Nordic Journal of Computing 1(1994), 135–156.

HEURISTICS FOR COMPLETION IN AUTOMATIC PROOFS BY STRUCTURAL INDUCTION*

OLAV LYSNE
Department of Informatics
University of Oslo
P.O. Box 1080 Blindern
N–0316 OSLO, NORWAY
Olav.Lysne@ifi.uio.no

Abstract. A method for proof by structural induction is studied, and problems of automatizing the method is investigated. We specially consider the equational part of such proofs and we observe that the ability to cope with possibly infinite searches for non-existent equational proofs is crucial. Completion as a means to find an equational proof of equivalence of two given terms is studied. By heuristics we weaken the requirements of completeness on the resulting set, and thereby present modifications of both standard completion and ordered completion which guarantee termination.

CR Classification: F.3.1

Key words: Automated Verification, Initial Algebra, Structural Induction

1. Introduction

In the mid seventies the idea of modeling data types algebraically emerged, the basic thought being that each state in a data structure could be represented by a ground term built from some set of function symbols [13, 14]. This gave syntactical control over the data types, and allowed functions operating on the data to be defined by induction on the structure of the terms. We have since the birth of this idea seen several programming languages that are built on this notion of abstract data types, see e.g. [5, 12, 15].

Having programs represented algebraically it is a natural next step to try to use algebraic methods to prove properties of them. We have basically seen two different approaches to such proofs. The first is *proof by structural induction* which was introduced by Burstall [4]. This proof method is based on explicit induction on the complexity of terms, and has later been elaborated by many authors [7, 9, 11, 13]. The second approach is *proof by consistency*, which is sometimes also referred to as inductionless induction. This is a method which is based on searching for inconsistencies by a process inspired

* This research was supported by The Research Council of Norway

Received October 1993. Accepted March 1994.

by Knuth and Bendix completion [20], and the idea first emerged in a paper by Musser [24]. This seminal paper triggered a lot of research extending the applicability and strength of proof by consistency [10, 18, 19, 21, 23].

Comparing the two proof methods is not an easy task since either one of them is superior on some examples. The general advantage of proof by consistency is that it is not based on interaction from the user. Proof by structural induction, however, is not that easily automatizable, first because it is difficult to decide which variable in the expression under consideration one should perform the induction on, and secondly because the equational part of proofs by structural induction is often non-trivial. Even so, it seems that proof by structural induction does succeed more often than proofs by inductionless induction. This view is supported by Garland and Guttag in [9], and Goguen writes in [11]: *Experience indicates that it is generally easier to prove things with structural induction... (structural induction) does not produce an uncontrollable explosion of strange new rules that often seem to gradually become less and less relevant.* It is anyhow an indisputable fact that the examples yet seen of proofs by consistency are dwarfed by some of the proofs found using e.g. the Boyer-Moore theorem prover [3].

In this paper we work towards an automatization of proofs by structural induction. The equational part of such proofs is given special attention, and we suggest using a completion based method. This does introduce the problem of *the explosion of gradually less and less relevant rules.* We therefore present heuristics handling the special termination problems attached to completion. Our method has been implemented in SIMULA, and the examples we provide are generated by our implementation. The outline of the paper is as follows: Sections 2 and 3 present preliminary notions, sections 4 and 5 handle standard completion, whereas ordered completion is treated in sections 6 and 7. Finally we demonstrate our methods by applying them to some of the most popular examples in the literature.

2. Notation and basic concepts

Let $\mathcal{F}$ be a finite set of function symbols, each with fixed arity. Furthermore let $\mathcal{V}$ be a countable set of variables. The set of *terms* $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is defined recursively as the smallest set containing $\mathcal{V}$ such that $f(t_1, \ldots, t_n)$ is a term if $t_1, \ldots, t_n$ are terms, $f \in \mathcal{F}$ and n is the arity of f ($n \geq 0$). A function with arity zero is called a constant. A term containing no variables is a *ground term.* The set of all ground terms is denoted $\mathcal{G}$.

A term may be viewed as an ordered tree in the obvious way. A *position* in a term can be viewed as a finite sequence of natural numbers, pointing out a path from the root of the term. The place in the term where the path ends is the actual position. The subterm of t at position i, written $t[i]$, is the subterm of t which has its root symbol at position i. The result of replacing the subterm of t at position i by the term u is written $t[i/u]$.

To each function symbol is assigned a *profile.* The profile is a pair of

types in which the first element is a Cartesian product of types defining the functions domain, and the second element is the codomain of the function. A *well formed* term is a term in which all functions has correctly typed arguments according to their profiles. Whenever we use the word *term* from now on, we implicitly assume that it is well formed.

Let $\mathcal{C}_{T_1} \subset \mathcal{F}, \ldots, \mathcal{C}_{T_n} \subset \mathcal{F}$ be disjoint sets of function symbols such that the codomain of a function in $\mathcal{C}_{T_i}$ is T_i. Furthermore let $\mathcal{C}$ be the union of all these sets. We call $\mathcal{C}$ the set of *constructors*. The set $\mathcal{GC}$ is defined to be the set of all well formed ground terms containing only function symbols from $\mathcal{C}$. The *value set* of type T_i is the set of terms in $\mathcal{GC}$ which has an element from $\mathcal{C}_{T_i}$ as its leading function symbol.

A *substitution* is a finite set of pairs consisting of one variable and one term. We shall denote substitutions like this: $\{x_1 \mapsto t_1, x_2 \mapsto t_2, \ldots\}$ where x_i is a variable and t_i is a term, or by lowercase Greek letters when no details are needed. Applying a substitution σ to a term t consists of simultaneously replacing every occurrence of a σ-variable in t with the term corresponding to the variable in σ. The result of the application is written $t\sigma$.

An *equation* is an unordered pair of terms, written $t = u$. A set E of equations defines a *variety* $\mathcal{V}(\mathcal{E})$. This is the class of algebras which are models of the equations in E. An equation $t = u$ is said to be valid in $\mathcal{V}(\mathcal{E})$, written $t =_E u$, if it is true in all models in the variety. It is well known that this is equivalent to $t = u$ being derivable from E using replacement of equals by equals.

Equations may also be used as definitions, defining functions by induction on their $\mathcal{GC}$ arguments. In order to be consistent with the type definitions above the *defining equations*, DE, should be such that for every term $t \in \mathcal{G}$ there should be a term $u \in \mathcal{GC}$ such that $t =_{DE} u$. The initial algebra (see [6] for further detail) in $\mathcal{V}(\mathcal{DE})$ is denoted $\mathcal{I}(\mathcal{DE})$, and the corresponding equivalence relation is $=_{\mathcal{I}(\mathcal{DE})}$. It is well known that $t =_{\mathcal{I}(\mathcal{DE})} u$ whenever $t\sigma =_E u\sigma$ for every ground substitution σ. If DE satisfies the restriction above, $t =_{\mathcal{I}(\mathcal{DE})} u$ whenever $t\sigma =_E u\sigma$ for every $\mathcal{GC}$-substitution σ.

A *rewrite rule* is an ordered pair of terms, written $s \to t$. It states that any subterm of a term l which is an instance of s may be replaced by the same instance of t in a rewrite step. If R is a set of rewrite rules we shall let $\to_R$ denote a rewrite step involving a rule from R, and $\to_R^*$ a possibly empty sequence of such steps. A *convergent set of rewrite rules* is a set which satisfies termination and confluence. That is, a term can only go through finitely many rewrite steps with the set of rules, and that no matter what rewrite path is chosen, the same irreducible term will be the result of the rewriting. By $t!_R$ we shall mean the term t fully reduced by the terminating set R of rewrite rules. A set R of rewrite rules is said to be *complete* for a theory consisting of the set E of equations if it is convergent, and all equational proofs that can be performed by using E has a corresponding rewrite proof in R and vice versa. Easy guidelines can be given to ensure that a set of defining equations constitute a convergent set of rewrite rules.

An ordering $\succ$ on terms is said to be *well founded* if there is no infinite

sequence of terms $t_1 \succ t_2 \succ \ldots$. It satisfies the *subterm property*, if $s \succ t$ whenever t is a subterm of s. Furthermore it is *monotone over substitutions*, if $s\sigma \succ t\sigma$ for any σ whenever $t \succ s$, and over *context application*, if $s[i/t] \succ s[i/t']$ whenever $t \succ t'$. We shall in the sequel assume that the term orderings have the subterm property as well as well foundedness and monotonicity.

3. Proof by structural induction

Let E be a set of equations. From Birkhoff's theorem we know that if there exists an equational proof by E between the terms t and u, then the equation $t = u$ is valid in all models of E and vice versa. Obviously $\mathcal{I}(\mathcal{DE})$ is a model of DE, thus whenever we find an equational proof between t and u we may conclude that $t = u$ is valid in the initial algebra as well. Unfortunately the indicated implication does only go one way, therefore the proof concept consisting of replacing equals by equals is not complete for the initial algebra.

We shall view structural induction as a proof inference for the initial algebra which allows us to replace one proof obligation with that of several others. If we are not able to prove an equation valid in some initial model by an equational proof, we may perform an induction inference and complete the inductive proof by finding some other equational proofs instead. Let us present a generalised inference rule for proof by structural induction. Let $A(f)$ be the arity of the function f. Furthermore let $\mathcal{C}_T$ be the set of constructors of the type T. The induction schema for proving $t =_{\mathcal{I}(DE)} u$ by induction on the variable x of type T is

$$\frac{t\{x \mapsto g(x_1, \ldots, x_{A(g)})\} \;=_{\mathcal{I}(DE_g)}\; u\{x \mapsto g(x_1, \ldots, x_{A(g)})\} \text{ ; for all } g \in \mathcal{C}_T}{t =_{\mathcal{I}(DE)} u}$$

where $DE_g = DE \cup \{t\{x \mapsto x_j\} = u\{x \mapsto x_j\} | 1 \leq j \leq A(g) \wedge x_j \in T\}$, and each of $x_1, \ldots, x_{A(g)}$ is either a new constant of type T, or a new variable of a type different from T. The requirement that every term in the formula shall be well formed gives us for each $x_1, \ldots, x_{A(g)}$ whether it is a fresh constant or a fresh variable. We are now ready to give a procedural description of proof by structural induction. Let P be the set of inductive consequences we want to prove:

LOOP WHILE NONEMPTY(P) DO

1. Choose an arbitrary item from P.
2. Try to find an equational proof for the item in the set of equations assigned to that particular item.
3. If found, remove the item from P.
4. If not, use the induction schema on one of the variables of the item, to replace this inductive problem with several others.
   If there are no variables then exit LOOP.

OD

IF EMPTY(P) THEN Terminate with SUCCESS ELSE Terminate with FAIL.

If we aim at automatizing the proof process we encounter two problems, the first of which is choosing the correct induction variable. There are cases where the wrong choice of induction variable entirely blocks the possibility of finding a proof, but very often this can be remedied by introducing an additional level of induction in which the correct variable is chosen (see e.g. the second part of example 1 in section 8). The second problem, which is the one we shall address in the rest of the paper, is the search for equational subproofs.

The fact that the definition of abstract data types in principle leads to convergent sets of rewrite rules makes the search for equational subproofs simple in many cases. During a proof by structural induction, however, lemmas and induction hypotheses are added to the picture, and these new equations frequently destroys both termination and confluence. This means that the equational proofs have to be searched for in a general setting, and the problem of existence of such proofs is undecidable.

The equational part of inductive proofs is treated in a rather ad hoc manner by existing theorem provers [11, 8, 7], in that the lemmas and induction hypotheses are either assumed to maintain convergence when added to DE, or in that they are only applied to DE-minimal forms. In the Boyer-Moore theorem prover [3] there is a feature called cross-fertilisation which is more powerful than the two mentioned above, but also this method presupposes that the equational proofs are of a special form. In [22] a general method for the equational part of proofs by structural induction is presented, but this method requires all defining equations to be left linear.

An approach which does not presuppose anything about the proof one is looking for is to start a completion procedure, and hope that it is able to transform the given set of equations into a convergent set of rewrite rules. Completion based semi decision processes for the equational validity problem have been developed by Hsiang and Rusinowitch [16] and Bachmair [1]. The problem with both these approaches is that they most often diverge if the proof they are looking for does not exist. While searching for a proof by structural induction, one will frequently look for equational proofs that do not exist, simply because the actual inductive proof requires another level of induction. Therefore a method for handling these cases is crucial.

We suggest a completion-based method to handle the equational part of inductive proofs, and present a flexible heuristic controlling the halting problem of the completion process. This heuristic is closely connected to the complexity of the equation we are trying to prove, and thus adjusts naturally to the expected complexity of the proofs. The way we approach the problem is by observing the following: When the completion process diverges, it creates an infinite convergent set of rewrite rules. Still only finitely many of these rules are necessary for computing the minimal form of the terms we are studying. The rest of this infinite set resembles what Goguen calls "new rules that gradually becomes less and less relevant". On this background we shall present completion algorithms which in a heuristic manner leaves

out the rewrite rules and equations that do not seem to contribute to the rewriting of the terms in the equation we are proving.

4. Completion wrt. a set of terms

KB-completion is a process which takes a set E of equations and a well founded term ordering $\succ$ as input, and attempts to generate a set R of rewrite rules which terminate by $\succ$ and which is complete wrt. E. It is usually described by a set of transition rules:

Orient	$(E \cup \{s = t\}, R)$	$\vdash$	$(E, R \cup \{s \to t\})$	if $s \succ t$.
Deduce	(E, R)	$\vdash$	$(E \cup \{s = t\}, R)$	if $\exists u.s \leftarrow_R u \to_R t$.
Simplify	$(E \cup \{s = t\}, R)$	$\vdash$	$(E \cup \{s = u\}, R)$	if $t \to_R u$.
Delete	$(E \cup \{t = t\}, R)$	$\vdash$	(E, R)	
Compose	$(E, R \cup \{s \to t\})$	$\vdash$	$(E, R \cup \{s \to u\})$	if $t \to_R u$.
Collapse	$(E, R \cup \{s \to t\})$	$\vdash$	$(E \cup \{u = t\}, R)$	if $\exists\{l \to r\} \in R$ s. t. $s \to_R u$ by $l \to r$, and $s \rhd l$

Where $\rhd$ is the specialisation ordering: $u \rhd v$ iff a subterm of u is an instance of v and not vice versa. We define a *state* to be the contents of E and R respectively between the application of two transition rules. In addition there is of course an initial and perhaps a terminal state.

In our setting we are interested in using the procedure to find a proof of given equations. This use of the completion procedure was first suggested by Huet in [17], where it was proved that the completion process yields as a semi-decision procedure granted that it does not fail on an equation not being orientable. When the proof we are looking for does not exist, however, completion might diverge. In order to use it in equational subproofs of proofs by structural induction we are therefore interested in restricting the process such that it terminates anyhow at a point when it is not likely that the proof we are looking for exists.

When we consider the rewrite rules that can be directly applied to a term t, we see that an instance of the left hand side of the rule must equal a subterm of t. A rewrite rule which does not fulfil this requirement could however still be utilised in a rewriting sequence of t at a later stage. To capture a set of rewrite rules which can not be used in *any* rewriting sequence for t, we use the fact that any terminating set of rewrite rules is contained within a monotone and well founded term ordering. From now on we shall only write term ordering, and implicitly assume that it is monotone and well founded.

DEFINITION 1. *If for a term ordering $\succ$ there exists no substitution σ such that $t \succeq l\sigma$ then t is said to be* incomparable to l *by $\succ$, or just incomparable if the ordering is obvious. If such a substitution exists, t is said to be* comparable to l.

It is convenient to introduce a notation for the case of s being comparable to t by $\succ$. For this purpose we shall use $s \overset{\succ}{\sim} t$.

LEMMA 1. *Let R be a set of rewrite rules that has a termination proof with the term ordering $\succ$. Let $l \to r$ be in R, and t be a term. If t is incomparable to l, then $l \to r$ can never be used in any rewriting sequence for t in R.*

PROOF. This is obvious from the fact that $\succ$ is monotone wrt. context application and substitutions, has the subterm property, and that any rewriting sequence from t will only contain terms that are less than t with respect to $\succ$. □

In KB-completion the term ordering by which the termination of the set of rewrite rules is proven, is input to the process. By means of lemma 1 we may therefore rule out the rewrite rules which do not contribute directly in the sense that they are able to rewrite the terms we are studying. Mark that we have not said that for a given term t there will only be finitely many rules that can possibly be of interest. Even if $\succ$ is expected not to give rise to any infinite decreasing sequence, it can for a given term give infinitely many finite decreasing sequences, and for many of the commonly used orderings this is indeed the case. Since we are striving for a finite method, we would like there to be only finitely many rules worth considering. Here we only assume that this is the case, and we return to this problem, and to decidability of comparability, later.

Let us take a look at the completion process. In order to present a modification of this process which will terminate in finite time, we first present this lemma on the behaviour of the set of rewrite rules during the process:

LEMMA 2. *If the set of rewrite rules R at a state in the completion process is able to rewrite the term t, the set R' in any future state will also be able to rewrite t.*

PROOF. The lemma states that the set of terms which can be rewritten by the set of rewrite rules in a completion process is monotonically increasing. The set of left hand sides in R is what decides which terms can be rewritten, thus the only transition rules which could defy the lemma is Collapse.

By studying the Collapse rule, we find that a rewrite rule $s \to t$ is removed from R whenever s can be rewritten by another rewrite rule $l \to r$. But then every term which could be rewritten by $s \to t$ can also be rewritten by $l \to r$, thus the set of reducible terms is not decreased. □

THEOREM 1. *A Completion procedure in which only finitely many rewrite rules can be added, and in which* Orient *is only used such that the left hand side of the new rewrite rule is irreducible, will terminate.*

PROOF. Since there are only finitely many rewrite rules that can be added, there can also be only finitely many distinct sets R in the procedure. Thus

there are also only finitely many sets of terms reducible by the different sets of rules.

If we orient an irreducible equation into the rewrite rule $l \to r$, obviously l was irreducible before the orientation, and is reducible afterwards, thus the set of reducible terms has increased. Since well foundedness of R is invariant, and R is finite, there can only be finitely many Simplify, Compose, Deduce, Collapse and Delete steps before one will have to apply Orient.

First, since no infinite Completion process can occur without an Orient step, and secondly the set of reducible terms can never decrease due to lemma 2 and there are only finitely many such sets possible, the completion process will terminate. □

At this point it seems convenient to give the transition rules for the modified completion procedure that has been indicated. We shall call it *Completion wrt. a set of terms*. This will be a process which takes as input a set E of equations, an ordering $\succ$, and a set T of terms, and manipulates E trying to create a set R of rules which is sufficiently strong to prove the equational equivalences that might exist between terms in T. The transition rules are the same as in standard completion, except for the Orient rule, which now looks like this:

$$(E \cup \{s = t\}, R) \quad \vdash \quad (E, R \cup \{s \to t\}) \qquad \begin{array}{l}\text{if } s \text{ irreducible and} \\ s \succ t \text{ and } \exists\, l \in T . l \overset{\succ}{\rightsquigarrow} s.\end{array}$$

LEMMA 3. *Completion wrt. a set of terms is sound with respect to the original E*

PROOF. Obvious. □

Having stopped the completion process before it has had a chance to create a convergent set of rewrite rules introduces another problem. We can no longer assume that each term in T has a unique normal form. Therefore we might have to generate the sets of all possible minimal forms of the two terms we are interested in, and investigate for a nonempty intersection.

DEFINITION 2. *A set R of rewrite-rules is said to be* sufficiently convergent for a term t *if t has only one irreducible form in R.*

Definition 2 indicates a method for determining sufficient convergence that appears rather complicated. The next lemma gives a way of checking for sufficient convergence which is sometimes more appropriate.

LEMMA 4. *Let completion wrt. the set T of terms have terminated with E and R. If every equation in E contains one term for which there is no term in T comparable to it by $\succ$, then R is sufficiently convergent for T.*

PROOF. We prove the lemma by showing that if one term in T has two distinct R-minimal forms, then there exists an equation in E in which both terms have a term in T comparable to it.

Assume that the term t in T has more than one irreducible form in R. Then let t' be the least term derivable from t which has two irreducible forms. The critical pair lemma of Knuth and Bendix [20] tells us that there must exist a critical pair between two rules that can rewrite t'. This critical pair must have been added to E in the completion process, but by the way we chose t', it cannot have been added as a rewrite rule to R.

Obviously, if we oriented this critical pair into a rewrite rule, the rule could participate in a rewrite sequence starting from t', thus also in one starting from t. Therefore t must be comparable to both terms in the equation. □

The checking of terms in E for possible orientation and comparability with the terms in T is inherent in the completion wrt. a set of terms procedure. Therefore the test indicated by lemma 4 is not a significant additional cost.

5. The orderings

We have now described a completion procedure for a system of equations wrt. the pair of terms we want to reason about. This process is guaranteed to terminate if there are only finitely many equations appearing during the procedure that can possibly be oriented into rewrite rules. Let us see what requirements termination of the process puts on the ordering.

LEMMA 5. *Let the ordering, $\succ$, that we put into the system contain the property that for a given term t we will only have finitely many l different up to variable renaming such that $t \succ l$. Then for any given finite set of terms T, our altered transition rule Orient will only be able to orient finitely many equations different up to variable renaming.*

PROOF. Let U be the set of terms for which there exists a term in T which is greater according to the ordering. Since there are only finitely many terms in T, there are also finitely many terms in U. Let V be the set of terms such that $v \in V$ iff $v\sigma \in U$ for some σ. Obviously V is finite, and contains all the left hand sides of rewrite rules possibly oriented by our transition rule.

Since there are only finitely many possible left hand sides, and since each right hand side has to be less than its corresponding left hand side, the requirement in the lemma guarantees that the number of possible rewrite rules is finite. □

Thus we must require our term ordering to satisfy that for every term there are only finitely many terms less than it. Even though this requirement gives the impression of being closely related to well foundedness, it is not a general property of the term orderings currently in use[1]. Any well-founded and transitive term ordering that does not meet the requirement can, however, be restricted in such a way that the requirement is met. There are many ways

[1] e.g. the path-orderings will satisfy $f(x) \succ g^n(x)$ for any n whenever f is considered greater than g. Orderings with polynomial interpretations, however, have the desired property.

to do this, but the one we suggest in this paper has been chosen basically because of two features; it is simple and intuitive, and it is flexible. The reason why flexibility is important is that completion wrt. a set of terms can be viewed as looking through a finite search-space for a proof of equivalence between two terms. If a solution is not found, we may want to be able to relax the restriction in order to increase the search space.

DEFINITION 3. *Finiteness Restriction (FR). Let $\succ$ be a term ordering and $k \geq 1$ be a constant. The finiteness restriction imposed on $\succ$ gives $\succ_{FR(k)}$, and is implemented by assigning a positive weight to every function symbol, and letting $s \succ_{FR(k)} t$ iff $s \succ t$ and $k \cdot sum_F(s) \geq sum_F(t)$. Here $sum_F(l)$ is the sum of the weights of all the function symbol occurrences in l.*

LEMMA 6. *The Finiteness restriction imposed on a term ordering gives a relation such that no term has infinitely many terms which are smaller than it with respect to the relation.*

PROOF. Since all function symbols have positive weights, this should be obvious. □

The tuning of the constant k is of course important to the performance of automatic proofs by structural induction, both for its ability to find proofs, and for computation time. This tuning will have to be done for each new term ordering, and in some cases also for each new set of definitions. Experience indicates, however, that when the weights of the functions are natural numbers from 1 and upwards, the method works at its best with $1.5 \leq k \leq 2$.

Imposing FR on an ordering will in general destroy transitivity, so the result cannot be called an ordering. But since the restricted relation is contained within an ordering which has the desired terminating properties, the (lack of) transitivity is not important. Mark, however, that for lemma 4 still to hold, we must state that the comparability check should be performed with the input ordering *without* FR.

In section 4 we defined what it means for one term to be comparable to another by an ordering. At that point we did not care to give any argument as to whether this is decidable or effectively implementable. The answer will of course depend very much on the ordering in question. One property of the orderings in general is, however, stated in this theorem:

THEOREM 2. *Let $\succ$ be a monotonic ordering satisfying the subterm property. Then the term t is comparable to s by $\succ$ iff there exists a substitution γ such that $t \succeq s\gamma$, and where the image of each variable is either a constant or a variable.*

PROOF. If such a γ exists, then of course this is a witness of t being comparable to s. The other way there exists a substitution σ such that $t \succeq s\sigma$. Let σ be $\{x_1 \mapsto t_1, x_2 \mapsto t_2, \ldots, x_n \mapsto t_n\}$ Because of the subterm property, for every i, $1 \leq i \leq n$, there will either be a variable v such that

$t_i \succeq v$ or a constant f such that $t_i \succeq f$. Let c_i be that variable or constant, and let γ be $\{x_i \mapsto c_i | i = 1, \ldots, n\}$. By context application $s\sigma \succeq s\gamma$, and by transitivity $t \succeq s\gamma$. □

COROLLARY. *Comparability is decidable for monotonic and well founded term orderings.*

The corollary holds because there will only be finitely many constants in the theory, and secondly because we only have to consider the variables that occur in t, as the next easy lemma shows:

LEMMA 7. *Let $\succ$ be a monotonic well founded term ordering. For $t \succ s$ it is a requirement that all variables in s are also in t.*

PROOF. Let $t \succ s$ and let s contain the variable x which is not in t. Furthermore, let σ be the substitution $\{x \mapsto t\}$. According to monotonicity $t\sigma \succ s\sigma$ which gives $t \succ s\sigma$. Obviously t is a subterm of $s\sigma$, thus by repeatedly replacing the subterm t by $s\sigma$ we get an infinite chain which is $\succ$-decreasing due to monotonicity over context application. □

In theorem 2 we assume that the ordering is transitive. When we impose FR on the ordering, transitivity can no longer be taken for granted. Therefore we shall now deal with comparability in the relations resulting from the finiteness restriction.

THEOREM 3. *Let $\succ$ be transitive, monotonic with respect to context application and have the subterm property. Then the term t is comparable to s by $\succ_{FR(k)}$ iff there exists a substitution γ such that $t \succeq_{FR(k)} s\gamma$, and which only substitutes by variables and constants.*

PROOF. As in the proof above, the existence of such a σ is a witness for comparability. Also as above we can from a given σ construct a γ with only variable and constant substitutions such that $t \succeq_{FR(k)} s\sigma \succeq s\gamma$. By transitivity of $\succ$ we get that $t \succeq s\gamma$, and because we obviously have $sum_F(s\sigma) \geq sum_F(s\gamma)$ we will also have $t \succeq_{FR(k)} s\gamma$. □

An equivalent corollary to that of theorem 2 is valid for theorem 3.

6. Equational validity by refutation

Completion used as an equational theorem prover has been elaborated by several authors [1, 16, 17] We shall now study the method of Bachmair, and extend our completion wrt. a set of terms into his method.

Knowing that equational validity has a semi decision procedure, the possibility of having a failing completion procedure is rather annoying. This led to the wish of also being able to handle unorientable equations in both

the completion process and in term rewriting. For example when we study the commutativity axiom

$$f(x,y) = f(y,x)$$

we see that even if the axiom is not orientable by itself, many of its instances are e.g. by the lexicographical path ordering:

$$f(g(x),x) \succ_{lpo} f(x,g(x))$$

Having this in mind, we see that we could still be able to use the equation in a terminating rewriting sequence, by for each application of the equation making sure that the resulting term is smaller in the term ordering than the original one. We shall write $\rightarrow_E^{\succ}$ to indicate the rewriting relation given by the set E of equations together with the term ordering $\succ$, and $\leftrightarrow_E$ to denote an arbitrary equational proof step by an equation in E. As previously an asterisk (e.g. $\rightarrow_E^{\succ *}$) is used to denote possibly empty sequences of steps.

The above caters for the simplification steps in the completion process. The part concerning the computation of critical pairs will also need a bit of care. In standard completion critical pairs are only created between oriented rules, because it is implicitly assumed that all equations are orientable. Since we shall extend the procedure to also be able to handle unorientable equations, we now have to compute critical pairs also between equations. The lines between rewrite rules and equations have now weakened, thus we may handle the completion process called *ordered completion* as if it only treated equations. These are the transition rules:

Deduce	(E)	$\vdash$	$(E \cup \{s = t\})$	if $\exists u.s \leftrightarrow_E u \leftrightarrow_E t$ such that $s \not\succeq u$ and $t \not\succeq u$
Simplify	$(E \cup \{s = t\})$	$\vdash$	$(E \cup \{s = u\})$	if $t \rightarrow_E^{\succ} u$ by $s' = u' \in E$ with $t \succ s$ and $t \rhd s'$
Delete	$(E \cup \{t = t\})$	$\vdash$	(E)	

Ordered completion has been described and elaborated in [1, 16] and [2]. Our version of the procedure is a slight simplification of the one in [1]. It is worth pointing out that the critical pairs considered by the *deduce* rule are only those stemming from two adjacent *decreasing* equational rewriting steps. Such critical pairs shall in the following be called *feasible*. We shall not need the full explanation of this procedure in our approach, so for details we simply refer to the mentioned papers. We are however interested in the following theorems:

THEOREM 4. *(Bachmair) Let t and u be two ground terms such that $t =_E u$, and let $\succ$ be an ordering which can be extended to be total on ground terms. Then fair ordered completion equipped with E and $\succ$ will within finite time create an E' such that $t \rightarrow_{E'}^{\succ *} s \leftarrow_{E'}^{\succ *} u$ for some s.*

THEOREM 5. *(Bachmair) Let s' and t' be skolemized versions of s and t. Furthermore let E' be $E \cup \{eq(x,x) = True,\ eq(s',t') = False\}$ such that the functions eq, $True$, $False$ and the skolem constants are not in E. Also let $\succ$ be an ordering total on ground terms such that both $True$ and $False$ have no term less than it. Then $s =_E t$ if and only if ordered completion with E' and $\succ$ creates the equation $True = False$.*

PROOF. (Sketch) Obviously $s =_E t$ iff $s' =_E t'$. Therefore $True =_{E'} False$ iff $s =_E t$ because the functions eq, $True$ and $False$ are not referred to in E. The reverse direction follows immediately. For the forward direction assume $s' =_E t'$. Ordered completion will by virtue of theorem 4 create an equational rewrite proof between $True$ and $False$ within finite time. Since these two terms are smaller than all the other ground terms, this proof will have to consist of only one application of the equation $True = False$. □

By virtue of theorem 5 ordered completion can be used as a semi decision process for the equational validity problem.

7. Ordered completion wrt. a set of terms

The motivation for wanting to alter the ordered completion procedure is the same as for standard completion. Even if ordered completion can be used as a semi decision procedure for equational validity, it will most likely diverge whenever the equational proof one is looking for does not exist.

In order to impose the same idea as we have presented for standard completion in this case, there are two problems to be solved. The first is that when using ordered completion as a theorem prover, we are not primarily looking for a rewrite proof between two terms. On the contrary what triggers the success of the procedure is the existence of the equation $True = False$. It is therefore not obvious how to find reasonable terms by which one can limit the search space. The second problem is to make sure that the limited search space is really finite.

We shall discuss the problem with the terms first. By studying the refutational theorem prover described in the last section, we see that it is based on the transformation of a proof between ground terms in a set $E' = E \cup \{eq(x,x) = True,\ eq(s',t') = False\}$ of equations of the form

$$True \leftrightarrow_{E'} eq(t,t) \leftrightarrow_E, \ldots, \leftrightarrow_E eq(s',t') \leftrightarrow_{E'} False$$

into a proof in the set E'' of the form

$$True \leftrightarrow_{E''} False$$

where $s' = t'$ is the equation we are trying to prove. We can further see that in the part of the original proof which is marked by dots, all the reasoning will be on the two immediate subterms of the term on the form $eq(x,y)$. This because E does not refer to the function eq.

From the above it is likely that the "biggest" terms which we will have to handle by the equations generated by ordered completion is of the same order of size as $eq(s', t')$. This term might, however, be twice as big as any of the terms s' and t', which are the terms involved in the equation we are proving.

By studying theorem 4, however, we find that we can simplify the ordered completion/proof by refutation procedure. After all, s' and t' are skolemized, and thus ground. Therefore we can remove from the method the two equations defining eq, and the method reduces to equational rewriting of s' and t' in the set of equations found by ordered completion. Theorem 4 guarantees that this method is also a semi decision procedure.

Now for the problem of limiting the search space. In standard completion wrt. a set of terms we were able to limit the number of critical pairs to be considered by refusing to orient equations into rewrite rules which could not rewrite any of the terms we were studying. Here critical pairs are computed between equations, so this approach will not be applicable.

We must in other words impose our restriction directly on the transition rule for computing critical pairs. The transition rule Deduce will then look like this:

$$\textit{Deduce} \qquad (E) \quad \vdash \quad (E \cup \{s = t\}) \qquad \begin{array}{l}\text{if } \exists u.s \leftrightarrow_E u \leftrightarrow_E t \\ \text{such that } s \not\succeq u \text{ and t} \not\succeq u \text{ and} \\ \exists\, l \in T.l \overset{\succ}{\rightsquigarrow} s \wedge l \overset{\succ}{\rightsquigarrow} t\end{array}$$

In words; there should be a term in T which is comparable to both terms in the new equation. *Delete* and *Simplify* are left unchanged.

LEMMA 8. *If at any ordered completion state, the set of equations E in the process is able to rewrite the term t, the set E' in any future state will also be able to rewrite t.*

PROOF. A trivial extension of the proof of lemma 2. □

THEOREM 6. *Let an Ordered Completion Procedure be restricted such that the Deduce transition step is only performed between equations which are irreducible. If there are only finitely many equations that can be added, the procedure will terminate.*

PROOF. We shall first show that if a critical pair between two equations e and e' has been computed, one will never have produced two equations identical to e and e' later so that the same critical pairs will have to be recomputed. The only way we could have another equation identical to e is by adding it to E by the standard set union operation. But then of course e itself must have disappeared from the set. The only way e can have disappeared is by it being reduced by another equation. By virtue of lemma 8 because it is identical to e, the new equation will also be reducible,

thus by the assumption that Deduce is only applied to irreducible equations, the new equation will not have its critical pairs computed.

Thus the above gives us that no critical pair is computed twice. Since there are only finitely many possible equations, there are only finitely many critical pairs between them.

Obviously we cannot have an infinite completion sequence with only a finite number of Deduce steps, thus the procedure must terminate. □

The above proof leans on the assumption that the procedure keeps track of which equations it has computed critical pairs from in order not to do the same operation again. This does not follow from the transition rules, but is a natural requirement on any implementation.

We have an analogue to lemma 4 for ordered completion wrt. a set of terms which is even stronger, because it does not introduce an additional test. Sufficient convergence for equations is here the obvious extension of that of rewrite rules.

LEMMA 9. *Let ordered completion wrt. the set T of skolemized (and therefore ground) terms, and with the term ordering $\succ$ which is total on ground terms, have terminated with E. Then E is sufficiently convergent for T.*

PROOF. Let E' be the set of critical pairs which has been disregarded as the result of the additional comparability test in *deduce*.

Assume that the term t in T has more than one irreducible form in E. Then let t' be the least term derivable from t which has two irreducible forms. The extended critical pair lemma of Bachmair [1] tells us that there must exist a feasible critical pair between two equations in E that can rewrite t'. Because $\succ$ is total on ground terms we know that by the way we chose t', the critical pair cannot have been added to E thus it must be in E'. But obviously t' is comparable to both terms in this critical pair, thus so is t. So the critical pair cannot be in E' either, and we have contradiction. □

By the alternative transition rule we have presented, we are guaranteed to have only finitely many equations to consider granted that the term ordering has the property of no term having infinitely many terms less than it. This transition rule for Deduce therefore gives us a heuristic for ruling out equations which do not seem to contribute to the proof. We have further proved that this heuristic gives us a terminating ordered completion procedure.

8. Computational experiments

We shall in this section present some experiments on how our heuristics perform. Since our aim is automatizing more than finding new and exotic proofs, we have deliberately chosen the examples among the most common ones in the literature. The examples are generated by a prototype implementation written in SIMULA. This implementation basically consists of the completion based methods, whereas the induction steps had to be performed by hand.

EXAMPLE 1. Consider the following simple theory for natural numbers which is written in ABEL-like syntax [5]:

TYPE NAT ==
FUNCTIONS:
0: $\longrightarrow$ NAT;
s: NAT $\longrightarrow$ NAT;
add: NAT $\times$ NAT $\longrightarrow$ NAT;
CONSTRUCTORS: 0, s
DEFINITIONS:
$add(x, 0) = x$
$add(x, s(y)) = s(add(x, y))$
END

We wanted to study how our heuristics behaved for the example of associativity of addition. We therefore started the procedure for proofs by structural induction as described with the following singleton set P.

$$add(add(x, y), z) \quad =_{\mathcal{I}(DE)} \quad add(x, add(y, z))$$

Since DE here consists of the two equations in the type definition, the above equation obviously has no equational proof in DE. We therefore used the induction scheme on the variable z which transformed P into

$$\begin{aligned} add(add(x, y), 0) &=_{\mathcal{I}(DE_0)} add(x, add(y, 0)) \\ add(add(x, y), s(Z)) &=_{\mathcal{I}(DE_s)} add(x, add(y, s(Z))) \end{aligned}$$

Here $DE_0 = DE$, and $DE_s = DE \cup \{add(add(x, y), Z) = add(x, add(y, Z))\}$. The capital Z is used to denote the new constant introduced by the induction scheme.

We then picked out the first equation in P, which is easily proven in DE by rewriting. The second equation we tried to prove by standard completion wrt. a set of terms. This implied completing $DE \cup \{add(add(x, y), Z) = add(x, add(y, Z))\}$ wrt. the terms $add(add(x, y), s(Z))$ and $add(x, add(y, s(Z)))$. We used lexicographic path ordering with the precedence $add > s > 0$ on the function symbols. The finiteness restriction was imposed on the ordering, with the constant $k = 1.7$.

Completion terminated with the following not complete set of rewrite rules.

$$\begin{aligned} add(x', add(0, add(Z, Z))) &\rightarrow add(x', add(Z, Z)) \\ add(x, add(0, Z)) &\rightarrow add(x, Z) \\ add(add(x, y), Z) &\rightarrow add(x, add(y, Z)) \\ add(x, 0) &\rightarrow x \\ add(x, s(y)) &\rightarrow s(add(x, y)) \end{aligned}$$

The above set is sufficiently convergent for the two terms we are considering. Furthermore it is strong enough to prove the equation.

The above is the obvious induction example, where the correct induction variable is considered immediately. If a computer was to pick the induction variable, it would in general not be as clever. If we for example tried to perform the above proof by doing induction first on x then on y and finally on z, the structural induction strategy would still succeed. The set P would be expanded into 8 equations, which could all be proved either by rewriting in DE or by standard completion wrt. a set of terms.

On the way, however it would be looking for several equational proofs which do not exist. It will e.g. be trying to prove

$$add(add(s(X), y), z) =_{\mathcal{I}(DE_s)} add(s(X), add(y, z))$$

where $DE_s = DE \cup \{add(add(X, y), z) = add(X, add(y, z))\}$. Doing this by standard completion wrt. a set of terms, the procedure will after some time give up with the following not complete set of rules:

$$\begin{aligned}
add(X, add(s(s(s(0))), z)) &\rightarrow add(s(s(s(X))), z) \\
add(s(s(s(add(X, y')))), z) &\rightarrow add(X, add(s(s(s(y'))), z)) \\
add(X, add(s(s(0)), z)) &\rightarrow add(s(s(X)), z) \\
add(s(s(add(X, y))), z) &\rightarrow add(X, add(s(s(y)), z)) \\
add(X, add(s(add(0, z')), z)) &\rightarrow add(X, add(s(z'), z)) \\
add(X, add(s(0), z)) &\rightarrow add(s(X), z) \\
add(X, add(add(0, z), z')) &\rightarrow add(X, add(z, z')) \\
add(s(add(X, y')), z) &\rightarrow add(X, add(s(y'), z)) \\
add(X, add(0, z)) &\rightarrow add(X, z) \\
add(add(X, y), z) &\rightarrow add(X, add(y, z)) \\
add(x, s(y)) &\rightarrow s(add(x, y)) \\
add(x, 0) &\rightarrow x
\end{aligned}$$

The 12 rewrite rules above were generated in 4-5 seconds on a SUN 3/60, which indicates that the search space is small enough for the procedure to be of practical interest.

Mark that the above proof could still be performed, even if we several times chose the "wrong" induction variable. These choices only meant that we had to look for some equational proofs which were not there. Because we have a heuristic stopping any divergent completion process, all these "blind alleys" terminated.

EXAMPLE 2. For demonstrating ordered completion wrt. a set of terms, we tried to prove an equally common property of addition, namely commutativity. Here we started with our P consisting of only one equation:

$$add(x, y) \quad =_{\mathcal{I}(DE)} \quad add(y, x)$$

For this equation there is no equational proof, thus we had to perform an induction step, giving us:

$$\begin{array}{rcl} add(0,y) & =_{\mathcal{I}(DE_0)} & add(y,0) \\ add(s(X),y) & =_{\mathcal{I}(DE_s)} & add(y,s(X)) \end{array}$$

In the same manner as previously we get $DE_0 = DE$ and $DE_s = DE \cup \{add(X,y) = add(y,X)\}$. We wanted to prove the second of these equations by ordered completion wrt. a set of terms, therefore we had to add the following induction hypothesis to DE_s:

$$add(X,y) \;=\; add(y,X)$$

Remember that the uppercase symbols X, Y and Z denote constants. In this case constants are either generated by the induction schema, or they are skolem constants. Ordered completion was then started wrt. the skolemized terms $add(s(X),Y)$ and $add(Y,s(X))$, and the process terminated with the following set :

$$\begin{array}{rcl} s(add(y,X)) & = & s(add(X,y)) \\ s(add(y,X)) & = & add(s(y),X) \\ add(s(y),X) & = & s(add(X,y)) \\ add(0,X) & = & X \\ add(X,y) & = & add(y,X) \\ add(x,0) & = & x \\ add(x,s(y)) & = & s(add(x,y)) \end{array}$$

This set of equations is not able to prove $add(X,s(Y)) = add(s(Y),X)$, thus we needed another level of induction. If left unrestricted, the algorithm would not have terminated. With our heuristics it gave up in the state described above after a couple of seconds, with the constant $k = 1.5$.

We then performed an induction step on both the equations in P, transforming P into

$$\begin{array}{rcl} add(0,0) & =_{\mathcal{I}(DE_{00})} & add(0,0) \\ add(0,s(Y)) & =_{\mathcal{I}(DE_{s0})} & add(s(Y),0) \\ add(s(X),0) & =_{\mathcal{I}(DE_{0s})} & add(0,s(X)) \\ add(s(X),s(Y)) & =_{\mathcal{I}(DE_{ss})} & add(s(Y),s(X)) \end{array}$$

Here $DE_{00} = DE$, $DE_{s0} = DE \cup \{add(0,Y) = add(Y,0)\}$, $DE_{0s} = DE \cup \{add(X,y) = add(y,X)\}$ and $DE_{ss} = DE \cup \{add(X,y) = add(y,X), add(s(X),Y) = add(Y,s(X))\}$. We wanted to prove last of these equations by ordered completion, thus we had to add the following set of equations to DE_{ss} for the duration of the subproof:

$$\begin{array}{rcl} add(X,y) & = & add(y,X) \\ add(s(X),Y) & = & add(Y,s(X)) \end{array}$$

After having generated the following equations

$$\begin{aligned} s(add(y,X)) &= s(add(X,y)) \\ s(add(y,X)) &= add(s(y),X) \\ add(s(y),X) &= s(add(X,y)) \\ add(0,X) &= X \\ add(s(X),Y) &= s(add(Y,X)) \\ add(X,y) &= add(y,X) \\ add(x,0) &= x \\ add(x,s(y)) &= s(add(x,y)) \end{aligned}$$

the system was able to prove $add(s(X),s(Y)) = add(s(Y),s(X))$ by equational rewriting. The rest of the nontrivial equations in P are easily proven the same way.

EXAMPLE 3. Now let us focus on a variant of the formula for the sum of the n first natural numbers

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

This require that we extended the previous definition of the natural numbers by the two functions *mult* and *sum*, defined like this:

$$\begin{aligned} mult(x,0) &= 0 \\ mult(x,s(y)) &= add(mult(x,y),x) \\ sum(0) &= 0 \\ sum(s(x)) &= add(s(x),sum(x)) \end{aligned}$$

The equation we proved was

$$add(sum(x),sum(x)) = mult(x,s(x))$$

In order to do this we needed three lemmas, namely associativity and commutativity of addition, which we already had proven, and the following property of *mult* which may be proven the same way using associativity and commutativity of addition as lemmas:

$$mult(s(x),y) = add(y,mult(x,y))$$

We denote by DE^* the previous DE extended with the new definitions and the lemmas.

The equation was not immediately provable, thus we had to perform an induction step on the only variable x, giving the following proof obligations:

$$\begin{aligned} add(sum(0),sum(0)) &=_{\mathcal{I}(DE_0^*)} mult(0,s(0)) \\ add(sum(s(X)),sum(s(X))) &=_{\mathcal{I}(DE_s^*)} mult(s(X),s(s(X))) \end{aligned}$$

Here $DE_0^* = DE^*$ and $DE_s^* = DE^* \cup \{add(sum(X), sum(X)) = add(X, mult(X, X))\}$.

The first of these equations was easily proven by rewriting. For the second we tried ordered completion wrt. the set $\{add(sum(s(X)), sum(s(X))), mult(s(X), s(s(X)))\}$ of terms. After the process had generated the following set of equations, the last equation was also provable by equational rewriting.

$$\begin{aligned}
add(s(y), x) &= s(add(x, y)) \\
add(0, x) &= x \\
add(z, add(y, x)) &= add(x, add(y, z)) \\
add(x, add(z, y)) &= add(z, add(x, y)) \\
add(sum(X), sum(X)) &= add(X, mult(X, X)) \\
mult(x, s(y)) &= add(x, mult(x, y)) \\
mult(s(x), y) &= add(y, mult(x, y)) \\
add(x, y) &= add(y, x) \\
add(x, add(y, z)) &= add(add(x, y), z) \\
add(x, s(y)) &= s(add(x, y)) \\
sum(s(x)) &= add(s(x), sum(x)) \\
add(x, 0) &= x \\
mult(x, 0) &= 0 \\
sum(0) &= 0
\end{aligned}$$

Mark that this proof would have been significantly simpler if we had used AC (associative and commutative) completion and rewriting, and our method extends trivially to AC theories. The proof was still performable by only using AC-lemmas, and thus illustrates the strength of completion based methods in proofs by structural induction when there are many lemmas destroying the initial convergence in DE.

9. Discussion

We have in this paper seen how one, by a slight change in two different completion processes, can make sure that completion will terminate when one knows what terms one actually wants to rewrite in the system. The method is meant for use in the equational part of proofs by structural induction. The price the method has to pay for its guarantee of termination is that it might terminate with fail, when a continuation of the process would lead to a convergent set of rules. The benefit of the process is that it enables us to perform proofs by structural induction in a completely automatic manner.

Whenever a method has been developed it is good science to include in the presentation examples where the method did *not* succeed in order to indicate the limit of the work. Here such an example would be a proof by

structural induction which does exist, but which the method was not able to find. Since the method in principle can simulate any inductive proof-tree, and we base the method on a semi-decision procedure for equational proofs, such examples are not apparent in this case. For any such proof it is most often simply a question of tuning the constant k properly. The problem of tuning this constant is perhaps the most critical part of the method. We have indicated that values between 1.5 and 2 seems to work fine for most examples, but we need more knowledge on the tuning of k in order to be able to handle really complex proofs in reasonable time.

Still we think that the strength of the method is not not in finding big and complex proofs. It is our experience that when proving the correctness of programs, the real difficulty lies in the vast mass of trivial proofs that have to be performed. An automatization of structural induction for this purpose should therefore rather aim at being able to handle simple proofs completely automatically than at being able to find really big and complex proofs.

References

[1] L. Bachmair. *Proof methods for equational theories*. PhD thesis, University of Illinois, Urbana-Champaign, 1987.

[2] L. Bachmair. *Canonical equational proofs*. Computer Science Logic, Progress in Theoretical Computer Science. Birkhäuser Verlag AG, 1991.

[3] R. S. Boyer and J. S. Moore. *A Computational Logic Handbook*. Academic Press, Inc., 1988.

[4] R. Burstall. Proving properties of programs by structural induction. *Computer Journal*, 12(1):41–48, 1969.

[5] O.-J. Dahl, D. F. Langmyhr, and O. Owe. Preliminary report on the specification and programming language ABEL. Research Report 106, Department of informatics, University of Oslo, Norway, 1986.

[6] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specifications I, Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1985.

[7] U. Fraus and H. Hussmann. A narrowing-based theorem prover. Distributed at 2nd International Conference on Algebraic and Logic Programming, Nancy (France), October 1990.

[8] S. J. Garland and J. V. Guttag. An overwiew of LP, the Larch Prover. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (North Carolina, USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 137–151. Springer-Verlag, April 1989.

[9] S. J. Garland and J. V. Guttag. Inductive methods for reasoning about abstract data types. In *Proceedings of the fifteenth annual ACM Symposium on Principles of Programming Languages*, pages 219–228, January 1988.

[10] J. A. Goguen. How to prove inductive hypotheses without induction. In W. Bibel and R. Kowalski, editors, *Proceedings of the 5th Conference on Automated Deduction*, volume 87 of *Lecture Notes in Computer Science*, pages 356–373. Springer-Verlag, 1980.

[11] J. A. Goguen. OBJ as a theorem prover with applications to hardware verification. Technical report, SRI International,Computer Science Lab, 1988.

[12] J. A. Goguen and T. Winkler. Introducing OBJ3. Technical report, SRI International,Computer Science Lab, 1988.

[13] J. V. Guttag. *The Specification and Application to Programming of Abstract Data*

Types. PhD thesis, Computer Science Department, University of Toronto, 1975.

[14] J. V. Guttag and J. J. Horning. The algebraic specification of abstract data types. *Acta Informatica*, 10:27–52, 1978.

[15] J. V. Guttag, J. J. Horning, and J. M.Wing. Larch in five easy pieces. Technical report, Digital Systems Research Center, 1985.

[16] J. Hsiang and M. Rusinowitch. On word problems in equational theories. In T. Ottmann, editor, *Proceedings 14th International Colloquium on Automata, Languages and Programming, Karlsruhe (Germany)*, volume 267 of *Lecture Notes in Computer Science*, pages 54–71. Springer-Verlag, July 1987.

[17] G. Huet. A complete proof of correctness of the Knuth and Bendix completion algorithm. *Journal of Computer and System Sciences*, 23(1):11–21, 1981.

[18] G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, pages 239–266, 1982.

[19] J.-P. Jouannaud and E. Kounalis. Automatic proofs by induction in theories without constructors. *Information and Computation*, 82(1):1–33, July 1989.

[20] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, 1970.

[21] O. Lysne. Proof by consistency in constructive systems with final algebra semantics. In *Proceedings 3rd International Conference on Algebraic and Logic Programming, Pisa (Italy)*, volume 632 of *Lecture Notes in Computer Science*, pages 276–290. Springer-Verlag, 1992.

[22] O. Lysne. The equational part of proofs by structural induction. *BIT*, 33:596–618, 1993.

[23] O. Lysne. On the connection between narrowing and proof by consistency. Accepted to the 12th Conference on Automated Deduction, Nancy (France), 1994. Proceedings to be published by Springer Verlag in the series Lecture Notes in Computer Science.

[24] 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.