Higher-Order Proof by Consistency

Henrik Linnestad*[1], Olav Lysne**[1] and Christian Prehofer***[2]

[1] Department of Informatics, University of Oslo, PB 1080 Blindern, 0316 Oslo, Norway.
[2] Institut für Informatik, Technische Universität München, Arcisstr. 21, D-80290 München, Germany.

Abstract. We investigate an integration of the first-order method of proof by consistency (PBC), also known as term rewriting induction, into theorem proving in higher-order specifications. PBC may be seen as well-founded induction over an ordering which contains the rewrite relation, and in this paper we extend this method to the higher-order rewrite relation due to Nipkow. This yields a proof procedure which has several advantages over conventional induction. First, it is less control demanding; second, it is more flexible in the sense that it does not instantiate variables precisely with every constructor, but instantiates according to the rewrite rules. We show how a number of technical problems can be solved in order for this integration to work, and point out some desirable refinements that involve challenging problems.

1 Introduction

The field of term rewriting has attracted much attention over the last twenty-odd years, largely triggered by seminal work of Knuth and Bendix on completion [14]. From the late seventies we have seen an ever increasing body of research on methods for the analysis of first-order rewrite systems. For a survey of this part of the field we refer to [4]. Due to their expressive power, higher-order logics are widely used for specification and verification. For the extension of term rewriting in this direction, there exist several different formalisms which integrate typed lambda calculus and term rewrite systems, including Klop [13], Breazu-Tannen [3] and Nipkow [22]. We follow the approach given in the latter work, where a rewriting relation modulo α-, β- and η-conversion is considered.

In this paper we adapt the first-order proof method called *inductionless induction*, or *proof by consistency*, to the higher-order setting. The rationale behind this method, which was first described in a paper by Musser [21], is that the Knuth and Bendix completion process can be used to prove or disprove properties of a rewrite system. This is roughly done by studying the new equations that emerge in the completion process wrt. a notion of consistency. Since 1980 we have seen a lot of work on this first-order method, removing some of its limitations [6, 8], relaxing its close connection with the full completion process [5, 1], and extending the set of rewrite-based specifications that the method applies to [2, 16, 17]. In [26] it was pointed out

* Email: **henrikl@ifi.uio.no**, Phone: +47 22 85 24 05. Partly supported by the Norwegian Research Council.
** Email: **olavly@ifi.uio.no**, Phone: +47 22 85 24 34.
*** Email: **prehofer@informatik.tu-muenchen.de**

that proof by consistency can be seen as induction over a well-founded ordering on the term universe.

The main motivation for this work is to utilize these sophisticated induction techniques for higher-order theorem proving. Assuming that recursive data types are given, we basically exploit the initial structures provided by these types to apply (implicit) induction schemes. This means that in order to prove an equation $s \simeq t$, we instead try to prove $s\sigma \simeq t\sigma$ for every substitution σ that assigns (almost) ground terms to first-order variables of data types. Although our data-types are essentially first-order, they may contain higher-order subterms, e.g. consider induction on lists of functions. For this we need a particular notion of substitutions, since we cannot reason about higher-order terms via ground instances.

Theorem provers like HOL [7] and Isabelle/HOL [23] apply an explicit induction scheme in which variables are instantiated with constructor terms spanning the data type. Our proposed integration of proof by consistency to higher-order equational reasoning can be seen as a generalization of this approach, where an explicit set of constructors for data types is not needed. Since most commonly used interactive theorem provers usually perform induction manually, our techniques are particularly interesting.

The paper is organized as follows. In section 2 we recall basic concepts and notation for term rewriting; our notation is roughly consistent with [22]. The proof procedure itself is given in section 3 followed by a correctness proof in section 4. Some examples are provided in section 5.

2 Preliminaries

From a set of *base types* $\mathcal{B}$, we construct the set of *types* $\mathcal{T}$ with the function space constructor $\to$ in the obvious way. Assume given a set of typed *variables* $\mathcal{V} = \cup_{\tau \in \mathcal{T}} \mathcal{V}_\tau$ and a set of *constants* (function symbols) $\mathcal{F} = \cup_{\tau \in \mathcal{T}} \mathcal{F}_\tau$, where $\mathcal{V} \cap \mathcal{F} = \emptyset$ and $\mathcal{V}_\tau \cap \mathcal{V}_{\tau'} = \mathcal{F}_\tau \cap \mathcal{F}_{\tau'} = \emptyset$ for all distinct τ, τ' from $\mathcal{T}$. The set of *simply typed* λ-*terms* (shorter: *terms*) is then defined inductively as follows:

$$\frac{x \in \mathcal{V}_\tau}{x : \tau} \qquad \frac{c \in \mathcal{F}_\tau}{c : \tau} \qquad \frac{s : \tau \to \tau' \quad t : \tau}{(s\ t) : \tau'} \qquad \frac{x : \tau \quad s : \tau'}{(\lambda x.s) : \tau \to \tau'}$$

Usually we shall implicitly assume that all terms, constants and variables are of suitable type, and omit the type specification.

Every occurrence of the variable x in a (sub-)term of the form $\lambda x.s$ is said to be *bound*. An occurrence of a variable which is not bound, is *free*. We use $\lambda\overline{x_n}.t$ as a simpler notation for $\lambda x_1 \ldots \lambda x_n.t$, and we write $s(t_1, \ldots, t_n)$ or $s(\overline{t_n})$ instead of $(\cdots((s\ t_1)\ t_2)\cdots\ t_n)$. We denote terms by s, t, u, v, constants by f, g, h, bound variables by lowercase w, x, y, z and free variables by uppercase F, G, H, W, X, Y, Z. Variables and constants are called *atoms*. We denote atoms by a, b.

A *position* ω in a term t is a sequence of integers identifying a subterm t/ω of t. In a term of the form $\lambda x.t$, the proper subterms are t and every proper subterm of t. In a term $s(t_1, \ldots, t_n)$, the proper subterms are $s, t_1, \ldots, t_n$ and every proper subterm of these terms. The empty position in a term t is written ϵ, and corresponds to the (non-proper) subterm t. We write $s[t]_\omega$ to indicate the term s, but with s/ω

replaced by the term t. Sometimes we omit the specification of the actual position ω and write $C[t]$ to express a term with t as subterm. Here, the term C is called a *context*.

Two terms s and t are *α-equivalent*, written $=_\alpha$, if each can be obtained from the other by a renaming of bound variables. A *β-reduction* is the transformation of a (sub-)term of the form $((\lambda x.s)\ t)$ into the term that is equal to s except that t is substituted for every occurrence of the variable x that is free in s. A term is in *β-normal form* if it cannot be β-reduced. An *η-reduction* is the transformation of a term $\lambda x.(t\ x)$ into t whenever t has no free occurrences of x. A term $t = \lambda\overline{x_n}.a(\overline{u_m})$ in β-normal form has an *η-long form* defined as $t\uparrow = \lambda\overline{x_{n+k}}.a(\overline{u_m\uparrow}, x_{n+1}\uparrow, \ldots, x_{n+k}\uparrow)$, where t is of type $\overline{\tau_{n+k}} \to \tau$ and $x_{n+1}, \ldots, x_{n+k}$ are fresh variables of appropriate types. The β-normal η-long form of a term t is written $t{\downarrow}$ or $\hat{t}$. If two terms s and t can be obtained from each other by α-, β- and η-conversion, we write $s \equiv t$. A term which is $\equiv$-equivalent to a term with no occurrences of free variables is called a *ground term*. A term t in β-normal form is called a *pattern* if the list of arguments to each occurrence of a free variable is (η-equivalent to) a list of distinct bound variables.

Substitutions are finite and type preserving mappings from variables to terms. We use $\sigma, \gamma, \mu, \rho, \theta$ to denote substitutions. If $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ is a substitution and s is a term, we define $s\sigma$ by $s\sigma = (\lambda\overline{x_n}.s)(\overline{t_n}){\downarrow}$. We say that σ is a *ground substitution* if $t\sigma$ is a ground term, for all terms t.

Let $\mathcal{E}$ be a set of equations. By $=_\mathcal{E}$ we denote *equality modulo* $\mathcal{E}$, that is, the relation resulting from taking α-, β- and η-conversion together with all instances of equations from $\mathcal{E}$ as axioms, and closing under reflexivity, symmetry, transitivity and the congruence laws associated with the constructions $\lambda x.s$ (abstraction) and $(s\ t)$ (application). A *rewrite rule* is an ordered pair $s \to t$ of terms of the same base type, both in β-normal η-long form, such that all variables with free occurrences in t has free occurrences in s as well. We additionally require that the left-hand side of all rewrite rules be patterns. A *rewrite system* is a set of rewrite rules, and may also be viewed as a set of (ordered) equations. Symbols not occurring at the root position of any left-hand side of $\mathcal{R}$ will be called a *constructor* of $\mathcal{R}$. Following Nipkow [22], a rewrite system $\mathcal{R}$ induces a relation $\underset{\mathcal{R}}{\Rightarrow}$ on terms, defined as follows:

$$s \underset{\mathcal{R}}{\Rightarrow} t \quad \Leftrightarrow \quad \exists\, (l \to r) \in \mathcal{R}, \omega, \sigma \mid \hat{s}/\omega \equiv l\sigma \wedge t \equiv \hat{s}[r\sigma]_\omega$$

It should be noted that this relation is invariant under $\equiv$, in the sense that $s' \equiv s \underset{\mathcal{R}}{\Rightarrow} t \equiv t'$ implies $s' \underset{\mathcal{R}}{\Rightarrow} t'$. Hence, $\underset{\mathcal{R}}{\Rightarrow}$ may be viewed as a relation on $\equiv$-equivalence classes of terms.

As in the first-order case, *critical pairs* are equations formed by unifying the left-hand side of one rule with a subterm of the left-hand side of another one and executing the corresponding rewrite step. For an exact definition, consult [22], which easily extends to our context. The set of critical pairs formed by superposing rules from a rewrite system $\mathcal{R}$ onto equations from a set $\mathcal{E}$ is denoted by $CP(\mathcal{R}, \mathcal{E})$.

To improve syntactical control over terms, we shall from now on assume that they are represented by their β-normal η-long forms.

Let $\to$ denote a binary relation on terms. By $\leftarrow$, $\leftrightarrow$, $\overset{+}{\to}$, $\overset{*}{\to}$ and $\overset{*}{\leftrightarrow}$ we denote the inverse of $\to$ and the symmetric, the transitive, the reflexive-transitive and the reflexive-symmetric-transitive closures of $\to$, respectively. We write $s \overset{!}{\to} t$ if $s \overset{*}{\to} t$

and t is *irreducible*, that is, there is no u such that $t \to u$. A substitution $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ is said to be irreducible if $t_1, \ldots, t_n$ are irreducible. If there is no infinite sequence $t_1 \to t_2 \to \cdots \to t_i \to \cdots$, we say that $\to$ is *terminating*. We say that $\to$ is *confluent* if $\overset{*}{\leftarrow} \circ \overset{*}{\rightarrow} \subseteq \overset{*}{\rightarrow} \circ \overset{*}{\leftarrow}$ and *convergent* if it is confluent and terminating. (The symbol $\circ$ is used for relation composition.) A convergent rewrite system $\mathcal{R}$ defines exactly one normal form (up to $\equiv$-equality) for each term t, which we denote by $t{\downarrow_{\mathcal{R}}}$.

Higher-order rewrite systems are not in general *stable*, in the sense that $s \underset{\mathcal{R}}{\rightarrow} t$ implies $s\sigma \underset{\mathcal{R}}{\rightarrow} t\sigma$ for arbitrary substitution σ. However, $\overset{*}{\underset{\mathcal{R}}{\rightarrow}}$ is stable, for every term rewrite system $\mathcal{R}$ (see [19]).

3 Higher-order proof by consistency

The goal of this paper is to achieve inductive proof methods for higher-order systems, that is, to prove or disprove inductive theorems. We do this by integrating a first-order method into the higher-order setting. Consequently, our equational reasoning will essentially be first-order, yet some problems pertaining to the higher-order setting must be solved in order for this to work. For the correctness proof of our procedure, we need to identify a subset of provable equations which will be of special interest to us. More precisely, the equations which we aim at proving are contained in the set of *initial consequences*.

Definition 1. A term t is *first-order rigid*, if all subterms of base type in the η-normal form of t are rigid (i.e. have no free variable as the root symbol). A substitution is called *first-order ground (fo-ground)* if it assigns first-order rigid terms to all variables.[3]

Definition 2 (Initial consequence). We say that a higher-order equation $s \simeq t$ is an *initial consequence* of a higher-order equation set $\mathcal{E}$ if $s\sigma =_{\mathcal{E}} t\sigma$ for all fo-ground substitutions σ.

This definition of initial consequences extends the first-order notion. As a motivation for its design, recall that we are essentially trying to adapt induction over data types to a higher-order setting. The idea is to prove all instances of candidate equations where variables are assigned terms which are as "concrete as possible". This stems from the fact that we must consider any instances, including free variables, for higher-order subterms.

As in the first-order case, we get an operational grip on the initial consequences of $\mathcal{E}$ through a convergent[4] rewrite system $\mathcal{R}$ which represents the same theory as $\mathcal{E}$. In that case, $=_{\mathcal{E}}$ coincides with $\overset{*}{\underset{\mathcal{R}}{\leftrightarrow}}$ ([22]). Since $\mathcal{R}$ is assumed to be convergent, we easily conclude that an equation $s \simeq t$ is an initial consequence of $\mathcal{E}$ iff the $\mathcal{R}$-normal forms of $s\sigma$ and $t\sigma$ are identical, for every fo-ground substitution σ. This last property is given a particular name:

[3] This implicitly refers to all variables in the terms of current interest.

[4] Actually, ground convergence suffices in the first-order case. We conjecture that 'fo-ground' convergence is sufficient for our higher-order setting, but we do not pursue this.

Definition 3 (Initial consistency). We say that a higher-order equation $s \simeq t$ is *initially consistent* with a convergent higher-order rewrite system $\mathcal{R}$ if for all fo-ground substitutions σ we have $s\sigma!_{\mathcal{R}} \equiv t\sigma!_{\mathcal{R}}$.

We see it as our main objective to prove equations. However, proof by consistency also has considerable refutational power; the first-order method is complete in this respect. For this reason, we pursue refutational aspects to some extent here. In the first-order setting, an equation is either consistent or inconsistent. We must take more care in our setting, since our concept of initial consistency may not exhaustively contain all provable equations. For the purpose of refuting equations, we develop a dual notion to initial consistency.

Definition 4. A position ω in a term t is *persistent* in t if all symbols in t above ω are either constructors or lambda binders.

Note that the root position in a term is always persistent.

Definition 5 (Initial inconsistency). We say that a higher-order equation $s = t$ is *initially inconsistent* with a convergent higher-order rewrite system $\mathcal{R}$ if for some fo-ground substitution σ there is a persistent position ω in $t\sigma!_{\mathcal{R}}$ and $u\sigma!_{\mathcal{R}}$ such that $t\sigma!_{\mathcal{R}}/\omega$ and $u\sigma!_{\mathcal{R}}/\omega$ have distinct constructors as roots.

The carrying idea behind the definition of initial inconsistency is as follows: if a term t has a persistent position ω then all symbols above ω persist whatever rewriting is performed, and whatever values are given to the variables. Thus, from the definition of constructor it is easy to see that if two terms s and t have distinct constructors in a position that is persistent in both of them, then for all interpretations of variables, s and t will rewrite to distinct terms.

A set $\mathcal{E}$ of equations is deemed initially consistent with $\mathcal{R}$ if every equation in $\mathcal{E}$ is initially consistent, and initially inconsistent if some equation in $\mathcal{E}$ is initially inconsistent. In order to conclude that $\mathcal{E}$ is initially consistent, we need to verify that there is no *inconsistency witness* of $\mathcal{E}$ wrt. $\mathcal{R}$, that is, an equation $s \simeq t \in \mathcal{E}$ and fo-ground substitution σ such that $s\sigma \overset{!}{\underset{\mathcal{R}}{\rightarrow}} s' \not\equiv t' \overset{!}{\underset{\mathcal{R}}{\leftarrow}} t\sigma$. We write this witness as $\mathcal{W} = (s \simeq t, \sigma)$. To conclude that $\mathcal{E}$ is initially *inconsistent* wrt. $\mathcal{R}$, we must find a *strong inconsistency witness* $\mathcal{W} = (s \simeq t, \sigma)$ of $\mathcal{E}$ wrt. $\mathcal{R}$, where $s \simeq t \in \mathcal{E}$ is initially inconsistent wrt. $\mathcal{R}$ and σ is a fo-ground substitution playing the part of the substitution in definition 5. Note that every strong witness is a witness.

Typical examples of strong witnesses are clashes between different constructors, e.g. in $0 \simeq s(0)$. For a non-strong witness, suppose that f and g are extensionally equal, e.g. both have identical definitions. Then $\lambda x.f(x) \simeq \lambda x.g(x)$ holds under extensionality, though this equation is not by our definition an initial consequence.[5] Furthermore some witnesses, such as $F(a) \simeq F(b)$, can be reduced to strong witnesses via projections, here $F \mapsto \lambda x.x$. Since such cases seem to appear rarely, we do not consider this further.

In the first-order case, the proof by consistency procedure basically computes critical pairs between a set $\mathcal{E}$ of equational conjectures and the convergent rewrite

[5] By extensionality, we have to prove $f(X) \simeq g(X)$ instead of $\lambda x.f(x) \simeq \lambda x.g(x)$, where X is a new free variable. This extension of our framework remains to be investigated.

system $\mathcal{R}$ representing the given theory. This necessitates a unification procedure. However, for higher-order terms, the unification problem is undecidable in general. (Consult Prehofer [25] for a treatment of higher-order unification.) As an important special case, Miller [20] showed that it is decidable whether two patterns are unifiable, and if they are, a most general unifier can be computed, and the Critical Pair Lemma is retained (Nipkow [22]). As is customary, we assume that the left-hand side of rewrite rules are patterns. But even if the equations we want to reason about happen to be patterns, the procedure may generate non-pattern equations, so we have no guarantee that unification will always behave as in the first-order case. For example, we run the risk of being confronted with a situation in which no minimal and complete set of unifiers exists. However, this happens very rarely in practice. In the second-order case with functional (linear) rules, it does not occur at all ([24]).

From now on, let $\mathcal{R}$ be a fixed, convergent rewrite system. To reason about our proposed procedure, we need an ordering on witnesses. For this purpose it is natural to exploit the orderings on higher-order terms proposed by e.g. [27, 18, 11, 10]. Our method can be seen as induction over any such ordering containing $\xrightarrow[\mathcal{R}]{+}$. In the following, we assume an ordering which is based on the well-founded ordering $\xrightarrow[\mathcal{R}]{+}$ and a subterm notion. We write $s \trianglerighteq t$ if $s \equiv C[t]$ for some context C which does not bind any free variables in t. We define $\trianglerighteq_{\mathcal{R}}$ as the transitive closure of $\trianglerighteq \cup \xrightarrow[\mathcal{R}]{*}$. The strict part of $\trianglerighteq_{\mathcal{R}}$, written $\triangleright_{\mathcal{R}}$, is a well-founded ordering (see [25]).

Definition 6. Let $\mathcal{W} = (s \simeq t, \sigma)$ and $\mathcal{W}' = (s' \simeq t', \sigma')$ be witnesses. We write $\mathcal{W} \geq_W \mathcal{W}'$ if $s\sigma \trianglerighteq_{\mathcal{R}} s'\sigma$ and $t\sigma \trianglerighteq_{\mathcal{R}} t'\sigma'$. We define $>_W$ as the strict part of $\geq_W$.

It is easy to see that $>_W$ is a well-founded ordering. We are now ready to introduce the important concept of *covering sets*, which is a straight-forward adaptation of the first-order notion, cf. [1].

Definition 7 Covering set. Let $\mathcal{E}$ and $\mathcal{E}'$ be sets of equations and $\mathcal{R}$ a convergent rewrite system. We say that $\mathcal{E}$ is covered by $\mathcal{E}'$, or that $\mathcal{E}'$ is a *covering set* for $\mathcal{E}$, *with respect to* $\mathcal{R}$ if for every (strong) witness $\mathcal{W} = (s \simeq t, \sigma)$ of an equation $s \simeq t$ in $\mathcal{E}$, there is a (strong) witness $\mathcal{W}' = (s' \simeq t', \sigma')$ of an equation $s' \simeq t'$ in $\mathcal{E}'$ such that $\mathcal{W} >_W \mathcal{W}'$.

We shall return to the operational aspects of this concept. For now, we only state the following evident result.

Lemma 1 *Assume that equation set $\mathcal{E}'$ covers equation set $\mathcal{E}$ with respect to $\mathcal{R}$. If $\mathcal{E}$ is initially inconsistent with $\mathcal{R}$ then $\mathcal{E}'$ is initially inconsistent with $\mathcal{R}$. Furthermore, if $\mathcal{E}$ is not initially consistent with $\mathcal{R}$, then $\mathcal{E}'$ is not initially consistent with $\mathcal{R}$.*

Proof by consistency is based on the computation of critical pairs between the given rewrite system $\mathcal{R}$ and the candidate equation $s \simeq t$ to be investigated. In the same fashion as completion can be seen as a proof normalization process, proof by consistency can be seen as witness normalization. Our correctness proof will illustrate this.

We now describe our proof procedure which accepts as input a set $\mathcal{E}_0$ of equations and aims at proving that every equation in $\mathcal{E}_0$ is initially consistent with $\mathcal{R}$, or at detecting an initially inconsistent equation in $\mathcal{E}_0$. The rewrite system $\mathcal{R}$ is left

unchanged throughout the process, and the data structure is only a set of (unordered) equations. The procedure is started with input $\mathcal{E}_0$, and each step in a derivation sequence $\mathcal{E}_0 \vdash \mathcal{E}_1 \vdash \cdots$ is governed by the following inference system:

DEDUCE:	$\dfrac{\mathcal{E}}{\mathcal{E} \cup \{s \simeq t\}}$	if $s \simeq t \in CP^*(\mathcal{R}, \mathcal{E})$.
SIMPLIFY:	$\dfrac{\mathcal{E} \cup \{s \simeq t\}}{\mathcal{E} \cup \{s' \simeq t\}}$	if $s \underset{\mathcal{R}}{\rightarrow} s'$.
DECOMPOSE:	$\dfrac{\mathcal{E} \cup \{c(\overline{s_n}) \simeq c(\overline{t_n})\}}{\mathcal{E} \bigcup \{s_i \simeq t_i\}_{i=1}^n}$	if c is a constructor symbol.
COVER:	$\dfrac{\mathcal{E} \cup \{s \simeq t\}}{\mathcal{E}}$	if $s \simeq t$ is covered by $\bigcup_i \mathcal{E}_i$.
DELETE:	$\dfrac{\mathcal{E} \cup \{t \simeq t\}}{\mathcal{E}}$	
PROVE:	$\dfrac{\emptyset}{\text{PROOF}}$	
REFUTE:	$\dfrac{\mathcal{E} \cup \{c(\overline{s_m}) \simeq c'(\overline{t_n})\}}{\text{REFUTATION}}$	if c and c' are distinct constructors.

A higher-order pattern t is a *strong pattern,* if every non-variable subterm of base type in the η-normal form is rigid. (This disallows subterms $F(\overline{y_n})$ of base type.) The set $CP^*(\mathcal{R}, \mathcal{E})$ quoted in the DEDUCE rule is the subset of the critical pairs $CP(\mathcal{R}, \mathcal{E})$ in which the involved unifier maps variables only to strong patterns. In practice, this is not a severe restriction, and it has the following pleasant and useful effect:

Lemma 2 *If σ is a fo-ground substitution and μ is a unifier involved in the computation of a critical pair in $CP^*(\mathcal{R}, \mathcal{E})$, then $\mu\sigma$ is fo-ground.*

4 Correctness of the procedure

We now prove the system to be correct in the sense that each step preserves initial consistency as well as initial inconsistency, and that the conclusions made by the halting inference rules are sound. To complete the first part of the proof, we employ our ordering on witnesses and show that they occur in a "down-hill" manner in the process. The careful reader will see that the ordering machinery is essentially only necessary for the COVER rule, where the absence of cycles in the ordering is crucial.

Proposition 3 *Let $\mathcal{E}_k \vdash \mathcal{E}_{k+1}$ be a derivation step of the above procedure with input $\mathcal{E}_0$. If $\mathcal{W} = (s \simeq t, \sigma)$ is a (strong) witness in $\mathcal{E}_k$, then there is a (strong) witness $\mathcal{W}' = (s' \simeq t', \sigma')$ in $\mathcal{E}_{k+1}$ such that $\mathcal{W} \geq_W \mathcal{W}'$.*

Proof. By inspection of each non-halting inference rule which removes an equation.
SIMPLIFY: If $s \underset{\mathcal{R}}{\rightarrow} s'$, then stability of $\underset{\mathcal{R}}{\overset{*}{\rightarrow}}$ yields $s\sigma \underset{\mathcal{R}}{\overset{*}{\rightarrow}} s'\sigma$ for all substitutions σ. Since $\mathcal{R}$ is convergent, $s\sigma!_{\mathcal{R}} = s'\sigma!_{\mathcal{R}}$. This implies that if $\mathcal{W} = (s \simeq t, \sigma)$ is a (strong) witness in $\mathcal{E}_k$, then $\mathcal{W}' = (s' \simeq t, \sigma)$ is a (strong) witness in $\mathcal{E}_{k+1}$. Furthermore, $\mathcal{W} \geq_W \mathcal{W}'$.
DECOMPOSE: If there is a witness $\mathcal{W} = (c(\overline{s_n}) \simeq c(\overline{t_n}), \sigma)$ in $\mathcal{E}_k$, then the $\mathcal{R}$-normal forms $c(\overline{s_n\sigma})!_{\mathcal{R}} = c(\overline{u_n})$ and $c(\overline{t_n\sigma})!_{\mathcal{R}} = c(\overline{v_n})$ would be distinct for some fo-ground substitution σ. But then, $s_i\sigma \underset{\mathcal{R}}{\overset{!}{\rightarrow}} u_i \not\equiv v_i \underset{\mathcal{R}}{\overset{!}{\leftarrow}} t_i\sigma$ for some i, implying that $\mathcal{W}' = (s_i \simeq t_i, \sigma)$ is a witness in $\mathcal{E}_{k+1}$. Furthermore, since $c(\overline{s_n})\sigma \rhd s_i\sigma$ and $c(\overline{t_n})\sigma \rhd t_i\sigma$, we have $\mathcal{W} >_W \mathcal{W}'$. Assume now that $\mathcal{W} = (c(\overline{s_n}) \simeq c(\overline{t_n}), \sigma)$ was a strong witness in $\mathcal{E}_k$, so in addition the terms $c(\overline{s_n\sigma})!_{\mathcal{R}} = c(\overline{u_n})$ and $c(\overline{t_n\sigma})!_{\mathcal{R}} = c(\overline{v_n})$ both have a persistent position ω in which they have occurrences of distinct constructors. Then, the position ω' with $\omega = i.\omega'$ is persistent in u_i and v_i, so $\mathcal{W}' = (s_i \simeq t_i, \sigma)$ is a strong witness as well.
COVER: Assume that $\mathcal{W} = (s \simeq t, \sigma)$ is a $>_W$-minimal (strong) witness of the equation $s \simeq t$ being removed. If $s \simeq t$ is covered by an equation $s' \simeq t' \in \mathcal{E}_j$, where $j \leq k$, then there is a (strong) witness $\mathcal{W}_j$ with $s' \simeq t'$ such that $\mathcal{W} >_W \mathcal{W}_j$, furthermore (by an induction argument over this proposition) there must be a sequence $\mathcal{W}_j \geq_W \mathcal{W}_{j+1} \geq_W \cdots \geq_W \mathcal{W}_k$, where each $\mathcal{W}_i$ is a (strong) witness of an equation in $\mathcal{E}_i$. By well-foundedness of $>_W$, we must have $\mathcal{W} \neq \mathcal{W}_k$. Now recall that $\mathcal{W}$ is assumed to be a $>_W$-minimal witness of $s \simeq t$. This implies that $\mathcal{W}_k$ is a witness of another equation in $\mathcal{E}_k$ than $s \simeq t$.
DELETE: Trivial, since every equation of the form $t \simeq t$ is initially consistent. □

Proposition 4 *If $\mathcal{E}_k \vdash \mathcal{E}_{k+1}$ is a derivation step of the above procedure with input $\mathcal{E}_0$ and there is no (strong) witness in $\mathcal{E}_k$, then there is no (strong) witness in $\mathcal{E}_{k+1}$.*

Proof. By inspection of the inference rules that introduce equations.
SIMPLIFY: Note that if $s \underset{\mathcal{R}}{\rightarrow} s'$, then $s\sigma!_{\mathcal{R}} \equiv s'\sigma!_{\mathcal{R}}$ for all substitutions σ. Thus, if $\mathcal{W}' = (s' \simeq t, \sigma)$ is a (strong) witness in $\mathcal{E}_{k+1}$, then $\mathcal{W} = (s \simeq t, \sigma)$ is a (strong) witness in $\mathcal{E}_k$.
DEDUCE: Suppose that μ is the mgu involved in the computation of a critical pair (in $CP^*(\mathcal{R}, \mathcal{E})$) for an equation $s \simeq t$, so that the actual critical pair deduced is a simplified version of $s\mu \simeq t\mu$. If the critical pair has a (strong) witness σ, then σ is also a (strong) witness of $s\mu \simeq t\mu$ (cf. the immediately preceding treatment of SIMPLIFY). By lemma 2, $\mu\sigma$ is fo-ground. Consequently, $\mu\sigma$ can be seen to be a (strong) witness of $s \simeq t$.
DECOMPOSE: If $\mathcal{W}' = (s \simeq t, \sigma)$ is a witness in $\mathcal{E}_{k+1}$, then $s\sigma!_{\mathcal{R}} \not\equiv t\sigma!_{\mathcal{R}}$. But then $c(\ldots, s, \ldots)\sigma!_{\mathcal{R}} \not\equiv c(\ldots, t, \ldots)\sigma!_{\mathcal{R}}$, so $\mathcal{W} = (c(\ldots, s, \ldots) \simeq c(\ldots, t, \ldots), \sigma)$ is a witness in $\mathcal{E}_k$. If $\mathcal{W}'$ is a strong witness, then there is a persistent position ω in s and t in which these terms have occurrences of distinct constructors. Assume that i is the position of s and t in $c(\ldots, s, \ldots)$ and $c(\ldots, t, \ldots)$, respectively. Then $\mathcal{W}$ is strong as well, since the position $i.\omega$ is persistent in $c(\ldots, s, \ldots)$ and $c(\ldots, t, \ldots)$. □

Applying Propositions 3 and 4 and inspecting the inference rules PROVE and REFUTE, we obtain the following correctness theorem.

Theorem 5 *Assume $\mathcal{E}_0$ is given as input to the above procedure. If the procedure terminates with* PROOF*, then $\mathcal{E}_0$ is initially consistent with $\mathcal{R}$; if it terminates with* REFUTATION*, then $\mathcal{E}_0$ is initially inconsistent with $\mathcal{R}$.*

For practical application, the COVER rule requires methods for deciding whether an equation is covered by the set $\bigcup_i \mathcal{E}_i$ of all generated equations. We do not address this question in full generality, but point out important tractable cases. (A first-order approach to this is summarized in [1].)

Definition 8 Complete position. A position ω in a term t is *inductively complete* with respect to a convergent rewrite system $\mathcal{R}$ if ω is not inside a flexible subterm of t and t/ω is not $\equiv$-equivalent to a variable and $(t/\omega)\sigma$ is an instance of the left-hand side of a rule in $\mathcal{R}$ whenever σ is an irreducible fo-ground substitution.

In the first-order case, it is well known that the property of *ground reducibility* is decidable for terms as well as for equations ([12]). As an important tractable case for higher-order systems, suppose that functions, say f, are totally defined by rewrite rules having left-hand sides $f(c_1(\overline{X_{n_1}}), \overline{Y}), \ldots, f(c_m(\overline{X_{n_m}}), \overline{Y})$, where $c_1, \ldots, c_m$ are constructors spanning a recursive data type. Then any (sub-)term of the form $f(X, \overline{s})$ will correspond to a complete position, unless it occurs below a free variable.

Lemma 6 *Let $\mathcal{R}$ be a convergent rewrite system and $s \simeq t$ be an equation in which a position ω is complete wrt. $\mathcal{R}$. Then, the set of critical pairs computed by superposing rules from $\mathcal{R}$ on $s \simeq t$ in position ω yields a covering set for $s \simeq t$.*

This lemma indicates how COVER can remove an equation after its critical pairs in a complete position have been computed. Another application is as follows: Suppose $s \underset{\mathcal{R}}{\rightarrow} s'$, so that some equation $s \simeq t$ may be simplified by $\mathcal{R}$ to some equation $s' \simeq t$ in $\bigcup_i \mathcal{E}_i$ (equal up to variable renaming) by applying a rule from $\mathcal{R}$ in a position in s which is not below a free higher-order variable. Then, we also have $s\sigma \underset{\mathcal{R}}{\rightarrow} s'\sigma$ for all substitutions σ. In this case it is easily seen that $s \simeq t$ is covered by $s' \simeq t$, so the former equation may be discarded. Finally, suppose given some equation $c(\overline{s_n}) \simeq c(\overline{t_n})$ such that for each j, the equation $s_j \simeq t_j$ occurs (up to variable renaming) in $\bigcup_i \mathcal{E}_i$ or is covered by $\bigcup_i \mathcal{E}_i$. Then $c(\overline{s_n}) \simeq c(\overline{t_n})$ is removed by the COVER rule. Note that the use of COVER sketched above will sometimes correspond to the application of an induction hypothesis.

5 Examples

Example 1. Consider the type of binary trees containing node elements of some fixed, possibly higher-order type. The binary trees are generated by the constructors *empty* giving the empty tree, and *root* taking a node of the tree and two (sub-)trees as arguments. Functions *rev* (mirror image of a tree) and *map* (application of a function to every node) are defined as follows:

$$\begin{array}{ll} rev(empty) & \rightarrow empty \\ rev(root(W, X, Y)) & \rightarrow root(W, rev(Y), rev(X)) \\ map(F, empty) & \rightarrow empty \\ map(F, root(W, X, Y)) & \rightarrow root(F(W), map(F, X), map(F, Y)) \end{array}$$

(Note that in order to improve readability we do not write a higher-order variable F in its η-expanded form $\lambda x.F(x)$ here.) We want to prove the equation

$$map(F, rev(Z)) \simeq rev(map(F, Z)) \tag{1}$$

There is a complete position identifying the subterm $rev(Z)$ in the left-hand side. Computing critical pairs here yields a covering set for equation (1), so we may use DEDUCE and COVER to replace (1) with $map(F, empty) \simeq rev(map(F, empty))$ and $map(F, root(W, rev(Y), rev(X))) \simeq rev(map(F, root(W, X, Y)))$. The former equation is simplified to $empty \simeq empty$ and then deleted; the latter simplifies to

$$\begin{array}{l} root(F(W), map(F, rev(Y)), map(F, rev(X))) \simeq \\ root(F(W), rev(map(F, Y)), rev(map(F, X))) \end{array} \tag{2}$$

If we try to decompose equation (2), we get the new equation $F(W) \simeq F(W)$ as well as $map(F, rev(Y)) \simeq rev(map(F, Y))$ and $map(F, rev(X)) \simeq rev(map(F, X))$. The first of these is deleted, and the last two are equal to (1) modulo variable renaming. Hence (2) is covered and we end up with the empty equation set, proving equation (1).

Example 2. Consider again binary trees defined in the previous example. Add to this rewrite system the rule $(F \circ G)(W) \to F(G(W))$ defining function composition. We now want to prove the equation

$$map(F \circ G, Z) \simeq map(F, map(G, Z)) \tag{3}$$

The empty position in the left-hand side of this equation is complete, so we may replace this equation with the set of critical pairs deduced in this position. First we obtain $empty \simeq map(F, map(G, empty))$, which simplifies to a trivial equation $empty \simeq empty$. The next critical pair is

$$root((F \circ G)(W), map(F \circ G, X), map(F \circ G, Y)) \simeq map(F, map(G, root(W, X, Y)) \tag{4}$$

which simplifies to

$$\begin{array}{l} root(F(G(W)), map(F \circ G, X), map(F \circ G, Y)) \simeq \\ root(F(G(W)), map(F, map(G, X)), map(F, map(G, Y))) \end{array} \tag{5}$$

Decomposing this equation we get the trivial equation $F(G(W)) \simeq F(G(W))$, as well as the equations $map((F \circ G), X) \simeq map(F, map(G, X))$ and $map((F \circ G), Y)) \simeq map(F, map(G, Y))$, which are both equal to (3) modulo variable renaming. Hence we can complete the proof by the COVER rule.

6 Conclusion and further work

We have presented a way of integrating the first-order method of *proof by consistency* to theorem proving in higher-order equational specifications and given a correctness proof of our proposed procedure. The main motivation is to expand the scope of this sophisticated induction technique.

There are several advantages of this approach over conventional induction on first-order data types. First, proof by consistency does not demand an explicit choice of variables on which to do induction, and can be viewed as a more general and less control-demanding method for inductive reasoning. Second, the instantiations for the variable can be more specific than just covering all constructors, since they depend on the rewrite rules, not on the data type declarations. In general, there can even be simultaneous induction on several variables. Furthermore, the convergent rewrite rules can be used for optimizations.

For using our technique in a higher-order theorem proving system, the precise relation between our notion of first-order ground terms and recursive data types should be formalized. For this, one needs to assume that all first-order types are data types, which is acceptable for program verification.

Our approach to higher-order proof by consistency as presented here would benefit from refinements in several directions. First, a facility for applying lemmata would certainly be helpful, just as it is in the first-order case. However, this extension relies heavily on orderings on terms, and it remains to be seen whether existing higher-order orderings are suitable for this problem.

First-order proof by consistency as described in [9] considers the property of *ground reducibility* to give more general criteria for refutation of equations. In full generality, this concept seems to be problematic in higher order, but might be tractable for interesting special cases.

In his treatment of higher-order narrowing, Loría-Sáenz [15] restricts candidate equations to *quasi-first-order* and rewrite rules to *simple* rewrite rules. We believe that corresponding restrictions for proof by consistency would lead to practically useful improvements in the directions that we have just sketched.

References

1. L. Bachmair. *Canonical Equational Proofs*. Birkhäuser, 1991.
2. E. Bevers and J. Lewi. Proof by consistency in conditional equational theories. In *Proc. 2nd International Workshop on Conditional and Typed Rewriting Systems*, volume 516 of *Lect. Not. in Comp. Sci.*, pages 194–205. Springer-Verlag, 1990.
3. V. Breazu-Tannen. Combining algebra and higher-order types. In *Proc. 3rd IEEE Symposium on Logic in Computer Science, Edinburgh (UK)*, July 1988.
4. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 6. Elsevier, 1990.
5. L. Fribourg. A strong restriction on the inductive completion procedure. In *Proc. 13th International Colloquium on Automata, Languages and Programming*, volume 226 of *Lect. Not. in Comp. Sci.*, pages 105–115. Springer-Verlag, 1986.
6. J. A. Goguen. How to prove inductive hypotheses without induction. In W. Bibel and R. Kowalski, editors, *Proc. of the 5th Conference on Automated Deduction*, volume 87 of *Lect. Not. in Comp. Sci.*, pages 356–373. Springer-Verlag, 1980.
7. M. J. C. Gordon. HOL: A proof generating system for higher-order logic. In G. Birtwistle et al., editor, *VLSI Specification, Verification and Synthesis*. Kluwer Academic Press, 1988.
8. G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25(2):239–266, 1982.

9. J.-P. Jouannaud and E. Kounalis. Automatic proofs by induction in equational theories without constructors. In *Proc. Logic in Computer Science*, pages 358–366, 1986.
10. J.-P. Jouannaud and A. Rubio. A recursive path ordering for higher-order terms in η-long β-normal form. In H. Ganzinger, editor, *Proc. 7th International Conference on Rewriting Techniques and Applications*, volume 1103 of *Lect. Not. in Comp. Sci.* Springer-Verlag, 1996.
11. S. Kahrs. Towards a domain theory for termination proofs. In *Proc. of the 6th International Conference on Rewriting Techniques and Applications*, volume 914 of *Lect. Not. in Comp. Sci.*, pages 241–255. Springer-Verlag, 1995.
12. D. Kapur, P. Narendran, and H. Zhang. On sufficient-completeness and related properties of term rewriting systems. *Acta Informatica*, 24(4):395–415, 1987.
13. J. W. Klop. *Combinatory Reduction Systems.* Mathematical Centre Tracts 127, Mathematisch Centrum,Amsterdam, 1980.
14. D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, 1970.
15. C. A. Loría-Sáenz. *A Theoretical Framework for Reasoning about Program Construction Based on Extensions of Rewrite Systems.* PhD thesis, Universität Kaiserslautern, 1993.
16. O. Lysne. Proof by consistency in constructive systems with final algebra semantics. In *Proc. 3rd International Conference on Algebraic and Logic Programming, Pisa (Italy)*, volume 632 of *Lect. Not. in Comp. Sci.*, pages 276–290. Springer-Verlag, 1992.
17. O. Lysne. Extending Bachmair's method for proof by consistency to the final algebra. *Information Processing Letters*, 51:303–310, 1994.
18. O. Lysne and J. Piris. A termination ordering for higher order rewrite systems. In *Proc. 6th Conference on Rewriting Techniques and Applications, Kaiserslautern (Germany)*, volume 914 of *Lect. Not. in Comp. Sci.*, pages 26–40. Springer-Verlag, 1995.
19. R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. Technical report, Institut für Informatik, Technische Universität München, August 1994.
20. D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. In *Extensions of Logic Programming*, volume 475 of *Lect. Not. in Comp. Sci.*, pages 253–281. Springer-Verlag, 1991.
21. D. L. Musser. On proving inductive properties in abstract data types. In *Proceedings of the 7th Annual ACM Symposium on Principles of Programming Languages*, pages 154–162, January 1980.
22. T. Nipkow. Higher-order critical pairs. In *Proc. of the 6th IEEE Symposium on Logic in Computer Science*, pages 342–359, 1991.
23. L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lect. Not. in Comp. Sci.* Springer-Verlag, 1994.
24. C. Prehofer. Decidable higher-order unification problems. In *Proc. 12th International Conference on Automated Deduction, Nancy*, volume 814 of *Lect. Not. in Art. Intell.*, pages 635–649. Springer-Verlag, 1994.
25. C. Prehofer. *Solving Higher-Order Equations: From Logic to Programming.* PhD thesis, Technische Universität München, 1995.
26. U. S. Reddy. Term rewriting induction. In *Proc. 10th International Conference on Automated Deduction, Kaiserslautern*, volume 449 of *Lect. Not. in Comp. Sci.*, pages 162–177. Springer-Verlag, 1990.
27. J. van de Pol. Termination proofs for higher-order rewrite systems. In *1st International Workshop on Higher-Order Algebra, Logic and Term Rewriting*, volume 816 of *Lecture Notes in Computer Science*, pages 305–325. Springer-Verlag, 1993.

This article was processed using the LaTeX macro package with LLNCS style