATE, we proposed in [7], to perform simplification using already proved lemmas.

SIMP-POSTULATE	$\langle \mathcal{E} \uplus \{s = t\}, \mathcal{H}\rangle \vdash$
	$\langle \mathcal{E} \cup \{s' = t\}, \mathcal{H} \cup \{l \rightarrow r\}\rangle$
	if $s \rightarrow_{\{l\rightarrow r\}} s', \mathcal{R} \vdash_{ind} l = r$ and
	$\mathcal{R} \cup \mathcal{H} \cup \{l \rightarrow r\}$ is terminating

C. Divergence Critic

Divergence critic [1] is the well-known method for lemma generation based on the identification of the accumulating term structure in the divergent sequence of equations. We briefly show the idea of the method with an example. Let us consider the append $@(xs, ys)$ and reverse $\mathsf{rev}(xs)$ functions on lists, defined as the first 4 rules in Figure 2. The append function returns the list obtained by appending the given two lists. The reverse function $\mathsf{rev}(xs)$ returns the list consisting of elements in the given list xs in reverse order. Then for example the equation $\mathsf{rev}(\mathsf{rev}(xs)) = xs$ is one of the inductive theorems on these functions. However, if we try to prove the theorem without additional lemmas, we fail to prove with the following divergent sequence.

$$\begin{array}{rcl} \mathsf{rev}(\mathsf{rev}(xs)) & = & xs \\ \mathsf{rev}(\mathsf{rev}(xs)@[y]) & = & y : xs \\ \mathsf{rev}((\mathsf{rev}(xs)@[y])@[z]) & = & z : (y : xs) \end{array}$$

We can observe that in each step of the divergent sequence, in LHS a singleton list $[y]$ is appended to the end of the argument of outer rev, while in RHS the element y of the singleton list is prepended. This pattern between two successive equations in the sequence is captured by the following two equations using contexts.

$$\begin{array}{rcl} C[s] & = & t \\ C[D[s]] & = & F[t] \end{array}$$

For the first two equations in the sequence, $C \equiv \mathsf{rev}(\Box), D \equiv \Box@[y], F \equiv y : \Box, s = \mathsf{rev}(xs)$, and $t = xs$. In the two equations the differences between two successive equations is represented by contexts D and F. Such contexts representing differences are calculated by the difference matching algorithm [12]. From this view, combining two equations we can derive new equation $C[D[s]] = F[C[s]]$ which can be used to remove the difference D accumulated in LHS. In this case, we have the following equation.

$$\mathsf{rev}(\mathsf{rev}(xs)@[y]) = y : \mathsf{rev}(\mathsf{rev}(xs))$$

Finally, we obtain the lemma which does not cause divergence by generalizing the equation. In this case, by generalizing the $s \equiv \mathsf{rev}(xs)$ with a fresh variable ws, we have the following lemma which is easy to prove without other lemmas and also can be used to solve original divergence.

$$\mathsf{rev}(ws@[y]) = y : \mathsf{rev}(ws)$$

III. CASE ANALYSIS

In this section, we demonstrate the difficulties in finding useful lemmas in theory exploration of inductive theorems through an example.

As the target axioms for theory exploration, we consider the functions on natural numbers and lists defined by the set $\mathcal{R}$ of rules shown in Figure 2. In this definition, natural numbers $0, 1, 2, \ldots$ are represented by the terms $0, \mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), \ldots$ using the constant function 0 representing 0 and successor function s. We represent lists by terms constructed by the constant nil representing empty list and binary function $:(x, xs)$ representing the list whose head element and the list after the head element are x and xs respectively. For simplicity, in the following descriptions we denote the term representing lists $x : (y : (z : \mathsf{nil}))$ by $[x, y, z]$. Also, we use infix notations $xs + ys$ instead of $+(xs, ys)$ for some binary operators such as $+, -, :, @$. The function $+$ and $-$, binary operator on natural numbers, represents addition and subtraction. The function $\mathsf{len}(xs)$ returns the number of elements in the given list xs. The function $\mathsf{take}(x, ys)$ returns the list consisting of first x elements in ys.

$$\mathcal{R} = \left\{ \begin{array}{lll}
\mathsf{nil}@xs & \to & xs \\
(x : xs)@ys & \to & x : (xs@ys) \\
\mathsf{rev}(\mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{rev}(x : xs) & \to & \mathsf{rev}(xs)@[x] \\
0{+}y & \to & y \\
\mathsf{s}(x){+}y & \to & \mathsf{s}(x{+}y) \\
0{-}y & \to & 0 \\
\mathsf{s}(x){-}0 & \to & \mathsf{s}(x) \\
\mathsf{s}(x){-}\mathsf{s}(y) & \to & x{-}y \\
\mathsf{len}(\mathsf{nil}) & \to & 0 \\
\mathsf{len}(x : xs) & \to & \mathsf{s}(\mathsf{len}(xs)) \\
\mathsf{take}(0, xs) & \to & \mathsf{nil} \\
\mathsf{take}(\mathsf{s}(x), \mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{take}(\mathsf{s}(x), y : ys) & \to & y : \mathsf{take}(x, ys) \\
\mathsf{drop}(0, xs) & \to & xs \\
\mathsf{drop}(\mathsf{s}(x), \mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{drop}(\mathsf{s}(x), y : ys) & \to & \mathsf{drop}(x, ys) \\
\mathsf{sh}(\mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{sh}(x : xs) & \to & x : \mathsf{sh}(\mathsf{rev}(xs)) \\
\mathsf{alt}(\mathsf{nil}, xs) & \to & xs \\
\mathsf{alt}(x : xs, ys) & \to & x : \mathsf{alt}(ys, xs)
\end{array} \right.$$

Figure 2. Axiom rules: functions on natural numbers and lists

The function sh (shuffle) changes the order of elements of the given list by repeating to take the first element and then the last element alternately. For example, we have $\mathsf{sh}([1, 2, 3, 4, 5]) \to^* [1, 5, 2, 4, 3]$. The function alt (alternate) combines the given two lists by taking elements from first and second arguments alternately. For example, we have $\mathsf{alt}([1, 2, 3], [4, 5, 6]) \to^* [1, 4, 2, 5, 3, 6]$. Combining this alternate function with basic list operations, we can calculate the result of shuffle function more efficiently by alternating the given list and the reversed one, then taking the first half of the list. This fact is represented by the following inductive theorem.

$$\mathsf{sh}(xs) = \mathsf{take}(\mathsf{len}(xs), \mathsf{alt}(xs, \mathsf{rev}(xs))) \tag{1}$$

Through the proof of this theorem by rewriting induction, we present how syntactically complex lemmas are utilized to deriving other important lemmas. First, by applying EXPAND rule, we obtain the following conjecture.

$$x : \mathsf{sh}(\mathsf{rev}(xs)) = \mathsf{take}(\mathsf{len}(x : xs), \mathsf{alt}(x : xs, \mathsf{rev}(x : xs)))$$

Then by repeatedly applying SIMPLIFY rule using rewrite rules in $\mathcal{R}$, we obtain the following equation.

$$x : \underline{\mathsf{sh}(\mathsf{rev}(xs))} = x : \mathsf{take}(\mathsf{len}(xs), \mathsf{alt}(\mathsf{rev}(xs)@[x], xs)) \tag{2}$$

At this point, we can apply the inductive hypothesis rule, obtained by orienting (1) from left to right, to the underlined part in (2). Note that this reasoning step cannot be performed in structural induction.

$$x : \mathsf{take}(\mathsf{len}(\mathsf{rev}(xs)), \mathsf{alt}(\mathsf{rev}(xs), \mathsf{rev}(\mathsf{rev}(xs)))) = x : \mathsf{take}(\mathsf{len}(xs), \mathsf{alt}(\mathsf{rev}(xs)@[x], xs)) \tag{3}$$

The LHS of (3) can be simplified using the following two inductive theorems of $\mathcal{R}$, which are relatively small in terms of number of symbols and so are discovered earlier than the original goal (1) in the theory exploration.

$$\begin{array}{lll}
\mathsf{len}(\mathsf{rev}(xs)) & = & \mathsf{len}(xs) \\
\mathsf{rev}(\mathsf{rev}(xs)) & = & xs
\end{array}$$

By applying these equations from left to right to LHS of (3), we obtain the following equations.

$$x : \mathsf{take}(\mathsf{len}(xs), \mathsf{alt}(\mathsf{rev}(xs), xs)) = x : \underline{\mathsf{take}(\mathsf{len}(xs), \mathsf{alt}(\mathsf{rev}(xs)@[x], xs))} \tag{4}$$

Next, we try to simplify RHS of (4) to make the whole equation to a trivial form. The key lemmas are the following inductive theorems.

$$\begin{array}{l}
\mathsf{alt}(\mathsf{rev}(xs)@ys, xs) = \mathsf{alt}(\mathsf{rev}(xs), xs)@ys \qquad (*) \\
\mathsf{take}(x, ys@zs) = \mathsf{take}(x, ys)@\mathsf{take}(x - \mathsf{len}(ys), zs) \\
\mathsf{len}(\mathsf{alt}(xs, ys)) = \mathsf{len}(xs) + \mathsf{len}(ys) \\
x - (x + y) = 0 \\
xs@\mathsf{nil} = xs
\end{array}$$

Using these lemmas from left to right, the underlined part of RHS of (4) is simplified to the corresponding part of LHS of (4) as follows:

$$\begin{array}{l}
\mathsf{take}(\mathsf{len}(xs), \mathsf{alt}(\mathsf{rev}(xs)@[x], xs)) \\
= \mathsf{take}(\mathsf{len}(xs), \mathsf{alt}(\mathsf{rev}(xs), xs)@[x]) \\
= \mathsf{take}(\mathsf{len}(xs), \mathsf{alt}(\mathsf{rev}(xs), xs))@ \\
\quad \mathsf{take}(\mathsf{len}(xs) - \mathsf{len}(\mathsf{alt}(\mathsf{rev}(xs), xs)), [x]) \\
= \mathsf{take}(\mathsf{len}(xs), \mathsf{alt}(\mathsf{rev}(xs), xs))@ \\
\quad \mathsf{take}(\mathsf{len}(xs) - (\mathsf{len}(\mathsf{rev}(xs)) + \mathsf{len}(xs)), [x]) \\
= \mathsf{take}(\mathsf{len}(xs), \mathsf{alt}(\mathsf{rev}(xs), xs))@ \\
\quad \mathsf{take}(\mathsf{len}(xs) - (\mathsf{len}(xs) + \mathsf{len}(xs)), [x]) \\
= \mathsf{take}(\mathsf{len}(xs), \mathsf{alt}(\mathsf{rev}(xs), xs))@\mathsf{take}(0, [x]) \\
= \mathsf{take}(\mathsf{len}(xs), \mathsf{alt}(\mathsf{rev}(xs), xs))@\mathsf{nil} \\
= \mathsf{take}(\mathsf{len}(xs), \mathsf{alt}(\mathsf{rev}(xs), xs))
\end{array}$$

However, we fail to prove the first lemma marked by $(*)$ because it causes following divergence:

$$\begin{array}{l}
\mathsf{alt}(\mathsf{rev}(xs)@ys, xs) = \mathsf{alt}(\mathsf{rev}(xs), xs)@ys \\
\mathsf{alt}(\mathsf{rev}(xs_1)@(x_1 : ys), x_1 : xs_1) = \\
\qquad \mathsf{alt}(\mathsf{rev}(xs_1)@[x_1], x_1 : xs_1)@ys \\
\mathsf{alt}(\mathsf{rev}(xs_2)@(x_2 : x_1 : ys), x_1 : x_2 : xs_2) = \\
\qquad \mathsf{alt}(\mathsf{rev}(xs_2)@(x_2 : [x_1]), x_1 : x_2 : xs_2)@ys
\end{array}$$

and in this case top-down techniques such as divergence critic does not work because there is no simple lemma which can be directly applied to the equations in the divergent sequence.

Instead of trying to find an inductive proof of a theorem directly, in some cases we can derive the theorem from a more general theorem which is easy to find its inductive proof. In this case, we can consider the following inductive theorem.

$$\mathsf{alt}(\mathsf{take}(\mathsf{len}(ys), xs), ys)@\mathsf{drop}(\mathsf{len}(ys), xs) = \mathsf{alt}(xs, ys) \tag{5}$$

By instantiating it with $\sigma = \{xs \mapsto \mathsf{rev}(ys)@xs\}$, we obtain the following theorem.

$$\mathsf{alt}(\mathsf{take}(\mathsf{len}(ys), \mathsf{rev}(ys)@xs), ys)@ \mathsf{drop}(\mathsf{len}(ys), \mathsf{rev}(ys)@xs) = \mathsf{alt}(\mathsf{rev}(ys)@xs, ys)$$

Then the original lemma marked by $(*)$ is obtained by simplifying the theorem by the following two lemmas.

$$\begin{aligned}\mathsf{take}(\mathsf{len}(ys), \mathsf{rev}(ys)@xs) &= \mathsf{rev}(ys)\\ \mathsf{drop}(\mathsf{len}(ys), \mathsf{rev}(ys)@xs) &= xs\end{aligned}$$

The above two lemmas are also hard to prove by induction directly. However, they can also be derived by instantiation and simplification from the following more general lemmas which are easy to prove.

$$\begin{aligned}\mathsf{take}(\mathsf{len}(ys), ys@xs) &= ys\\ \mathsf{drop}(\mathsf{len}(ys), ys@xs) &= xs\end{aligned}$$

In theory exploration, already proved equations are used as lemmas to prove remaining conjectures. So it is important to try to prove useful lemmas such as (5) at an earlier stage of exploration. In our experiments, we used the syntactical simplicity and generality for estimation of usefulness: equations with less symbols and more variables are more useful. This simple heuristic works fine for finding basic and syntactically simple lemmas such as the laws of identity $x+0=x$ earlier. With such simple heuristic, however, semantically general but syntactically complex equations such as (5) are picked later in exploration and so are cannot be used as lemmas in proof of more simpler theorems such as $(*)$.

IV. Discussion

In this section, we discuss about approaches to solve the problem of how to find syntactically complex lemmas, shown in the previous section.

One possible approach is to consider other syntactical metrics for usefulness which is not too sensitive to the syntactical complexity. Instead of simply using the number of all symbols (function symbols and variables) for the complexity, it may be better to use the ratio of the number of function symbols to the number of variables. It also may be effective to distinguish the defined symbols and constructor symbols in calculating the complexity from the number of function symbols because constructor symbols in a term correspond to restriction on the set of data represented by the term, and so equations containing many constructor symbols such as $\mathsf{drop}(\mathsf{s}(\mathsf{s}(\mathsf{s}(x))), :(y, :(z, :(w, \mathsf{nil})))) = \mathsf{nil}$ are considered to be less general one stating on more specific set of data.

Another approach is to analyze the usefulness of an equation with respect to derivability by the equation. In the context of theory exploration where there is set $\mathcal{C}$ of conjecture equations to be proved, we can estimate the importance of an conjecture equation $e \in \mathcal{C}$ as a lemma in this set by the number of conjectures in $\mathcal{C} \setminus \{e\}$ which turns to be derivable as an equational consequence (without inductive proofs) by the help of the equation e. Although the equational derivability is undecidable in general, using the efficient congruence closure algorithm [13] on ground equations we can approximately decide the derivability of equations at reasonable cost.

V. Conclusion

In this paper, we have shown the usefulness of syntactically complex lemmas in theory exploration for inductive theorems through an example of list manipulation functions, and discussed on possible approaches for finding such important lemmas.

The further investigation on the effectiveness of the proposed approaches for finding useful lemmas by experimentation is our future work.

References

[1] T. Walsh, "A divergence critic for inductive proof," *Journal of Artificial Intelligence Research*, vol. 4, pp. 209–235, 1996.
[2] P. Urso and E. Kounalis, "Term partition for mathematical induction," in *Proc. of 14th International Conference on Rewriting Techniques and Applications*, vol. 2706, 2003, pp. 352–366.
[3] T. Aoto, "Sound lemma generation for proving inductive validity of equations," in *Proc. of IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2008)*, 2008, pp. 13–24.
[4] R. McCasland, A. Bundy, and S. Autexier, "Automated discovery of inductive theorems," *Studies in Logic, Grammar and Rhetoric*, vol. 10, no. 23, pp. 135–149, 2007.
[5] M. Johansson, L. Dixon, and A. Bundy, "Conjecture synthesis for inductive theories," *Journal of Automated Reasoning*, vol. 47, no. 3, pp. 251–289, 2011.
[6] K. Claessen, M. Johansson, D. Rosén, and N. Smallbone, "Automating inductive proofs using theory exploration," in *Proceedings of the 24th International Conference on Automated Deduction*, ser. CADE'13. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 392–406.
[7] H. Sato and M. Kurihara, "Discovering inductive theorems using rewriting induction," in *Proc. of 2016 IEEE International Conference on Systems, Man, and Cybernetics*, 2016, pp. 989–993.
[8] U. Reddy, "Term rewriting induction," in *Proc. of 10th Int. Conf. on Automated Deduction*, ser. Lecture Notes in Computer Science, vol. 814, 1990, pp. 162–177.
[9] T. Aoto, "Dealing with non-orientable equations in rewriting induction," in *Proc. of 17th International Conference on Rewriting Techniques and Applications*, ser. Lecture Notes in Computer Science, vol. 4098, 2006, pp. 242–256.
[10] F. Baader and T. Nipkow, *Term Rewriting and All That*. Cambridge University Press, 1998.
[11] TeReSe, *Term Rewriting Systems*. Cambridge University Press, 2003.
[12] D. Basin and T. Walsh, "Difference matching," in *Proc. of 11th International Conference on Automated Deduction*, ser. Lecture Notes in Artificial Intelligence, vol. 607, 1992, pp. 295–309.
[13] R. Nieuwenhuis and A. Oliveras, "Fast congruence closure and extensions," *Information and Computation*, vol. 205, no. 4, pp. 557–580, 2007.