New Uses of Linear Arithmetic in Automated Theorem Proving by Induction [*]

Deepak Kapur and M. Subramaniam
Computer Science Department, State University of New York, Albany, NY 12222, U.S.A.
email: kapur@cs.albany.edu, subu@cs.albany.edu

Abstract. Zhang, Kapur and Krishnamoorthy introduced a cover set method for designing induction schemes for automating proofs by induction from specifications expressed as equations and conditional equations. This method has been implemented in the theorem prover *Rewrite Rule Laboratory* (*RRL*) and a proof management system *Tecton* built on top of *RRL,* and it has been used to prove many nontrivial theorems and reason about sequential as well as parallel programs. The cover set method is based on the assumption that a function symbol is defined using a finite set of terminating (conditional or unconditional) rewrite rules. The termination ordering employed in orienting the rules is used to perform proofs by well-founded induction. The left side of the rules are used to design different cases of an induction scheme, and recursive calls to the function made in the right side can be used to design appropriate instantiations for generating induction hypotheses. A weakness of this method is that it relies on syntactic unification for generating an induction scheme for a conjecture. This paper goes a step further by proposing semantic analysis for generating an induction scheme for a conjecture from a cover set. We discuss the use of a decision procedure for Presburger arithmetic (quantifier-free theory of numbers with the addition operation and relational predicates $>, <, \neq, =, \geq, \leq$) for performing semantic analysis about numbers. The decision procedure is used to generate appropriate induction schemes for a conjecture using cover sets of function taking numbers as arguments. This extension of the cover set method automates proofs of many theorems which otherwise, require human guidance and hints. The effectiveness of the method is demonstrated using some examples which commonly arise in reasoning about specifications and programs. It is also shown how semantic analysis using a Presburger arithmetic decision procedure can be used for checking the completeness of a cover set of a function defined using operations such as $+$ and $-$ on numbers. Using this check, many function definitions used in a proof of the prime factorization theorem stating that every number can be factored uniquely into prime factors, which had to be checked manually, can now be checked automatically in *RRL.* The use of the decision procedure for guiding generalization for generating conjectures and merging induction schemes is also illustrated.

Key Words: Induction, Automated Theorem Proving, Heuristics, Linear Arithmetic, Presburger Arithmetic, Generalization, Semantic Unification.

1. Introduction

While mechanizing proofs by induction, a crucial problem that often needs to be addressed is to decide on an induction scheme that leads to an appropriate induction hypothesis (or a set of induction hypotheses) to carry out the proof. This is related to an appropriate choice of a variable (or a finite set of variables) in a conjecture to perform induction on. In their seminal work, Boyer and Moore proposed designing induction schemes based on terminating function definitions. Typically

[*] Partially supported by the National Science Foundation Grant no. CCR-9303394 and subcontract CB0249 of SRI contract MDA904-92-C-5186 with The Maryland Procurement Office.

the definitions of function symbols appearing in a conjecture are considered as candidates for designing induction schemes. The soundness of an induction scheme so designed is based on a well-founded ordering used to show the termination of the function definition. This insight often results in generation of appropriate induction hypotheses which in many cases, can be used to prove (or disprove) a conjecture. This approach is supported in Boyer and Moore's theorem prover along with other heuristics [1, 2], and has been successfully used to prove many nontrivial theorems as well as reason about hardware and software.

Inspired by Boyer and Moore's work, Zhang, Kapur and Krishnamoorthy [13, 12] introduced a cover set method for designing induction schemes for automating proofs by induction from equations. This method has been implemented in the theorem prover *Rewrite Rule Laboratory* (*RRL*) [10] and a proof management system *Tecton* built on top of *RRL* [7], and has been used to prove many nontrivial theorems and reason about sequential as well as parallel programs. The cover set method is based on the assumption that a function symbol is defined using a finite set of terminating (conditional or unconditional) rewrite rules. The termination ordering employed in orienting the rules is used to perform proofs by well-founded induction. The left sides of the rules are used to design different cases of an induction scheme, and recursive calls to the function made in the right side can be used to design appropriate instantiations for generating induction hypotheses. Since the cover set method is based on a well-founded ordering used to define a function possibly specified using nonconstructors, an induction scheme generated by this method is a generalization of the principle of mathematical induction for natural numbers and the principle of structural induction obtained from the constructors of a data structure.

In this paper, we go a step further. It is the case that many data structures can be represented in many different ways; for example, numbers can be represented using 0 and s (to stand for the *successor* function) as constructors as well as in terms of $0, 1$ and $+$. Some functions, such as $gcd, divides, rem$, etc., are more convenient and easier to define using $+$ instead of s. Finite sets can be represented using the *null* set, a constructor that *inserts* an element, or alternatively, using the null set, *singleton* sets and the *union* operation. Similarly, lists can be represented using the *empty* list, and *cons* constructor, or alternatively, the empty list, *singleton* lists and *append* operator on lists. While reasoning about functions defined on such data structures, there is a need to convert from one representation to the other.

We exhibit the use of semantic information about a data structure for reconciling the use of different representations while attempting a proof of a conjecture. In this paper, we focus on generating an induction scheme using semantic information, that enables the application of induction hypotheses which otherwise cannot be applied because of different uses of a function. We illustrate this idea with an example using a decision procedure for Presburger arithmetic (the quantifier-free theory of numbers with the addition operation and relational predicates $>, <, \neq, =, \geq, \leq$, also called the *linear arithmetic*) for performing semantic analysis.

First we provide an informal review of the cover set method in mechanizing induction using a simple example. Then we use another related example to discuss the use of the linear arithmetic procedure for performing semantic analysis for reconciling two different uses of a function in a conjecture.

Consider the following definition of the divisibility predicate on natural numbers; the functions $0, s, <, +$, etc. have the usual meaning.

1. $divides(u, 0) \;=\; true$, 2. $divides(u, v) \;=\; false \; if \; (v < u) \;\wedge\; (v \neq 0)$,
3. $divides(u, u + v) \;=\; divides(u, v)$.

The above equations can be oriented left to right and the resulting rewrite rules are terminating.[1] The above defines *divides* whenever the first argument is non-zero or both the arguments are 0.

A possible conjecture to prove is:

$$(P_1): \quad divides(x, y + y) \; if \; divides(x, y) \wedge x > 0.^2$$

The reader can verify that if this conjecture is attempted using the principle of mathematical induction, the proof gets quite complicated. However using the cover set method, from the definition of $divides(x, y)$ two base cases corresponding to the first two rewrite rules are generated, and the induction step is generated using the third rewrite rule. The cover set based on the definition of *divides* is:

$$(\mathcal{C}_1): \; \{\langle\langle u, 0\rangle, \{\}, \{\}\rangle, \langle\langle u, v\rangle, \{\}, \{v < u, v \neq 0\}\rangle, \langle\langle u, u + v\rangle, \{\langle u, v\rangle\}, \{\}\rangle\}.$$

For every rule in the definition, there is a triple in the cover set, whose first component is the tuple corresponding to the arguments of *divides* in the left side, the second component is a finite set consisting of the arguments of all the recursive calls to *divides*, if any, in the right side of the rule, and the third component is the finite set of conditions from the literals in the condition, if any, of the rule. A triple could have empty second and third components.

The induction scheme generated from $divides(x, y)$ at position 2.2.1[3] for the above conjecture based on the cover set of *divides* is:

[1] See however a remark later. If semantic information such as linear arithmetic is used for rewriting as well, then rule 3 is not terminating, for example, when u is 0. The termination condition required is $u + v > v$.

[2] Since *divides* is not defined when its first argument is 0 and its second argument is non-zero, the above conjecture is not true if we drop the condition $x > 0$ from it. The completeness of the definition of *divides* is discussed in detail in section 4.

[3] A position is a sequence of nonnegative integers used to refer to a subterm in a term. An equation will be considered as a term with = as the binary predicate; a conditional equation will be considered as a term with = as the binary predicate whose second argument is an *if* term, where *if* is a considered as a binary function. In the above example, the position of $divides(x, y)$ is 2.2.1 as the conjecture is viewed as an abbreviation for

$$divides(x, y + y) = true \; if \; divides(x, y) \wedge x > 0.$$

$$\begin{aligned}(I_1):\ \{&\langle\langle\{x \rightarrow u, y \rightarrow 0\}, \{\}, \{2.2.1 \leftarrow divides(u,0)\}\rangle, \{\}\rangle,\\ &\langle\langle\{x \rightarrow u, y \rightarrow v\}, \{v < u, v \neq 0\}, \{2.2.1 \leftarrow divides(u,v)\}\rangle, \{\}\rangle,\\ &\langle\langle\{x \rightarrow u, y \rightarrow u+v\}, \{\}, \{2.2.1 \leftarrow divides(u,u+v)\}\rangle,\\ &\qquad \{\langle\{x \rightarrow u, y \rightarrow v\}, \{\}, \{2.2.1 \leftarrow divides(u,v)\}\rangle\}\rangle\}.\end{aligned}$$

An induction scheme is a finite set of tuples, each tuple corresponding to an induction case or subgoal.

The first component of the tuple is used to generate the conclusion in the induction subgoal; the second component is used to generate the induction hypotheses, if any. The first component of the induction case is a 3-tuple, whose first component is a substitution to be made on the conjecture, the second component is a finite set of the conditions for which the induction case is applicable, and the third component is how the subterm being used for generating the induction scheme must be replaced so that the rule of the definition from which the induction case is generated, can be applied. The second component of the induction case is a finite set of 3-tuples for generating the induction hypotheses in the same way as the conclusion is generated. An induction case in which the second component is the empty set, corresponds to a base case.

Note that the cover set $\mathcal{C}_1$ covers all the values of *divides* where the first argument is non-zero or both the arguments are zero. Hence the induction scheme generated by the cover set can be used to prove properties under the condition that the first argument of *divides* is non-zero.

For the above conjecture, the first base case is obtained by using the first tuple in the induction scheme (it comes from the first rule), and the substitutions for the variables are $x = u, y = 0$:

$$divides(u, 0+0)\ if\ divides(u,0) \wedge u > 0.$$

This formula simplifies to true using the definition of $+$ and the first rule. The second base case is obtained from the second tuple (and it comes from the second rule) and the substitutions for the variables are $x = u, y = v$ under the condition $v < u\ and\ v \neq 0$:

$$divides(u, v+v)\ if\ divides(u,v) \ \wedge\ (u > 0) \wedge (v < u)\ \wedge\ v \neq 0,$$

which trivially simplifies to true since $divides(u,v)$ in the condition simplifies to $false$ under the condition $v < u\ and\ v \neq 0$ using the second rule.

The induction step comes from the third tuple in the induction scheme (and it comes from the third rule); the substitution for the variables in the conclusion are $x = u, y = u + v$:

$$divides(u, (u+v) + (u+v))\ if\ divides(u, u+v) \wedge u > 0,$$

with the substitutions for the variables in the induction hypothesis coming from the second component as $x = u, y = v$:

$$divides(u, v+v)\ if\ divides(u,v) \wedge u > 0.$$

Using the associativity and commutativity properties of + and applying the third rule thrice, the conclusion above reduces to:

$$divides(u, v + v)\ if\ divides(u, v) \wedge u > 0,$$

which is the induction hypothesis. So the conjecture is proved.

The reader would have noticed how using the well-founded ordering suggested by the definition of *divides* led to an induction hypothesis which turned out to be useful in proving this conjecture.

Now consider a related conjecture:

$$(P_2): \quad divides(2, x) = not(divides(2, s(x))),$$

where 2 is an abbreviation for $s(s(0))$.[4] Using the cover set of *divides*, the following induction scheme for $divides(2, x)$ at position 1 is generated:

$$\begin{aligned}(I_2):\ \{&\langle\langle\{x \rightarrow 0\}, \{\}, \{1 \leftarrow divides(2, 0)\}\rangle, \{\}\rangle,\\ &\langle\langle\{x \rightarrow v\}, \{v < 2, v \neq 0\}, \{1 \leftarrow divides(2, v)\}\rangle, \{\}\rangle,\\ &\langle\langle\{x \rightarrow 2 + v\}, \{\}, \{1 \leftarrow divides(2, 2 + v)\}\rangle, \{\langle\{x \rightarrow v\}, \{\},\\ &\qquad\qquad \{1 \leftarrow divides(2, v)\}\rangle\}\rangle\}.\end{aligned}$$

There are two base cases: $divides(2, 0) = not(divides(2, s(0)))$, which is proved using rules 1 and 2; similarly, $divides(2, v) = not(divides(2, s(v)))\ if\ v < 2 \ \wedge\ v \neq 0$, which simplifies to $not(divides(2, s(1))) = false$, since $v \neq 0 \ \wedge\ v < 2$ implies $v = 1 = s(0)$.

If we check for applicability of definitions and lemmas modulo the theory of linear arithmetic, rewriting can get very expensive. Since rewriting is a primitive operation in a rewrite-based theorem prover such as *RRL*, performing rewriting modulo a theory such as linear arithmetic can slow the theorem prover down considerably. It is thus necessary to use the linear arithmetic procedure in a judicious manner to widen the scope of the cover set method and at the same time, maintain efficiency. In this paper, we do not rewrite modulo the linear arithmetic theory.

We discuss how the subgoal $divides(2, s(1))$ can be proved without rewriting modulo linear arithmetic after considering the induction step generated from the third case in the induction scheme.

For the induction step, the conclusion is:

$$divides(2, 2 + v) = not(divides(2, s(2 + v))),$$

assuming the induction hypothesis: $divides(2, v) = not(divides(2, s(v)))$.

Rule 3 is now applicable on the left side of the conclusion. But what about the right side? Using semantic information about natural numbers, s and $+$, it

[4] This example is taken from the *Nqthm* Corpus. This conjecture is proved there with the help of an explicit induction hint.

would be possible to see that $s(2+v) = 2 + s(v)$[5], so rule 3 is applicable to $not(divides(2, s(2+v)))$ to give $not(divides(2, s(v)))$. As stated above, for reasons of efficiency, we do not wish to rewrite modulo the theory of linear arithmetic, so we achieve this by *merging* induction schemes for the two different occurrences of *divides* in the conjecture. So an induction scheme for $divides(2, s(x))$ at position 2.1 is also generated using the cover set of *divides*. The induction schemes for $divides(2, x)$ and $divides(2, s(x))$ are then merged to give the scheme:

$$\begin{aligned}(I_3): \ \{&\langle\langle\{x \to 0\}, \{\}, \{1 \leftarrow divides(2,0), 2.1 \leftarrow divides(2, s(0))\}\rangle, \{\}\rangle,\\ &\langle\langle\{x \to 1\}, \{\}, \{1 \leftarrow divides(2,1), 2.1 \leftarrow divides(2, 2+0)\}\rangle, \{\}\rangle,\\ &\langle\langle\{x \to 2+v\}, \{\}, \{1 \leftarrow divides(2, 2+v), 2.1 \leftarrow divides(2, 2+(v+1))\}\rangle,\\ &\qquad \{\langle\{x \to v\}, \{\}, \{1 \leftarrow divides(2,v), 2.1 \leftarrow divides(2, v+1)\}\rangle\}\rangle\}.\end{aligned}$$

The first base case is proved as it was done before. The second base case becomes $divides(2,1) = not(divides(2, 2+0))$ which is established now by the rules 2 and 3 of the definition of *divides*.

In the induction step, the conclusion is:

$$divides(2, 2+v) = not(divides(2, 2+(v+1))),$$

with the hypothesis: $divides(2,v) = not(divides(2, v+1))$. The conclusion, by the use of rule 3, is reduced to

$$divides(2, v) = not(divides(2, v+1)),$$

to which now the hypothesis is applicable. The reason for keeping replacements for subterms being used for generating induction schemes should be evident. In the above conclusion, the subterm at position 1 is already in the form $divides(2, 2+v)$ so that rule 3 in the definition of *divides* is directly applicable; a similar remark applies for the subterm $divides(2, 2+(v+1))$ at position 2.1.

As the reader must have observed, such semantic analysis can be performed in the case of natural numbers using the linear arithmetic decision procedure since reasoning about $0, s, +, =, \geq$ needs to be performed.

In this paper, we show how Presburger arithmetic can be used in mechanizing induction to reconcile different representations of natural numbers – the one using successor and the other using $+, <$. We discuss how a decision procedure for Presburger arithmetic, also called LA, allows to go back and forth between these two representations in a judicious manner for generating appropriate induction hypotheses. In this way, more and more aspects of proofs by induction can be automated. We discuss how LA can be used to elaborate on a cover set of a function to generate appropriate instantiations for variables in a conjecture to automate its proof.

[5] Otherwise, it may be necessary to rewrite backwards which would have to be done with a user providing guidance.

We also illustrate how semantic analysis using LA is helpful in checking completeness of functions defined using $+$ and $<$, completeness of a cover set, as well as for merging induction schemes and generalization heuristics in the context of mechanizing proofs by induction.

The main reason for focusing on a linear arithmetic procedure is that it is already integrated in several theorem proving systems such as *Nqthm, PVS, NEVER* and *RRL*. Currently in *RRL*, the linear arithmetic procedure is used to prove simple facts or discharge conditions arising in lemmas or definitions [9]. In this paper, we are proposing another important use of the linear arithmetic procedure. The approach presented in this paper should apply to other data structures with multiple representations such as finite lists, finite sequences, finite sets and multisets, etc., and for which it is possible to devise decision procedure (or heuristics) to go back and forth among different representations.

2. Cover Sets, Induction Schemes, Linear Arithmetic

In this section we show how a linear arithmetic decision procedure LA can be used to generate induction schemes. We first give definitions of a cover set and an induction scheme based on a cover set; then we illustrate how on certain class of conjectures, the cover set method fails because it relies solely on syntactic unification (i.e. unification modulo the empty theory). Linear arithmetic can be used to remedy the problem since LA can be used to perform semantic unification. We motivate the algorithm using the running example of *divides* given in the introduction. An algorithm for generating an induction scheme for a conjecture from a cover set is given.

2.1. Cover Sets

Given a set F of function symbols and X of variables, $T(F, X)$ denotes the set of terms constructed using these. Given a term t, by $Vars(t)$, we denote the set of variables that occur in t. A term t whose outermost function symbol is f is called an f-term. An f-term of the form $f(x_1, \cdots, x_n)$ where x_i's are distinct variables is called a *basic* f-term. A substitution σ is a finite map from variables to terms written as $\{x_1 \rightarrow s_1, \cdots, x_k \rightarrow s_k\}$, $x_i \in X$ and $s_i \in T(F, X)$, $1 \leq i \leq k$. $\sigma \circ \tau$ denotes the composition of substitutions σ and τ. Given a substitution σ, E_σ denotes σ viewed as equations: $\sigma = \{x_1 \rightarrow s_1, \cdots, x_k \rightarrow s_k\}$, $E_\sigma = \{x_1 = s_1, \cdots, x_k = s_k\}$. $=_E$ is used to denote equality amongst terms with respect to a set of (possibly conditional) equations E.

A definition of a function symbol $f \in F$ is a finite set of conditional equations of the form $l_i = r_i$ if $cond_i$, $1 \leq i \leq m$, where $cond_i$ is optional; l_i is an f-term from $T(F, X)$, r_i is a term in $T(F, X)$, and $cond_i$ is a conjunction of literals built out of terms in $T(F, X)$ such that $Vars(r_i) \subseteq Vars(l_i)$ and $Vars(cond_i) \subseteq Vars(l_i)$. Further it is assumed that each conditional equation in a definition can be oriented into a terminating rewrite rule, i.e. there exists a well-founded reduction ordering $\succ$ over $T(F, X)$ such that $l_i \succ r_i$ and $l_i \succ cond_i$ [4].

Constructors of a data structure need not be free (as in the case of integers and finite sets). It is assumed that constructor relations can be expressed as a finite set of equations. A function may be defined using nonconstructor symbols.

Let E be a finite set of equations possibly relating constructors and consisting of definitions of nonconstructors used in defining other nonconstructors.

A cover set $\mathcal{C}_f$ for a function f with a definition D is a finite set of triples where each triple is derived from a conditional rewrite rule in D. For a conditional rule $l \rightarrow r \ if \ cond$, the first component of the triple is the tuple of arguments of f in l, the second component is a set consisting of all tuples serving as arguments to f in r and the third component is the set of all literals in *cond*. A triple could have its second and third components be the empty set. $Vars(\mathcal{C}_f)$ denotes the variables of a cover set $\mathcal{C}_f$.

A cover set $\mathcal{C}_f$ for a function $f(x_1, \cdots, x_n)$ is *complete* if and only if for any n-tuple $\langle e_1, \cdots, e_n \rangle$ of ground constructor terms over domains of f, there exists a cover set triple, $\mathrm{c}_i = \langle \langle s_1, \cdots, s_n \rangle, \{\cdots, \langle s'_1, \cdots, s'_n \rangle, \cdots\}, cond \rangle$ and a ground substitution σ such that $\sigma(s_i) =_E e_i$, $1 \leq i \leq n$ and $\sigma(cond) =_E true$. The triple c_i is said to *cover* the tuple $\langle e_1, \cdots, e_n \rangle$. If a cover set $\mathcal{C}_f$ of a function is complete then it follows that f is completely defined.

If a function definition is given solely using constructors, i.e. all the rules in the definition involve only the function symbol being defined and constructor symbols, the definition is called *constructor-based*. A cover set generated from a constructor-based definition contains terms only over constructors. Further if there are no relations on constructors, implying E is the empty set, then in the definition of a complete cover set, $=_E$ is replaced by $=$ and matching can be used. Contextual rewriting with linear arithmetic procedure implemented in [10, 9] is used to discharge the conditions while checking completeness of cover sets. Henceforth, unless stated explicitly, all the cover sets of the function symbols are assumed to be complete.

2.2. Induction Schemes

Given a conjecture C, different nonvariable subterms in C may suggest different ways of performing induction on variables in C. *RRL* currently supports a variety of heuristics to choose an appropriate induction when there are multiple alternatives. Each of these possible alternatives can be specified as an induction scheme. We first discuss the induction scheme generated from a basic f-term $f(x_1, \cdots, x_n)$ appearing in C. Later we discuss how the induction scheme can be generated from a nonbasic f-term $f(t_1, \cdots, t_n)$ of C, where t_i's are not necessarily variables.

The induction scheme suggested by a basic f-term, $f(x_1, \cdots, x_n)$ in C is derived directly from a cover set $\mathcal{C}_f$ of f since it is similar to $\mathcal{C}_f$. Such an induction scheme is called *basic*. For example, the induction scheme I_1 in the introduction is obtained from $divides(x, y)$ and its cover set, and is basic. As discussed earlier, an induction scheme is a finite set of induction cases of the form

$$\langle\langle \sigma_c, cond_c, repl_c\rangle, \{\cdots, \langle\theta_i, cond_i, repl_i\rangle, \cdots\}\rangle,$$

where $\langle \sigma_c, cond_c, repl_c\rangle$ is used to generate the induction conclusion and each triple in $\{\cdots, \langle\theta_i, cond_i, repl_i\rangle, \cdots\}$ is used to generate an induction hypothesis. Each of these components of an induction case is obtained from a cover set triple $\langle\langle s_1, \cdots, s_n\rangle, \{\cdots, \langle s_1^i, \cdots, s_n^i\rangle, \cdots\}, cond\rangle$ in a cover set $\mathcal{C}_f$ of f as follows: $\sigma_c = \{x_1 \rightarrow s_1, \cdots, x_n \rightarrow s_n\}$, $cond_c = cond$, and $repl_c = \{p \leftarrow f(s_1, \cdots, s_n)\}$, where p is the position of $f(x_1, \cdots, x_n)$ in C; similarly, $\theta_i = \{x_1 \rightarrow s_1^i, \cdots, x_n \rightarrow s_n^i\}$, $cond_i = cond$, and $repl_i = \{p \leftarrow f(s_1^i, \cdots, s_n^i)\}$.

The substitutions σ_c and each of the substitutions θ_i are linked through the variables shared amongst the left hand side and the recursive calls on the right hand side of the rule from which the cover set triple is derived. The variables whose substitutions are not invariant across the induction conclusion and the hypotheses are called *induction variables*. In the induction scheme I_1 generated from the cover set of $divides(x, y)$, x, y are both induction variables. Given an induction scheme ϕ, let $indvars(\phi)$ denote the induction variables of ϕ and $Vars(\phi)$ denote all the variables substituted for in ϕ.

The induction subgoal corresponding to the above induction case is: [6]

$$(\sigma_c(C[repl_c])\ if\ cond_c)\ if\ \{\bigwedge_i\ \theta_i(C[repl_i])\ if\ cond_i\}.$$

2.2.1. *Completeness and Soundness of Basic Induction Schemes*

An induction scheme can be used for an inductive proof attempt of a given conjecture provided it is *complete* and *sound*.[7] The completeness and soundness of a basic induction scheme is directly linked to those of the underlying cover set from which it was obtained. A complete cover set results in a complete induction scheme. For constructor-based definitions with free constructors, the soundness of a basic induction scheme follows from the reduction ordering $\succ$ used to prove the termination of the definition of f.

When there are relations amongst constructors or the definition of f uses non-constructors, i.e., E above is non-empty, a reduction ordering $\succ$ preserving the congruence relation $=_E$ must be used for ensuring that the arguments appearing in the recursive calls on the right side (and/or condition) of each rule in the definition of f is lower than its left side. A reduction ordering $\succ$ preserves $=_E$ iff $u =_E v \succ s =_E t$ implies $u \succ t$ for all terms u, v, s, t. We use $\succ_E$ to denote such an ordering. For any two terms t_1 and t_2, $t_1 \succ_E t_2$ iff for any ground substitution σ and $t =_E \sigma(t_1)$, $t \succ \sigma(t_2)$ [13, 12].

[6] $C[\{p_1 \leftarrow t_1, \cdots, p_n \leftarrow t_n\}]$ denotes the conjecture C with its subterm at position p_i replaced by the term t_i $1 \leq i \leq n$.

[7] See, however, subsection 4.1.1 in which it is shown how incomplete cover sets can be used to generate induction schemes that can also be useful in proofs by induction.

Alternatively, only those instantiations of every rule defining f can be used for generating the cover set for which the recursive calls to f in the right side and the condition are lower than the left side with respect to $\succ_E$. In order to ensure this, the third component of a cover set triple *cond* should be strengthened to include an additional *termination condition* ensuring that every n-tuple in the second component in the triple is lower than the n-tuple appearing in the first component. This approach would be adopted later in this section when a linear arithmetic decision procedure is used for generating cover sets and induction schemes.

THEOREM 1. *Induction scheme ϕ generated from a basic f-term $t = f(x_1, .., x_n)$ of a conjecture $C(\cdots, t, \cdots)$ and a complete cover set $\mathcal{C}_f$ is complete and sound.*

Proof. Completeness : Since $\mathcal{C}_f$ is complete, any n-tuple $\langle e_1, \cdots, e_n\rangle$ of ground constructor terms is *covered* by some cover set triple $c_i = \langle\langle s_1, \cdots, s_n\rangle, \{\cdots, \langle s_1^j, \cdots, s_n^j\rangle\}, cond\rangle$, i.e. for a ground substitution β, $\beta(f(s_1, \cdots, s_n)) =_E f(e_1, \cdots, e_n)$ and $\beta(cond) =_E true$, and it is therefore covered by the corresponding induction case $\phi_i = \langle\langle\sigma_c, cond_c, repl_c\rangle, \{\cdots, \langle\theta_j, cond_j, repl_j\rangle, \cdots\}\rangle$, where $\sigma_c = \{x_1 \rightarrow s_1, \cdots, x_n \rightarrow s_n\}$, $\theta_j = \{x_1 \rightarrow s_1^j, \cdots, x_n \rightarrow s_n^j\}$, $cond_c = cond_j = cond$.

Soundness: By contradiction. Assume that $C(\cdots, t, \cdots)$ is not true but the subgoal obtained from each induction case of ϕ is true. Let $\langle e_1, \cdots, e_n\rangle$ be the smallest n-tuple of ground constructor terms with respect to the ordering $\succ_E$ used to prove the termination of f, that is a counterexample to $C(\cdots, t, \cdots)$, i.e. $\beta \circ \sigma_c(C) =_E false$, where $\beta(f(s_1, \cdots, s_n)) =_E f(e_1, \cdots, e_n)$. The induction subgoal corresponding to ϕ_i given above, ($\sigma_c(C)$ *if cond*) assuming ($\bigwedge_j(\theta_j(C)$ *if cond*) is $true$. Since $\beta \circ \sigma_c(C)$ is equivalent to $false$ and $\beta(cond) =_E true$, $\beta \circ \theta_j(C)$ is $false$ for some j, implying that there is a smaller counterexample to C, since $\beta(f(s_1^j, \cdots, s_n^j))$ is lower in the ordering $\succ_E$ than $f(e_1, \cdots, e_n)$ (by the definition of a cover set, $f(s_1, \cdots, s_n) \succ_E f(s_1^j, \cdots, s_n^j)$ for any j, and since $\succ_E$ is preserved under $=_E$, $f(e_1, \cdots, e_n) \succ_E f(\beta(s_1^j), \cdots, \beta(s_n^j)))$. And, this is a contradiction. ∎

2.2.2. *Nonbasic Induction schemes*

The completeness of an induction scheme ϕ generated from a nonbasic f-term $f(t_1, \cdots, t_n)$ ensures that every ground instance of t_i's is *covered* by some induction case of ϕ.

DEFINITION. *Complete Induction Schemes:* An induction scheme ϕ generated from an f-term $f(t_1, \cdots, t_n)$ is *complete* iff for every ground substitution β there is an n-tuple $\langle e_1, \cdots, e_n\rangle$ of ground constructor terms over the domains of f such that $\beta(f(t_1, \cdots, t_n)) =_E f(e_1, \cdots, e_n)$, there exists an induction case ϕ_j: $\langle\langle\sigma_c$, $cond_c$, $repl_c\rangle$, $\{\cdots, \langle\theta_i$, $cond_i, repl_i\ \rangle, \cdots\}\rangle$ and a ground substitution γ such that $\gamma \circ \sigma_c(f(t_1, \cdots, t_n)) =_E f(e_1, \cdots, e_n)$, and $\gamma(cond_c) = true$.

Soundness of an induction scheme ϕ guarantees that if a conjecture C was proved by induction using ϕ then C is indeed a theorem. More precisely, an induction scheme ϕ generated from an f-term $f(t_1, \cdots, t_n)$ of a given conjecture C is sound iff a proof of C by induction on ϕ implies that for every n-tuple $\langle e_1, \cdots, e_n \rangle$ of ground constructors such that $\beta(f(t_1, \cdots, t_n)) =_E f(e_1, \cdots, e_n)$ for a ground substitution β, and $\beta(C) = true$.

Below we give an algorithm for generating an induction scheme from a nonbasic f-term $t = f(t_1, \cdots, t_n)$ appearing in a conjecture C from a cover set $\mathcal{C}_f$ of f. The algorithm uses unification to generate different cases of the induction scheme from $\mathcal{C}_f$. If $t = f(x_1, \cdots, x_n)$, it generates a basic induction scheme discussed above. In order to keep the presentation simple, the algorithm is given assuming the definition of f to be constructor-based and that there are no relations on constructors(E is empty). The algorithm can be generalized to consider an arbitrary E if syntactic unification (unification modulo the empty theory) below is replaced by $=_E$ unification. Of course, $=_E$ unification must be decidable and finitary, and each of the most general unifiers (*mgus*) in steps 2(a) is used to generate an induction case, and every *mgu* in step 2(b) is used to generate an induction hypothesis.

2.2.3. *Algorithm for Generating an Induction Scheme*

- **Input** : A conjecture C of the form $l = r \;\; if \;\; cond$, a definition D of an n-ary function f and a term $t = f(t_1, \cdots, t_n)$ at a position p in C.
- **Output** : An induction scheme based on D.
- **Method**:

  1. *Initialize*: Compute the cover set $\mathcal{C}_f$ from D.
  2. *Compute the induction scheme*: For each cover set triple,
     c = $\langle \langle s_1, \cdots, s_n \rangle, \{\cdots, \langle s_1^i, \cdots, s_n^i \rangle, \cdots\}, cond \rangle$ in $\mathcal{C}_f$ do:
     a) *Generate induction conclusion*: Let σ be the *mgu* of t and $s = f(s_1, \cdots, s_n)$. The induction conclusion is: $\langle \sigma_c, cond_c, repl_c \rangle$ where σ_c is the restriction of σ on $Vars(t)$; $cond_c = \sigma(cond)$; $repl_c = \{p \leftarrow \sigma(s)\}$.
     b) *Generate the induction hypotheses*: Let σ_i be the *mgu* of t and $s' = f(s_1^i, \cdots, s_n^i)$ for some i. The i^{th} induction hypothesis based on σ_i is: $\langle \theta_i, cond_i, repl_i \rangle$ where θ_i is the restriction of σ_i on $Vars(t)$; $cond_i = \sigma_i(cond)$; $repl_i = \{p \leftarrow \sigma_i(s')\}$.

In the above algorithm for generating an induction scheme, if unification of t with $s = f(s_1, \cdots, s_k)$ (or $f(s'_1, \cdots, s'_k)$) fails, then the safest conclusion is that the cover set method has failed to derive an induction scheme corresponding to the term under consideration (especially if t has occurrences of nonconstructors). If t_i's are constructor terms, then the above algorithm based on syntactic unification can be used to generate a sound and complete scheme from an f-term t as illustrated by the following theorem.

THEOREM 2. *The induction scheme ϕ generated from $t = f(t_1, \cdots, t_n)$ of a conjecture $C(\cdots, t, \cdots)$ and a constructor-based complete cover set $\mathcal{C}_f$, where t_i, $1 \leq i \leq n$ is a constructor term, assuming no relations on constructors, is complete and sound.*

Proof. *Completeness*: Without loss of generality assume that $Vars(t) \cap Vars(\mathcal{C}_f) = \{\}$. Since the cover set $\mathcal{C}_f$ is complete, for any n-tuple $\langle e_1, \cdots, e_n \rangle$ of ground constructor terms, there exists a ground substitution β and a cover set triple $c_i = \langle \langle s_1, \cdots, s_n \rangle, \{\cdots, \langle s_1^k, \cdots, s_m^k \rangle, \cdots\}, cond \rangle$ such that $\beta(s) = e$ and $\beta(cond) = true$ where $s = f(s_1, \cdots, s_n)$, and $e = f(e_1, \cdots, e_n)$. Let ϕ_i be the induction case corresponding to c_i with the conclusion $\langle \delta_c, \delta(cond), \{p \leftarrow f(\delta(s_1), \cdots, \delta(s_n))\} \rangle$ where δ is the *mgu* of s and t, δ_c is the restriction of δ on $Vars(t)$ and p is the position of t in C.

If for some ground substitution γ, $\gamma(t) = e$ then s and t are unifiable, and $\beta' = \beta \cup \gamma$ is a unifier of s and t ($\beta'(s) = e = \beta'(t)$ since $Vars(s) \cap Vars(t) = \{\}$). So, $\beta' = \theta \circ \delta$ for some substitution θ. Further, since $\beta(cond) = true$, $\beta'(cond) = true$. This implies that $\theta \circ \delta_c(t) = e$ and $\theta \circ \delta(cond) = true$ and therefore, $\langle e_1, \cdots, e_n \rangle$ is *covered* by the conclusion, of the induction case ϕ_i. Hence the induction scheme ϕ is complete.

Soundness: By contradiction along the same lines as that of Theorem 1. ∎

2.3. Linear Arithmetic and Cover sets

When we fail to syntactically unify an arbitrary f-term t with s, no induction case corresponding to the associated cover set triple is generated. If t and/or s involves nonconstructor terms, this however, does not preclude the existence of an n-tuple $\langle e_1, \cdots, e_n \rangle$ of ground constructor terms over the domains of f, which is equivalent to an instance of t as well as an instance of s. Such an n-tuple would not be *covered* by any of the induction cases of the scheme generated and hence the resulting scheme would be incomplete and therefore, unsound. This problem can be avoided if unification of t and s is performed modulo a theory containing function symbols other than f occurring in t as well as in s. Consider for example the data structure of numbers (natural numbers or integers); s, t could involve $+$. In the next two sections we describe how a linear arithmetic decision procedure can be used to carry out such unification modulo the theory of linear arithmetic for generating complete and sound induction schemes for f-terms whose arguments are terms over the theory of linear arithmetic, and the function f is defined using terms over the same theory.

The theory of linear arithmetic denotes the quantifier-free first order theory of numbers (integers or natural numbers), numeric variables, the arithmetic operations (successor(s), predecessor(p) and addition($+$)) with the arithmetic relations

$(>, <, \neq, =, \geq, \leq)$. A linear term is either a number or a variable or is of the form $f(t_1, \cdots, t_n)$ where f is an n-ary arithmetic operation and t_i, $1 \leq i \leq n$, are linear terms. If p is an n-ary arithmetic relation, $p(t_1, \cdots, t_n)$, where t_i's are linear terms denotes a linear atom. Linear literals are either linear atoms or negations of linear atoms. $=_{LA}$ is used to denote the equality amongst terms with respect to the linear arithmetic theory.

Let $CE = \{s_i = t_i | 1 \leq n\}$ be a set of equations with a context *cond* where s_i's and t_i's are linear terms and *cond* is a conjunction of linear literals. Let $\{x_1, \cdots, x_m\}$ be the variables occurring in CE and *cond*. An m-tuple of $\langle e_1, \cdots, e_m\rangle$ of numbers is a solution to CE under *cond* iff $\sigma(cond) =_{LA} true$ and $\sigma(s_i) =_{LA} \sigma(t_i)$, for all $1 \leq i \leq n$, where $\sigma = \{x_j \rightarrow e_j | 1 \leq j \leq m\}$.

By the linear arithmetic procedure LA, we assume the existence of an algorithm specified as follows.

- *Input:* A set of equations $CE : \{s_i = t_i | 1 \leq i \leq n\}$ and a context *cond*, where s_i's and t_i's are linear terms and *cond* is a conjunction of linear literals.

- *Output:* No, if CE cannot be solved. Otherwise, yes, with a solved form of CE as a substitution σ of the form $\{x_j \rightarrow u_j\}$ satisfying the occur-check, where u_j are linear terms possibly involving new variables introduced while solving CE, and feasibility constraints c_c, a conjunction of linear literals restricting values that variables can take.

- *Completeness and Soundness:* The input and the output of the LA are related such that any m-tuple $e = \langle e_1, \cdots, e_m\rangle$ of numbers is a solution to the equations CE with context *cond* if and only if e is a solution to E_σ with context $\{\sigma(cond), c_c\}$.

For equations, a solved form, if any, can be obtained using an algorithm for solving linear diophantine equations and conditions on variables, if needed, can be included in feasibility constraints. We assume that given a linear literal of the form $x \leq c$ where c is a constant it is possible to enumerate using the procedure the various values for x, $x = c_i$ where $c_i \leq c$. The requirement of completeness is only an algorithm for solving a set of linear equations under a context. Further, since the input is a restricted subset of universally quantified linear arithmetic theory it can be shown that a complete procedure would either detect the unsolvability of the given set of equations under the given context or would lead to a solution that is expressible as a single substitution along with feasibility constraints.

DEFINITION. *LA-based Function Definitions:* A definition D of an n-ary function f is LA-based iff for each rule $l \rightarrow r$ *if cond* of D, the arguments to f in l and r are linear terms; *cond* is a conjunction of linear literals.

Most of the definitions in this paper are LA-based definitions. For example, the definition of *divides* in the introduction is an LA-based definition.

For using LA to do semantic analysis for generating an induction scheme from a cover set, there is a need to modify the definition of cover set slightly. Since we propose to use LA to perform semantic unification of the terms in the conjecture with those in the cover set generated from definitions of functions, it is not enough if the definitions of the functions are oriented into terminating rules based on a well-founded reduction ordering. Such orderings need not be preserved under semantic transformations. For instance, given a rule such as $f(x+0) \rightarrow f(x)$ in the definition of f, semantic unification with a term $f(n)$ in a conjecture using LA, can replace the term $x+0$, by x which is semantically equivalent to it, to produce an induction case of the form $\langle\langle\{n \rightarrow x\},\{\},\{\}\rangle,\{\langle\{n \rightarrow x\},\{\},\{\}\rangle\}\rangle$, which is unsound. To avoid this, we require that the reduction ordering $\succ$ preserve the semantic congruence relation amongst numbers induced by $=_{LA}$. For soundness, for every rule $l \rightarrow r \ if \ cond$, where l is an f-term and the arguments to f are linear terms, arguments in recursive calls of f in r (as well as $cond$) must be lower than the arguments to f in l in a semantic well-founded ordering over numbers. This additional requirement is incorporated in every triple in the cover set of f by strengthening its condition component $cond$ to include a *termination* condition.

For natural numbers, a lexicographic order on k-tuples using $>$ works as a well-founded order. For integers, a lexicographic order using $>$ on absolute values of linear terms would work. A semantic well-founded ordering on linear terms must be preserved under substitutions. A well-founded semantic ordering on linear terms, $\succ_{LA}$ is defined as : $t_1 \succ_{LA} t_2$ iff for every substitution σ from variables to numbers, $|\sigma(t_1)| > |\sigma(t_2)|$ where $|t|$ denotes the absolute value of a linear term t. $\succ_{lex}$ denotes the lexicographic extension of $\succ_{LA}$ to tuples of linear terms. For example, for the third rule defining *divides*, $\langle u, u+v\rangle > \langle u, v\rangle$ which is the case only if $u+v > v$; so the termination condition $u+v > v$ is added to the third component in the triple corresponding to the third rule in the cover set for *divides*.

DEFINITION. *LA-based Cover Sets:* A cover set $\mathcal{C}_f$ of an n-ary function f is LA-based iff for any cover set triple $c_i = \langle\langle s_1, \cdots, s_n\rangle, \{\cdots, \langle s_1^k, \cdots, s_n^k\rangle, \cdots\}, cond \wedge T\rangle$ of $\mathcal{C}_f$, s_i and s_i^k, $1 \leq i \leq n$, for any k, are linear terms and $cond$ is a conjunction of linear literals and T, the termination condition, is a conjunction of linear literals ensuring that $\langle s_1, \cdots, s_n\rangle \succ_{lex} \langle s_1^k, \cdots, s_n^k\rangle$, for any k.

Based on the above definition, the LA-based cover set of the predicate *divides* introduced in section 1 gets modified as follows. $\mathcal{C}_{divides} : \{\langle\langle u, 0\rangle, \{\}, \{\}\rangle, \langle\langle u, v\rangle, \{\}, \{v < u, v \neq 0\}\rangle, \langle\langle u, u+v\rangle, \{\langle u, v\rangle\}, \{u+v > v\}\rangle\}$.

DEFINITION. *Complete LA-based Cover Sets:* An LA-based cover set $\mathcal{C}_f$ for an n-ary function f is *complete* iff for any n-tuple of numbers $\langle e_1, \cdots, e_n\rangle$ there exists a cover set triple $\langle\langle s_1, \cdots, s_n\rangle, \{\cdots, \langle s_1', \cdots, s_n'\rangle, \cdots\}, cond \wedge T\rangle$ and a ground substitution σ such that $\sigma(s_i) =_{LA} e_i$ is for all i, $1 \leq i \leq n$, and $\sigma(cond \wedge T) = true$.

In discussions below we assume that the termination condition T is integrated into *cond*, the condition governing the cover set triples and hence it will not be explicitly specified.

2.4. Linear Arithmetic based Induction Schemes

For generating an induction scheme from an f-term t in a conjecture C and the cover set of f, if the arguments of f in t corresponding to the induction variables are non-variable terms, then the algorithm in section 2.2.3 often fails. For example, take t to be $divides(x, y+y)$ in the conjecture discussed earlier. It is not possible to derive an induction scheme using the cover set $\mathcal{C}_1$ since, $y+y$ in C cannot be unified with 0 for the first cover set triple itself.

Another possible situation to consider is when the arguments to the term t in the argument positions corresponding to the induction variables are not distinct variables. For example, take t to be $divides(x, x)$ in some conjecture. Again the cover set method fails to generate an induction scheme for this case because x needs to be unified with both u and $u+v$ to generate an induction case from the last triple of the cover set, which is not possible.

Failure in both cases is because of purely syntactic view taken in attempting to unify terms expressed using 0 and $+$. In the first case, $y+y$ can be unified with 0 using semantic analysis giving the most general unifier $\{y \rightarrow 0\}$. Similarly, in the second case also, x can be unified with both u and $u+v$ giving the most general unifier $\{x \rightarrow u, v \rightarrow 0\}$. There is thus a need to relate arbitrary terms expressed using $0, s, +$, and this can be achieved using the linear arithmetic procedure (LA). Using LA, both the cases above can be solved.

Using LA, we can generalize the algorithm in subsection 2.2.3 which uses syntactic unification for generating the induction scheme for a conjecture C, to perform semantic unification of a selected subterm $t = f(t_1, \cdots, t_n)$ and the cover set associated with the outermost symbol f of t. Given a cover set triple, c $= \langle\langle s_1, \cdots, s_n\rangle, \{\cdots, \langle s_1^j, \cdots, s_n^j\rangle, \cdots\}, cond\rangle$, constraint equations $\{t_i = s_i \mid 1 \leq i \leq n\}$ are set up; LA is invoked to produce substitution σ of $Vars(t) \cup Vars(\mathrm{c})$ with context *cond*. If the third component of c is empty then the context is assumed to be *true*. The restriction of σ on $Vars(t)$ produces the substitution corresponding to the induction conclusion; similarly, substitution σ_j of $Vars(t) \cup Vars(\mathrm{c})$ for induction hypotheses are generated by invoking LA on the constraint equations $\{t_i = s_i^j \mid 1 \leq i \leq n\}$ with context *cond*. Restriction of σ_j on $Vars(t)$ is the substitution corresponding to the j^{th} induction hypothesis.

Since we wish to avoid rewriting modulo linear arithmetic, we associate with the induction conclusion and with each of the induction hypotheses, a set of *positional* replacements of the form $\{p \leftarrow \sigma(f(s_1, \cdots, s_k))\}$ and $\{p \leftarrow \sigma_i(f(s'_1, \cdots, s'_k))\}$ respectively, where p is the position of the term t in the conjecture C. These replacements enable the use of the rules from the definition of f for simplification as seen in the case of conjecture P_2 earlier.

For example, consider the term $divides(2, x)$ at position 1 in the conjecture P_2 presented in the introduction. From the first cover set triple of *divides*, we get $\{x \to 0\}$, as the substitution. From the second triple, the substitution is obtained from the constraints $\{2 = u, x = v\}$ with context $\{v < u, v \neq 0\}$, which using LA, simplify to $\{x \to 1, u \to 2, v \to 1\}$. From the third cover set triple, the constraints corresponding to the induction conclusion are: $\{2 = u, x = u + v\}$ with context $\{u + v > v\}$, which simplify to $\{x \to 2 + v, u \to 2\}$.[8] In a similar manner, the substitution corresponding to the induction hypothesis is obtained to be $\{x \to v, u \to 2\}$. Hence the induction scheme is given as:

$$\{\langle\langle\{x \to 0\}, \{\}, \{1 \leftarrow divides(2, 0)\}\rangle, \{\}\rangle, \langle\langle\{x \to 1\}, \{\}, \{1 \leftarrow divides(2, 1)\}\rangle, \{\}\rangle,\\ \langle\langle\{x \to 2 + v\}, \{\}, \{1 \leftarrow divides(2, 2 + v)\}\rangle, \{\langle\{x \to v\}, \{\},\\ \{1 \leftarrow divides(2, v)\}\rangle\}\rangle\}.$$

The induction scheme for the term $divides(2, s(x))$ at position 2.1 in the conjecture is computed as follows. No induction case is generated corresponding to the first cover set triple, since 0 and $s(x)$ cannot be unified using LA. From the second cover set triple, the constraints are: $\{u' = 2,\ v' = s(x)\}$ with context $\{v' < u',\ v' \neq 0\}$, which simplifies using LA to $\{x \to 0, u' \to 2, v' \to s(0)\}$. From the third cover set triple, the constraints corresponding to the conclusion are: $\{\ u' = 2,\ u' + v' = s(x)\}$ with context $\{u' + v' > v'\}$, which simplifies to: $\{x \to v' + 1, u' \to 2\}$. The constraints for the hypothesis are: $\{u' = 2,\ v' = s(x)\}$ with context $\{u' + v' > v'\}$, which simplifies to: $\{x \to v' - 1, u' \to 2\}$ along with the feasibility constraint $\{v' \geq 1\}$.

For $divides(2, s(x))$, the induction scheme is specified as:

$$\{\langle\langle\{x \to 0\}, \{\}, \{2.1 \leftarrow divides(2, s(0))\}\rangle, \{\}\rangle,\\ \langle\langle\{x \to v' + 1\}, \{\}, \{2.1 \leftarrow divides(2, 2 + v')\}\rangle, \{\langle\{x \to v' - 1\}, \{v' \geq 1\},\\ \{2.1 \leftarrow divides(2, v')\}\rangle\}\rangle\}.$$

2.4.1. *Modified Algorithm for Generating an Induction Scheme*

- **Input** : A conjecture C of the form $l = r \ \ if \ \ cond$, an LA-based definition D of an n-ary function f and a term $t = f(t_1, \cdots, t_n)$ at a position p in C.

- **Output** : An induction scheme based on D.

- **Method**:

  1. *Initialize*: Compute the LA-based cover set $\mathcal{C}_f$ using D.

  2. *Compute induction schemes*: For each cover set triple,
     c = $\langle\langle s_1, \cdots, s_n\rangle, \{\cdots, \langle s'_1, \cdots, s'_n\rangle, \cdots\}, cond\rangle$ in $\mathcal{C}_f$ do:

[8] Note that $v \geq 0$ is an implicit feasibility constraint for any natural number v which is left unspecified.

a) *Generate induction conclusion*: The induction conclusion of the form $\langle \sigma_c, cond_c, repl_c \rangle$ is generated as:

   i) Set up constraint equations $CE = \{t_i = s_i \mid 1 \leq i \leq n\}$.

   ii) Solve CE using LA with context $\{cond\}$. If solvable, let $\sigma = \{x_i \rightarrow u_i\}$ be the substitution obtained which satisfies the occur-check, where u_i are linear terms possibly involving new variables and $x_i \in Vars(t) \cup Vars(\mathrm{c})$. Let c_c be the set of feasibility constraints over $Vars(\mathrm{c})$. No solution corresponds to a vacuous induction case.

   iii) Let μ be the restriction of σ on $Vars(\mathrm{c})$. σ_c is the restriction of σ on $Vars(t)$; $cond_c$ is $\mu(c_c) \cup \mu(cond)$ [9]; $repl_c$ is $\{p \leftarrow f(\sigma(s_1), \cdots, \sigma(s_k))\}$.

b) *Generate the set of induction hypotheses*: The i^{th} hypothesis $\langle \theta_i, cond_i, repl_i \rangle$ is generated as:

   i) Set up constraint equations $CE = \{t_i = s'_i \mid 1 \leq i \leq k\}$. Solve CE using LA as before, with context $\{cond_c, CE_\mu\}$. Let c'_c be the new feasibility constraints obtained and let σ_i be the new substitution obtained.

   ii) Use σ_i and c'_c to compute θ_i, $cond_i$ and $repl_i$ as before. No solution corresponds to the i^{th} hypothesis not being generated.

The induction scheme output by the above algorithm can be applied to the conjecture as discussed earlier. It should be noted that in the above modified algorithm the cases that do not yield solutions can be ignored since the underlying LA procedure is assumed to be sound and complete.

2.4.2. *Examples*

We illustrate the algorithm on some examples involving the greatest common divisor(gcd) function. The properties illustrated below cannot be proved with the cover set induction method without using LA. The methods implemented for induction in $Nqthm$ [1] are also unsuccessful in establishing these properties (these properties can be proved using hints and other tricks, however). The conjectures are based on following definition of the gcd function.

1. $gcd(0, x) \rightarrow x$, 2. $gcd(x, 0) \rightarrow x$,
3. $gcd(x + y, y) \rightarrow gcd(x, y)$, 4. $gcd(x, x + y) \rightarrow gcd(x, y)$.

The first conjecture that we consider is:

$$(P_3): \quad gcd(m + m, 2) = 2.$$

[9] $\sigma(\{s_1, \cdots, s_k\}) = \{\sigma(s_1), \cdots, \sigma(s_k)\}$.

1. *Initialize* : The LA-based cover set $\mathcal{C}_{gcd}$ for gcd is $\{\langle\langle 0,x\rangle,\{\},\{\}\rangle,\langle\langle x,0\rangle,\{\},\{\}\rangle,\langle\langle x+y,y\rangle,\{\langle x,y\rangle\},\{x+y>x\}\rangle,$ $\langle\langle x,x+y\rangle,\{\langle x,y\rangle\},\{x+y>y\}\rangle\}$.
   The conditions governing the last two cover set triples are terminating conditions. For the third rule, $\langle x+y,y\rangle \succ_{lex} \langle x,y\rangle$ iff $x+y > x$. Similarly the last rule defining gcd leads to the termination condition $x+y > y$.

2. Compute the induction scheme:

   a) The first cover set triple would generate the induction case: $\langle\langle\{m\rightarrow 0\},\{\},\{1\leftarrow gcd(0,2)\}\rangle,\{\}\rangle$. The second cover set triple would not generate an induction case, since $2=0$ cannot be solved.

   b) Using the third cover set triple $\langle\langle x+y,y\rangle,\{\langle x,y\rangle\},\{x+y>x\}\rangle$ to generate the substitution for the induction conclusion, the constraint equations are: $\{m+m=x+y, y=2\}$ with context $\{x+y>x\}$, and the substitution obtained is $\{m\rightarrow z+1, x\rightarrow z+z, y\rightarrow 2\}$ where z is a new variable introduced while solving for $m+m=x+2$. The constraint equations for the hypothesis are: $\{m+m=x, y=2\}$ with context $\{x=z+z, y=2\}$, and the substitution obtained is: $\{m\rightarrow z, y\rightarrow 2\}$. The induction case generated is: $\langle\langle\{m\rightarrow z+1\},\{\},\{1\leftarrow gcd(z+z+2,2)\}\rangle,$ $\{\langle\{m\rightarrow z\},\{\},\{1\leftarrow gcd(z+z,2)\}\rangle\}\rangle$.

   c) For the fourth cover set triple, the constraint equations for the conclusion are: $\{x=m+m, x+y=2\}$ with context $\{x+y>y\}$. The substitution obtained is $\{m\rightarrow 1, x\rightarrow 2, y\rightarrow 0\}$. The constraint equations for the hypothesis are: $\{x=m+m, y=2\}$ with context $\{y=0, x=2\}$, which are unsatisfiable. So there is no induction hypothesis for this case, and the induction case generated is: $\langle\langle\{m\rightarrow 1\},\{\},\{1\leftarrow gcd(2,2+0)\}\rangle,\{\}\rangle$.

   The scheme generated by the algorithm is:

$$\begin{aligned}\{&\langle\langle\{m\rightarrow 0\},\{\},\{1\leftarrow gcd(0,2)\}\rangle,\{\}\rangle,\\ &\langle\langle\{m\rightarrow 1\},\{\},\{1\leftarrow gcd(2,2+0)\}\rangle,\{\}\rangle,\\ &\langle\langle\{m\rightarrow z+1\},\{\},\{1\leftarrow gcd(z+z+2,2)\}\rangle,\\ &\qquad\qquad\{\langle\{m\rightarrow z\},\{\},\{1\leftarrow gcd(z+z,2)\}\rangle\}\rangle\}.\end{aligned}$$

   Since any pair of numbers $\langle e_1,e_2\rangle$ *covered* by the induction case 2 is also *covered* by case 3, case 2 can be dropped. Using this induction scheme, the conjecture is easily proved.

The next conjecture that we consider is:

$$(P_4):\quad gcd(m,s(m))=1.$$

1. *Initialize* : The LA-based cover set $\mathcal{C}_{gcd}$ is the same as before.

2. Compute the induction scheme:

   a) For the first cover set triple, the constraint equations: $\{m = 0, s(m) = x\}$ are trivially solvable. The substitution obtained is $\{m \rightarrow 0, x \rightarrow 1\}$. The induction case generated is: $\langle\langle\{m \rightarrow 0\}, \{\}, \{1 \leftarrow gcd(0,1)\}\rangle, \{\}\rangle$.

   b) The second cover set triple does not lead to an induction case since the constraint equations: $\{m = x, s(m) = 0\}$ do not have a solution.

   c) The third cover set triple does not lead to an induction case either since the constraint equations for the conclusion: $\{m = x + y, s(m) = y\}$ with context $\{x + y > x\}$ do not have a solution.

   d) For the fourth cover set triple, the constraint equations for the conclusion are: $\{m = x, s(m) = x + y\}$ with context $\{x + y > y\}$. The substitution obtained is: $\{m \rightarrow x\}$ with feasibility constraints $\{x > 0\}$. The constraint equations for the hypothesis: $\{m = x, s(m) = y\}$ with context $\{y = 1, x > 0\}$ do not have a solution and the hypothesis discarded.

The induction scheme generated is given below. The subgoal generated from the first induction case of this scheme reduces to true by rule 1 in the definition of gcd. The subgoal generated from the second case reduces to $gcd(x, 1) = 1$ which can be established by induction (using the scheme generated from the subterm $gcd(x, 1)$ using LA).

$$\{\langle\langle\{m \rightarrow 0\}, \{\}, \{1 \leftarrow gcd(0,1)\}\rangle, \{\}\rangle,$$
$$\langle\langle\{m \rightarrow x\}, \{x > 0\}, \{1 \leftarrow gcd(x, x+1)\}\rangle, \{\}\rangle.\}$$

2.4.3. *Completeness and Soundness of Schemes*

In section 2.1, we proved the completeness and the soundness of induction schemes generated from basic f-terms and complete cover sets. It was also shown in section 2.2 that the induction schemes generated by syntactic unification with the cover set triples, for a nonbasic f-term $f(t_1, \cdots, t_n)$ need not be complete even though the underlying cover set is complete. For a nonbasic f-term complete and sound schemes can be obtained if instead of syntactic unification, unification is performed modulo the equational theory E, comprising of definitions and relations amongst constructors. This is possible only if unification modulo E is decidable and finitary which is always not the case. We prove below that induction schemes got from complete LA-based cover sets using LA, and an f-term $f(t_1, \cdots, t_n)$, where t_i's are linear terms, by the algorithm in section 2.4.1, is complete and sound. The proof is along the lines of Theorem 2.

THEOREM 3. *Induction scheme ϕ generated from $t = f(t_1, \cdots, t_n)$ of a conjecture $C(\cdots, t, \cdots)$ and a complete LA-based cover set $\mathcal{C}_f$ is complete and sound.*

Proof. Completeness: Without loss of generality assume that $Vars(t) \cap Vars(\mathcal{C}_f) = \{\}$. Since $\mathcal{C}_f$ is a complete LA-based cover set, for any n-tuple $\langle e_1, \cdots, e_n\rangle$ of numbers, there exists a ground substitution β and a cover set triple, $c_i = \langle\langle s_1, \cdots, s_n\rangle,$

$\{\cdots,\langle\ s_1^k,\ \cdots, s_n^k\rangle,\cdots\}$, $cond\rangle$ such that $\beta(s) =_{LA} e$ and $\beta(cond) = true$, where $s = f(s_1,\cdots,s_n)$ and $e = f(e_1,\cdots,e_n)$. Let ϕ_i be the induction case corresponding to c$_i$ with the conclusion: $\langle\langle\delta_c, \delta(cond)\wedge c_c, \{p \leftarrow f(\delta(s_1),\cdots,\delta(s_n))\}\rangle$. δ be the *mgu* of s and t obtained by solving $CE = \{s_i = t_i | 1 \leq i \leq n\}$ under the context *cond*, using LA, with the feasibility constraints c_c. δ_c is the restriction of δ to $Vars(t)$ and p is the position of t in C.

If for some ground substitution γ, $\gamma(t) =_{LA} e$, then s and t are unifiable modulo $=_{LA}$ and $\beta' = \beta \cup \gamma$, is a unifier of s and t ($\beta'(s) =_{LA} e =_{LA} \beta'(t)$, $Vars(s) \cap Vars(t) = \{\}$) and since $\beta(cond) = true$, $\beta'(cond) = true$. Therefore, by the completeness and soundness of LA, $\beta' = \theta\circ\delta$ where $\theta(c_c) = true$, for some substitution θ. This implies that $\theta\circ\delta_c(t) =_{LA} e$ and $\theta\circ\delta(cond) = true$ and $\theta(c_c) = true$ and therefore, $\langle e_1,\cdots,e_n\rangle$ is *covered* by the induction case ϕ_i corresponding to c$_i$. Hence ϕ is complete.

Soundness: By contradiction. Assume that $C(\cdots,t,\cdots)$ is not true but the subgoal obtained from each of the induction cases of ϕ is true. Let $\langle e_1,\cdots,e_n\rangle$ be the smallest n-tuple of numbers, with respect to the semantic ordering $\succ_{LA}$ used to prove the termination of the definition of f, that is a counterexample to $C(\cdots,t,\cdots)$, i.e. $\theta\circ\delta_c(C) = false$. The induction subgoal corresponding to ϕ_i given above, $(\delta_c(C)\ \mathit{if}\ (\delta(cond)\wedge c_c))$ assuming $(\bigwedge_k(\theta_k^h(C)\ \mathit{if}\ \ (\theta_k(cond)\wedge c_c')))$ is $true$. Since $\theta(\delta_c(C))$ is $false$, $\theta(\delta(cond)) = true$ and $\theta(c_c) = true$, $\theta(\theta_k^h(C)) = false$ for some k, implying that there is a smaller counterexample to C, since $f(\theta(\theta_k(s_1^k)),\cdots,\theta(\theta_k(s_n^k)))$ is lower in the ordering $\succ_{LA}$ than e (by definition of LA-based cover set, $s\succ_{LA} f(s_1^k,\cdots,s_n^k)$ for any k and since $\succ_{LA}$ is $=_{LA}$ preserving, $f(\theta(\delta(s_1)),\cdots,\theta(\delta(s_n)))\succ_{LA} f(\theta(\delta(s_1^k)),\cdots,\theta(\delta(s_n^k))))$. A contradiction. ∎

3. Merging Induction Schemes

Often the induction schemes suggested by the various subterms of a given conjecture share induction variables amongst each other. An inductive proof attempt of the conjecture based on one of these schemes only is not likely to succeed in such cases. For instance, if $t_1 = f(x,y)$ and $t_2 = g(z,x)$ are any two subterms of a given conjecture where f and g are binary functions defined recursively on both of their arguments, then attempting a proof of the conjecture by induction based only on the scheme suggested by the term t_1, would result in an induction step with the conclusion containing $t_1' = \sigma(f(x,y)) = f(\sigma(x),\sigma(y))$ and $t_2' = \sigma(g(z,x)) = g(z,\sigma(x))$. The choice of induction scheme ensures that the term t_1' can be simplified to match the induction hypothesis but the same need not be true for the term t_2' since the variable z in t_2' does not get instantiated.

For example, based on the following definition of the *same* predicate, defining equality on *lists* built out of constructors *nil* and *cons*,

$$\begin{array}{l} 1.\ same(nil, nil) \rightarrow true, \qquad 2.\ same(nil, cons(u_1, v_1)) \rightarrow false, \\ 3.\ same(cons(u_1, v_1), nil) \rightarrow false, \\ 4. same(cons(u_1, v_1), cons(u_2, v_2)) \rightarrow (u_1 = u_2) \wedge same(v_1, v_2), \end{array}$$

a conjecture that can be attempted is:

$$(P_5):\ same(u, w)\ if\ (same(u, v)\ \wedge same(v, w)).$$

The schemes suggested by the subterms $same(u, v)$, $same(v, w)$ and $same(u, w)$ are three possible candidates for attempting a proof of the conjecture (P_5) by induction. The scheme suggested by the subterm $same(u, v)$ is given as follows.

$$\begin{array}{l} (I_4): \{\langle\langle\{u \rightarrow nil, v \rightarrow nil\}, \{\}, \{\}\rangle, \{\}\rangle, \\ \quad \langle\langle\{u \rightarrow nil, v \rightarrow cons(x_1, y_1)\}, \{\}, \{\}\rangle, \{\}\rangle, \\ \quad \langle\langle\{u \rightarrow cons(x_1, y_1), v \rightarrow nil\}, \{\}, \{\}\rangle, \{\}\rangle, \\ \quad \langle\langle\{u \rightarrow cons(x_1, y_1), v \rightarrow cons(x_2, y_2)\}, \{\}, \{\}\rangle, \{\langle\{u \rightarrow y_1, v \rightarrow y_2\}, \{\}, \{\}\rangle\}\rangle\}. \end{array}$$

The induction scheme suggested by $same(u, w)$ is very similar to the above scheme and can be obtained by replacing the variable v by the variable w in each of the induction cases of the above scheme. The induction scheme suggested by the subterm $same(v, w)$ is given as follows.

$$\begin{array}{l} (I_5):\ \{\langle\langle\{v \rightarrow nil, w \rightarrow nil\}, \{\}, \{\}\rangle, \{\}\rangle, \\ \quad \langle\langle\{v \rightarrow nil, w \rightarrow cons(x'_1, y'_1)\}, \{\}, \{\}\rangle, \{\}\rangle, \\ \quad \langle\langle\{v \rightarrow cons(x'_1, y'_1), w \rightarrow nil\}, \{\}, \{\}\rangle, \{\}\rangle, \\ \quad \langle\langle\{v \rightarrow cons(x'_1, y'_1), w \rightarrow cons(x'_2, y'_2)\}, \{\}, \{\}\rangle, \{\langle\{v \rightarrow y'_1, w \rightarrow y'_2\}, \{\}, \{\}\rangle\}\rangle\}. \end{array}$$

Attempting a proof of (P_5) by induction based on only one of the above schemes such as (I_4), would result in the induction step case with the conclusion,

$$same(cons(x_1, y_1), w)\ if\ same(cons(x_1, y_1), cons(x_2, y_2)) \wedge same(cons(x_2, y_2), w),$$

and the hypothesis, $same(y_1, w)\ if\ same(y_1, y_2) \wedge same(y_2, w)$. The conclusion simplifies using the last rule in the definition of $same$ to,

$$same(cons(x_1, y_1), w)\ if\ (x_1 = x_2) \wedge same(y_1, y_2) \wedge same(cons(x_2, y_2), w).$$

Since the conclusion cannot be simplified any further, the hypothesis does not match the conclusion and the proof attempt by induction fails.[10] The failure is due to the induction variable w not being instantiated in the subterms $same(u, w)$ and $same(v, w)$ of (P_5) in the conclusion, as discussed before. The reader can easily verify that a proof attempt of (P_5) based on the other two schemes would also fail

[10] Since the definitions are constructor-based a complete induction scheme can be generated from the non-basic term $same(cons(x_1, y_1), w)$ and a further induction based on this scheme could alternatively be used to establish the conjecture (P_5).

since one of the variables u or v would remain uninstantiated in the induction step of these proof attempts.

This situation can be remedied if the above two induction schemes (I_4) and (I_5) are merged and an induction scheme which instantiates the induction variables simultaneously in all the terms is generated. Further, merging of schemes also eliminates the need to arbitrarily choose from amongst competing schemes. We merge the induction scheme (I_4) into the scheme (I_5) by merging each induction case of (I_4) into as many cases of (I_5) as possible based on the following definition of merging of induction cases.

DEFINITION. *Mergeable Induction Cases:* An induction case $i_1 = \langle\langle \sigma_c, cond_c, repl_c\rangle, \{\cdots, \langle\ \theta_i, cond_i,\ repl_i\rangle, \cdots\}\rangle$ merges with an induction case $i_2 = \langle\langle \sigma'_c, cond'_c,\ repl'_c\rangle, \{\cdots,\ \langle\theta'_j, cond'_j,\ repl'_j\rangle, \cdots\}\rangle$ to give an induction case of the form $\langle\langle\sigma,\ mcond_c,\ mrepl_c\rangle, MH\rangle$ if the following conditions hold.

- $\sigma_c(x)$ is unifiable with $\sigma'_c(x)$ with the mgu δ for all $x \in Vars(\sigma_c) \cap Vars(\sigma'_c)$. $\sigma = (\delta \circ \sigma_c) \cup (\delta \circ \sigma'_c)$; $mcond_c = \delta(cond_c) \cup \delta(cond'_c)$; $mrepl_c = \delta(repl_c) \cup \delta(repl'_c)$.
- The $(i,j)^{th}$ merged hypothesis is:[11] For each triple, $\langle\theta_i, cond_i, repl_i\rangle$ of i_1 the triple $\langle(\delta \circ \theta_i) \cup \gamma'_j, \delta(cond_i) \cup \delta(cond'_j), \delta(repl_i)\rangle$ is included in MH if it is consistent. If the second component of i_2 is empty (a basis case) then γ'_j is the restriction of σ'_c on variables appearing only in i_2. Otherwise, γ'_j is the restriction of θ'_j on such variables.

  Similarly, for each triple, $\langle\theta'_j, cond'_j, repl'_j\rangle$ of i_2, the triple $\langle(\delta \circ \theta'_j) \cup \gamma_i, \delta(cond'_j) \cup \delta(cond_i), \delta(repl'_j)\rangle$ is included in MH if it is consistent. If the second component of i_1 is empty then γ_i is the restriction of σ_c on variables appearing only in i_1. Otherwise, γ_i is the restriction of θ_i on such variables.

In order to merge (I_4) into (I_5) we consider each induction case of (I_4) with all induction cases of (I_5). The first induction case of (I_4) merges only with the first two induction cases of (I_5) since for the other cases, the substitutions for the shared variable v, nil and $cons(x, y)$, cannot be unified. The merged cases are:

$$\langle\langle\{u \to nil, v \to nil, w \to nil\}, \{\}, \{\}\rangle, \{\}\rangle,$$
$$\langle\langle\{u \to nil, v \to nil, w \to cons(x'_1, y'_1)\}, \{\}, \{\}\rangle, \{\}\rangle.$$

The second induction case of (I_4) merges only with the last two induction cases of (I_5) for the same reasons as above and the merged cases are:

$$\langle\langle\{u \to nil, v \to cons(x_2, y_2), w \to nil\}, \{\}, \{\}\rangle, \{\}\rangle,$$
$$\langle\langle\{u \to nil, v \to cons(x_2, y_2), w \to cons(x'_2, y'_2)\}, \{\}, \{\}\rangle,$$
$$\{\langle\{u \to nil, v \to y_2, w \to y'_2\}, \{\}, \{\}\rangle\}\rangle.$$

[11] It is assumed that the same reduction ordering is used for proving the termination of the two definitions, the induction schemes corresponding to which are being merged. Otherwise, the merged induction scheme need not be sound since the hypotheses of both the schemes being merged are included in the hypotheses of the merged schemes.

Similarly for the third induction case of (I_4) the merged cases obtained are:

$$\langle\langle\{u \rightarrow cons(x_1, y_1), v \rightarrow nil, w \rightarrow nil\}, \{\}, \{\}\rangle, \{\}\rangle,$$
$$\langle\langle\{u \rightarrow cons(x_1, y_1), v \rightarrow nil, w \rightarrow cons(x'_2, y'_2)\}, \{\}, \{\}\rangle, \{\}\rangle.$$

The final induction case of (I_4) merges with the third and the fourth induction cases of (I_5) and the merged induction cases obtained are:

$$\langle\langle\{u \rightarrow cons(x_1, y_1), v \rightarrow cons(x_2, y_2), w \rightarrow nil\}, \{\}, \{\}\rangle,$$
$$\{\langle\{u \rightarrow y_1, v \rightarrow y_2, w \rightarrow nil\}, \{\}, \{\}\rangle\}\rangle,$$
$$\langle\langle\{u \rightarrow cons(x_1, y_1), v \rightarrow cons(x_2, y_2), w \rightarrow cons(x'_2, y'_2)\}, \{\}, \{\},$$
$$\{\langle\{u \rightarrow y_1, v \rightarrow y_2, w \rightarrow y'_2\}, \{\}, \{\}\rangle\}\rangle.$$

Note that while merging the final case of (I_4) with the final case of (I_5) the two merged hypotheses are identical and hence only one of them is retained. Since, for $\delta = \{x'_1 \rightarrow x_2, y'_1 \rightarrow y_2\}$, $\theta_1 = \{u \rightarrow y_1, v \rightarrow y_2\}$ and $\theta'_2 = \{v \rightarrow y'_1, w \rightarrow y'_2\}$, the substitutions, $(\delta \circ \theta_1) \cup \gamma'_2 = \{u \rightarrow y_1, v \rightarrow y_2, w \rightarrow y'_2\} = (\delta \circ \theta'_2) \cup \gamma_1$, where $\gamma_1 = \{u \rightarrow y_1\}$, is the restriction of θ_1 on variables occurring only in (I_4) and $\gamma'_2 = \{w \rightarrow y'_2\}$, is the restriction of θ'_2 on variables occurring only in (I_5). Using the above merged cases the conjecture (P_5) is easily proved. The above procedure for merging induction schemes preserves soundness and completeness as illustrated by the following theorem.

THEOREM 4. *An induction scheme ψ obtained by merging any two complete and sound basic induction schemes ϕ_f and ϕ_g which are mergeable is complete and sound.*

Proof. Completeness: Let ϕ_f and ϕ_g be respectively generated from the terms $t = f(x_1, \cdots, x_n)$ and $s = g(y_1, \cdots, y_k)$ of a conjecture $C(\cdots, t, \cdots, s, \cdots)$. Since the induction schemes ϕ_f and ϕ_g are complete, for any $n+k$-tuple $\langle e_1, \cdots, e_{n+k}\rangle$ of ground constructor terms, there exist induction cases: $\phi^i_f = \langle\langle \sigma_c, cond_c, repl_c\rangle$, $\{\cdots, \langle\theta_h, cond_h, repl_h\rangle, \cdots\}\rangle$ and $\phi^j_g = \langle\langle\sigma'_c, cond'_c, repl'_c\rangle, \{\cdots, \langle\theta'_h, cond'_h, repl'_h\rangle,$ $\cdots\}\rangle$ such that for some ground substitutions γ and γ', $\gamma \circ \sigma_c(t) = e$, $\gamma(cond_c)$ $= true$ and $\gamma' \circ \sigma'_c(s) = e'$, $\gamma(cond'_c) = true$ where $e = f(e_1, \cdots, e_n)$ and $e' =$ $g(e_{n+1}, \cdots, e_{n+k})$.

Without loss of generality, we assume ϕ^i_f and ϕ^j_g to be mergeable induction cases. Let $\beta = \gamma \cup \gamma'$; β agrees on its substitution on common variables in t and s. For a common variable $x_i = y_j$, $\sigma_c(x_i)$ and $\sigma'_c(y_j)$ are unifiable. $\beta(cond_c) = \beta(cond'_c) =$ $true$. The merged induction case $\psi_{(i,j)}$ generated from ϕ^i_f and ϕ^j_g, is $\langle\langle\sigma$, $mcond_c$, $mrepl_c\rangle$, $\{\cdots, \langle\ \beta_{(i,j)}, mcond_{(i,j)}, mrepl_{(i,j)}\ \rangle, \cdots\}\rangle$, where $\sigma = (\delta \circ \sigma_c) \cup (\delta \circ \sigma'_c)$ with δ being the *mgu* of $\sigma_c(x)$ and $\sigma'_c(x)$, $x \in Vars(t) \cap Vars(s)$, and $mcond_c =$ $\delta(cond_c) \cup \delta(cond'_c)$. Thus $\beta = \theta \circ \delta$ for some substitution θ. Since $\theta \circ \sigma(t) = e$, and $\theta \circ \sigma(s) = e'$, and $\theta(mcond_c) = \theta \circ \delta(cond_c) \wedge \theta \circ \delta(cond'_c) = true$, the $n+k$-tuple, $\langle e_1, \cdots e_{n+k}\rangle$ is *covered* by the induction case $\psi_{(i,j)}$. Therefore, the induction scheme ψ is complete.

Soundness: By contradiction. Assuming a counterexample made of ground constructor terms serving as instances of t and s, a smaller counterexample is constructed in which either an instance of t or an instance of s is smaller. Assume that the conjecture $C(\cdots, t, \cdots, s, \cdots)$ is not true but the subgoal obtained from each induction case of ψ is true. Let $\langle e_1, \cdots, e_{n+k}\rangle$ be the smallest $n+k$-tuple of ground constructor terms with respect to the ordering $\succ$ used to prove the termination of the definitions of g and f, that is a counterexample to C, i.e $\theta \circ \sigma(C)$ $= \mathit{false}$. The induction subgoal obtained from $\psi_{(i,j)}$, $(\sigma(C)\ \mathit{if}\ \mathit{mcond}_c)$ assuming $(\bigwedge_{(i,j)} \beta_{(i,j)}(C)\ \mathit{if}\ \mathit{mcond}_{(i,j)})$ is true. Since $\theta(\sigma(C)) = \mathit{false}$, and $\theta(\mathit{mcond}_c) = \mathit{true}$, one of $\theta(\beta_{(i,j)}(C))$ is false. But each $\beta_{(i,j)}$ (by definition mergeable induction cases) is either of the form $(\delta \circ \theta_h) \cup \theta_1$, or is of the form $(\delta \circ \theta'_h) \cup \theta_2$, both of which imply the existence of a counterexample $\langle e'_1, \cdots, e'_{n+k}\rangle$ to C which is lower in the ordering $\succ$ than $\langle e_1, \cdots, e_{n+k}\rangle$, where θ_1 and θ_2 are respectively the restrictions of θ'_h and θ_h on variables occurring only in them. By definition of cover set, $\sigma_c(t) \succ \theta_h(t)$ and $\sigma'_c(s) \succ \theta'_h(s)$ and for any hypothesis obtained from ϕ^i_f, $\theta(\sigma(t)) \succ \theta(\delta \circ \theta_h(t))$, i.e. $e \succ f(e'_1, \cdots, e'_n)$ and $\theta(\sigma(s)) \succ \theta(((\delta \circ \theta_h) \cup \theta_1)(s))$, since for any variable y of s which does not occur in t, $\theta(\sigma'_c(y)) \succ \theta(\theta'_h(y))$ and for a common variable y' of s and t, $\theta(\delta \circ \sigma'_c(y')) \succ (\delta \circ \theta_h(y'))$, since $\delta \circ \sigma'_c(y') = \delta \circ \sigma_c(y')$. Thus, $\langle e, e'\rangle \succ$ $\langle f(e'_1, \cdots, e'_n), g(e'_{n+1}, \cdots, e'_{n+k})\rangle)$ for the hypothesis got from ϕ^i_f. Similar result follows for the hypothesis got from ϕ^j_g by symmetry. A contradiction. ∎

3.1. Merging using Linear Arithmetic

As discussed in the previous subsection, one of the crucial preconditions for merging two induction cases is to have syntactically unifiable substitutions for the common induction variables; the merged induction case is obtained from the *mgu* of these substitutions. If the definitions involved are not constructor-based or if there are relations among the constructors (E is not empty) then the merged scheme obtained by the above described procedure using syntactic unification need not be complete. For such cases, the substitutions for the common induction variables in the induction cases have to be semantically unified (with respect to $=_E$).[12]

For example, in $P_2 : \mathit{divides}(2, x) = \mathit{not}(\mathit{divides}(2, s(x)))$, the induction schemes suggested by $\mathit{divides}(2,\ x)$ and that suggested by $\mathit{divides}(2, s(x))$, given at the end of section 2.4, cannot be merged using the above procedure to obtain a complete and sound scheme since the substitution $\{x \to v + 2\}$ in the scheme of $\mathit{divides}(2, x)$ is not unifiable with the substitution for x in any of cases of the scheme of $\mathit{divides}(2, s(x))$ and similarly the substitution $\{x \to v' + 1\}$ in the latter scheme is irreconcilable with any of the cases of the former scheme.

[12] This can be done provided unification modulo $=_E$ is decidable and finitary. Further, merged induction cases corresponding to each *mgu* of the substitutions of the common induction variables need to be generated for the merged scheme to be complete.

However these two induction schemes can be merged if we use linear arithmetic to reconcile these substitutions. The linear arithmetic procedure is again used to perform the semantic unification of the substitutions of the common induction variables amongst the schemes being merged. For soundness, the same semantic ordering $\succ_{LA}$ must be used to prove the termination of the definitions of the functions from which the schemes are derived.

We illustrate the merging algorithm by merging the scheme for $divides(2, x)$ with the scheme for $divides(2, s(x))$. The precise details are given in the algorithm in subsection 3.1.1. The first element of the scheme for $divides(2, x)$ merges only with the first element of $divides(2, s(x))$ to give: $\langle\langle\{x \rightarrow 0\}, \{\}, \{1 \leftarrow divides(2, 0), 2.1 \leftarrow divides(2, s(0))\}\rangle, \{\}\rangle$.

The second element of the scheme for $divides(2, x)$ merges only with the second element of $divides(2, s(x))$ to give: $\langle\langle\{x \rightarrow 1\}, \{\}, \{1 \leftarrow divides(2, 1), 2.1 \leftarrow divides(2, 2 + 0)\}\rangle, \{\}\rangle$.

Note that the hypothesis $\langle\{x \rightarrow v' - 1\}, \{v' \geq 1\}, \{2.1 \leftarrow divides(2, v')\}\rangle$ is discarded since the constraint equations of the induction conclusions of the two elements being merged are $\{x = 1, x = v' + 1\}$ which implies $v' = 0$ and the condition $v' \geq 1$ governing the hypothesis is unsatisfiable.

Finally, the third element of $divides(2, x)$ merges only with the second element of $divides(2, s(x))$ to give: $\langle\langle\{x \rightarrow 2 + v\}, \{\}, \{1 \leftarrow divides(2, 2 + v), 2.1 \leftarrow divides(2, 2 + (v + 1))\}, \{\langle\{x \rightarrow v\}, \{\}, \{1 \leftarrow divides(2, v), 2.1 \leftarrow divides(2, v + 1)\}\rangle\}\rangle$. Note that to obtain the above merged induction scheme element, we reconcile the two substitutions of x: $x \rightarrow v + 2$ and $x \rightarrow v' + 1$ producing $v' = v + 1$.

As was shown in the introduction, using the resulting induction scheme (I_3) the conjecture P_2 can be easily proved.

3.1.1. *Linear Arithmetic based Algorithm for Merging Induction Schemes*

Below we give an algorithm using LA for merging induction schemes that share common induction variables of number type. The algorithm can be generalized to consider any arbitrary E by using $=_E$ unification instead of semantic unification using LA. Of course, in order to do so, $=_E$ unification must be decidable and finitary as mentioned before.

- **Input**: Induction schemes I_1 and I_2 which meet the following conditions.
    1. $indvars(I_1) \cap indvars(I_2) \neq \{\}$.
    2. The substitutions are over the theory of linear arithmetic for the common induction variables of I_1 and I_2.
- **Output** : An induction scheme I_{out}.
- **Method**:
    1. *Merge two distinct schemes I_1 and I_2*: For each induction case: $i_1 = \langle\langle\sigma_c, cond_c, repl_c\rangle, \{\cdots, \langle\theta_i, cond_i, repl_i\rangle, \cdots\}\rangle$ of I_1 do

a) For each induction case:
   $i_2 = \langle\langle\sigma'_c, cond'_c, repl'_c\rangle, \{\cdots, \langle\theta'_j, cond'_j, repl'_j\rangle, \cdots\}\rangle$ of I_2 do

   i) Let $CE = \sigma_c \cup \sigma'_c$ where σ_c and σ'_c are viewed as equations. Solve CE using LA with context $\{cond_c, cond'_c\}$. If solvable, let $\delta = \{x_i \rightarrow s_i | x_i \in Vars(\sigma_c) \cup Vars(\sigma'_c), x_i \notin s_i\}$ with feasibility constraints c_c. CE being unsolvable, corresponds to no merged induction case being generated.

   ii) Compute the merged induction scheme element $\langle\langle\sigma, mcond_c, mrepl_c\rangle, MH\rangle$, where

   A) σ is the union of the substitutions $\delta \circ \sigma_c$ and $\delta \circ \sigma'_c$; $mcond_c$ is the union of $\delta(cond_c)$, $\delta(cond'_c)$ and $\delta(c_c)$; $mrepl_c$ is the union of the two replacements $\delta(repl_c)$ and $\delta(repl'_c)$.

   B) For each triple $\langle\theta_i, cond_i, repl_i\rangle$ in the hypotheses of i_1, the triple $\langle(\delta \circ \theta_i) \cup \gamma'_j, \delta(cond_i) \cup \delta(cond'_j), \delta(repl_i)\rangle$ is included in MH if $(\delta \circ \theta_i) \cup \gamma'_j$ when viewed as equations is consistent with respect to LA with context $\{cond_i, cond'_j\}$. If the second component of i_2 is empty then γ'_j is restriction of σ'_c on variables occurring only in i_2. Otherwise, γ'_j is the restriction of θ'_j on such variables.
   Similarly, for each triple $\langle\theta'_j, cond'_j, repl'_j\rangle$ in the hypotheses of i_2, the triple $\langle(\delta \circ \theta'_j) \cup \gamma_i,\ \delta(cond'_j) \cup \delta(cond_i),\ \delta(repl'_j)\rangle$ is included in MH if $(\delta \circ \theta'_j) \cup \gamma_i$ when viewed as equations is consistent with respect to LA with context $\{cond'_i, cond'_j\}$. If the second component of i_1 is empty, then γ_i is the restriction of σ_c on variables occurring only in i_1. Otherwise, γ_i is the restriction of θ_i on such variables.

2. Return the set of merged cases to be the induction scheme I_{out}.

3.1.2. *Examples*

We illustrate the algorithm on an example based on the definition of *gcd* given earlier and the following definition of *even* predicate on natural numbers.

1. $even(0) \rightarrow true$, 2. $even(1) \rightarrow false$, 3. $even(s(s(u))) \rightarrow even(u)$.

Consider proving the conjecture,

$$(P_6): \quad gcd(m, 2) = 2 \;\; if \;\; even(m).$$

The scheme suggested by $even(m)$ at position 2.2 in (P_6) is:

$$\{\langle\langle\{m \rightarrow 0\}\rangle, \{\}, \{2.2 \leftarrow even(0)\}\rangle, \{\}\rangle, \langle\langle\{m \rightarrow 1\}\rangle, \{\}, \{2.2 \leftarrow even(s(0))\}\rangle, \{\}\rangle,$$
$$\langle\langle\{m \rightarrow s(s(u))\}, \{\}, \{2.2 \leftarrow even(s(s(u)))\}\rangle,$$
$$\{\langle\{m \rightarrow u\}\rangle, \{\}, \{2.2 \leftarrow even(u)\}\rangle\}\rangle\},$$

and the scheme suggested by the $gcd(m,2)$ at position 1 in (P_6) using LA is:

$$\{\langle\langle\{m \rightarrow 0\},\{\},\{1 \leftarrow gcd(0,2)\}\rangle,\{\}\rangle,\langle\langle\{m \rightarrow 1\},\{\},\{1 \leftarrow gcd(1,1+1)\}\rangle,\{\}\rangle,$$
$$\langle\langle\{m \rightarrow 2\},\{\},\{1 \leftarrow gcd(2,2+0)\}\rangle,\{\}\rangle,$$
$$\langle\langle\{m \rightarrow x+2\},\{\},\{1 \leftarrow gcd(x+2,2)\}\rangle,\{\langle\{m \rightarrow x\},\{\},\{1 \leftarrow gcd(x,2)\}\rangle\}\rangle\}.$$

Merging the scheme suggested by the subterm $even(m)$ into the scheme suggested by the subterm $gcd(m,2)$ is done as follows.

1. The first induction case of $even(m)$ merges, only with the first induction case of $gcd(m,2)$. For the other cases, the constraint equations: $\{0 = 2\}$, $\{0 = 1\}$, and $\{0 = x+2\}$ respectively, are unsolvable. The merged case is: $\langle\langle\{m \rightarrow 0\},\{\},\{1 \leftarrow gcd(0,2), 2.2 \leftarrow even(0)\}\rangle,\{\}\rangle$.

2. The second induction case of $even(m)$ merges only with the second induction case of $gcd(m,2)$ since the constraint equations are unsolvable for the rest of the cases. The merged case is: $\langle\langle\{m \rightarrow 1\}, \{\},\{1 \leftarrow gcd(1,1+1),\ 2.2 \leftarrow even(1)\}\rangle,\ \{\}\rangle$.

3. For the third induction case of $even(m)$ with the third induction case of $gcd(m,2)$ the constraint equations for the conclusion are: $\{m = 2, m = s(s(u))\}$. The substitution obtained is: $\{m \rightarrow 2, u \rightarrow 0\}$. The constraint equations for the hypothesis, $\{m = u, u = 0\}$ are trivially solvable. The merged case obtained is $\langle\langle\{m \rightarrow 2\},\{\},\{1 \leftarrow gcd(2,2+0),\ 2.2 \leftarrow even(2)\}\ \rangle,\{\langle\{m \rightarrow 0\},\{\},\{2.2 \leftarrow even(0)\}\rangle\}\rangle$.

4. For the third induction case of $even(m)$ with the final induction case of $gcd(m,2)$, the constraint equations for the conclusion are: $\{m = x+2, m = s(s(u))\}$. The substitution obtained is: $\{u \rightarrow x\}$. The constraint equations for the hypothesis are trivially solvable and the merged induction case obtained is: $\langle\langle\{m \rightarrow x+2\},\{\},\ \{1 \leftarrow gcd(x+2,2), 2.2 \leftarrow even(s(s(x)))\ \}\rangle,\{\langle\{m \rightarrow x\},\{\},\{1 \leftarrow gcd(x,2),\ 2.2 \leftarrow even(x)\}\rangle\}\rangle$.

Using the merged induction cases the conjecture is easily proved.

As the next example consider the following conjecture from [11],

$$(P_7): \quad quot(m,4) = hf(hf(m)),$$

based on the following two definitions, $quot$ which computes the quotient of dividing a natural number x by a natural number y and hf which halves a natural number,

1. $quot(x,0) \rightarrow 0$, 2. $quot(x,y) \rightarrow 0\ if\ (x < y)$,
3. $quot(x+y,y) \rightarrow s(quot(x,y))\ if\ (y \neq 0)$.
1. $hf(0) \rightarrow 0$, 2. $hf(s(0)) \rightarrow 0$, 3. $hf(s(s(u))) \rightarrow s(hf(u))$.

The induction scheme obtained from $hf(m)$ at position 2.1 in (P_7) is:

$$\{\langle\langle\{m \to 0\}, \{\}, \{2.1 \leftarrow hf(0)\}\rangle, \{\}\rangle, \langle\langle\{m \to s(0)\}, \{\}, \{2.1 \leftarrow hf(s(0))\}\rangle, \{\}\rangle,$$
$$\langle\langle\{m \to s(s(u))\}, \{\}, \{2.1 \leftarrow hf(s(s(u)))\}\rangle,$$
$$\{\langle\{m \to u\}, \{\}, \{2.1 \leftarrow hf(u)\}\rangle\},$$

and that obtained from $quot(m, 4)$ at position 1 in (P_7) using LA is:

$$\{\langle\langle\{m \to 0\}, \{\}, \{1 \leftarrow quot(0, 4)\}\rangle, \{\}\rangle, \langle\langle\{m \to 1\}, \{\}, \{1 \leftarrow quot(1, 4)\}\rangle, \{\}\rangle,$$
$$\langle\langle\{m \to 2\}, \{\}, \{1 \leftarrow quot(2, 4)\}\rangle, \{\}\rangle, \langle\langle\{m \to 3\}, \{\}, \{1 \leftarrow quot(3, 4)\}\rangle, \{\}\rangle,$$
$$\langle\langle\{m \to x + 4\}, \{\}, \{1 \leftarrow quot(x + 4, 4)\}\rangle,$$
$$\{\langle\{m \to x\}, \{\}, \{1 \leftarrow quot(x, 4)\}\rangle\}\rangle\}.$$

The scheme suggested by the subterm $hf(m)$ is merged into the scheme suggested by the subterm $quot(m, 4)$ as follows.

1. The first two cases of $hf(m)$ merge only with the first two cases of $quot(m, 4)$ resulting in $\langle\langle\{m \to 0\}, \{\}, \{\}\rangle, \{\}\rangle$ and $\langle\langle\{m \to s(0))\}, \{\}, \{\}\rangle, \{\}\rangle$ respectively.

2. The last case of $hf(m)$ merges with the third and the fourth cases of $quot(m, 4)$, resulting in $\langle\langle\{m \to 2\}, \{\}, \{1 \leftarrow quot(2, 4), 2.1 \leftarrow hf(s\ (s(0)))\ \}\rangle, \langle\{m \to 0\}, \{\}, \{1 \leftarrow quot(0, 4), 2.1 \leftarrow hf(0)\}\rangle\}\rangle$, and $\langle\langle\{m \to 3\}, \{\}, \{1 \leftarrow quot(3, 4), 2.1 \leftarrow hf(s(s(s(0))))\}\rangle, \langle\{m \to 1\}, \{\}, \{1 \leftarrow quot(1, 4), 2.1 \leftarrow hf(s(0))\}\rangle\}\rangle$ respectively.

3. For the last case, of $hf(m)$ with the final case of $quot(m, 4)$ the constraint equations for the conclusion are: $\{m = s(s(u)), m = x + 4\}$. The substitution obtained is: $\{m \to x + 4, u \to x + 2\}$. The constraint equations set up for the two individual hypotheses are: $\{m = x, u = x+2\}$, resulting in the substitution $\{m \to x\}$ and $\{m = u, u = x + 2\}$, resulting in the substitution $\{m \to x + 2\}$. Two hypotheses are generated corresponding to these two substitutions. The merged induction case generated is:
$$\{\langle\langle\{m \to x + 4\}, \{\}, \{1 \leftarrow quot(x + 4, 4), 2.1 \leftarrow hf(s(s(x + 2)))\}\rangle,$$
$$\{\langle\{m \to x\}, \{\}, \{1 \leftarrow quot(x, 4), 2.1 \leftarrow hf(x)\}\rangle,$$
$$\langle\{m \to x + 2\}, \{\}, \{1 \leftarrow quot(x + 2, 4)2.1 \leftarrow hf(x + 2)\}\rangle\}.$$

As should be evident from the induction step, there is an induction hypothesis being contributed by each of the induction schemes corresponding to $quot(m, 4)$ and $hf(m)$. Using the above merged induction cases the conjecture is easily proved.

3.1.3. *Completeness and Soundness of a Merged Induction Scheme*

The completeness and soundness of the induction scheme generated by the above algorithm for merging sound and complete induction schemes using LA for semantic unification is established in the same way as was done earlier for completeness and soundness of merging of basic inductions schemes. Instead of unification over $=_E$, unification modulo LA is done. For each individual induction scheme, the proof is patterned after that of Theorem 3, establishing completeness and soundness of of an induction scheme generated using LA for semantic unification. So the following proof combines proofs of Theorems 3 and 4.

THEOREM 5. *The induction scheme ψ, obtained by merging any two sound and complete induction schemes ϕ_f and ϕ_g corresponding to complete LA-based cover sets $\mathcal{C}_f$ and $\mathcal{C}_g$ respectively, is complete and sound.*

Proof. Completeness: Let ϕ_f and ϕ_g be respectively generated from $t = f(t_1, \cdots, t_n)$ and $s = g(s_1, \cdots, s_k)$ of a conjecture $C(\cdots, t, \cdots, s, \cdots)$. For any $n+k$-tuple $\langle e_1, \cdots, e_{n+k}\rangle$ of numbers such that for some ground substitutions, β and β', $\beta(t) =_{LA} e$ and $\beta'(s) = e'$, where $e = f(e_1, \cdots, e_n)$ and $e' = g(e_{n+1}, \cdots, e_{n+k})$, there exist induction cases $\phi_f^i = \langle\langle\sigma_c, cond_c, repl_c\rangle, \{\cdots, \langle\theta_h, cond_h, repl_h\rangle, \cdots\}\rangle$ and $\phi_g^j = \langle\langle\sigma'_c, cond'_c, repl'_c\rangle, \{\cdots, \langle\theta'_h, cond'_h, repl'_h\rangle, \cdots\}\rangle$ which *cover* $\langle e_1, \cdots, e_{n+k}\rangle$. Thus for some ground substitutions γ and γ', $\gamma \circ \sigma_c(t) =_{LA} e$, $\gamma(cond_c) = true$ and $\gamma' \circ \sigma'_c(s) =_{LA} e'$, $\gamma'(cond'_c) = true$,

Let $\beta'' = \gamma \cup \gamma'$; β'' agrees on its substitution on common variables in t and s. For a common variable x, $\sigma_c(x)$ and $\sigma'_c(x)$ are unifiable modulo LA. $\beta''(cond_c) = \beta''(cond'_c) = true$. It follows along the same lines as Theorem 4 that the $n+k$-tuple, $\langle e_1, \cdots, e_{n+k}\rangle$ is *covered* by the induction case, $\psi_{(i,j)}$, generated from ϕ_f^i and ϕ_g^j by the algorithm 3.1.1. and therefore, the induction scheme ψ is complete.

Soundness: By contradiction. Assuming a counterexample of numbers serving as instances of t and s, a smaller counterexample is constructed in the same manner as in Theorem 4, in which either an instance of t or an instance of s is smaller. Let $\langle e_1, \cdots, e_{n+k}\rangle$ be the smallest $n+k$-tuple of numbers with respect to the ordering $\succ_{LA}$ used to prove the termination of the definitions of g and f, that is a counterexample to C. Since $\theta(\sigma(C)) = false$, and $\theta(mcond_c) = true$, one of $\theta(\beta_{(i,j)}(C))$ is $false$. But each $\beta_{(i,j)}$ computed by the algorithm in section 3.1.1 is either of the form $(\delta \circ \theta_h) \cup \theta_1$, or is of the form $(\delta \circ \theta'_h) \cup \theta_2$ both of which imply the existence of a counterexample $\langle e'_1, \cdots, e'_{n+k}\rangle$ of numbers to C which is lower in the ordering $\succ_{LA}$ than $\langle e_1, \cdots, e_{n+k}\rangle$, where θ_1 and θ_2 are respectively the restrictions of θ'_h and θ_h on variables occurring only in them. A contradiction. ■

4. Checking Completeness of Cover Sets

An important property of a cover set is that it be a complete cover for the data structure under consideration. As proved earlier, an induction scheme generated using a complete cover set is sound. This is ensured if the terminating rewrite rules defining a function whose left sides are used as the basis for constructing the cover set, completely define the function. In [5, 8], algorithms for checking completeness of constructor-based definitions expressed as terminating rewrite rules are discussed; for background information on this topic, an interested reader may also consult [6]. For LA-based definitions the linear arithmetic procedure can be used for checking the completeness. This fills a major gap in the use of the cover set method for mechanizing induction in *RRL.* For example, in the proof of unique prime factorization theorem reported in [13], the completeness of the definitions of some of the functions (such as div, rem, gcd, $primefac$, etc.) and the associated cover sets was established manually. Using the linear arithmetic procedure, all these proofs can be carried out automatically in *RRL.* We develop an algorithm for checking completeness of function definitions and the associated cover sets using the linear arithmetic procedure. We focus on function definitions in which all arguments are of number type; the method can be easily extended to consider definitions taking arguments from numbers as well as other data structures such as lists and sequences. This algorithm serves as a decision procedure for checking the completeness of LA-based cover sets and thus could be used as a heuristic for checking the completeness of LA-based function definitions.

4.1. Algorithm to Check Completeness of a Function Definition

- **Input** : An LA-based cover set $\mathcal{C}_f$ of an n-ary function $f(x_1, \cdots, x_n)$.
- **Output** : A quantifier-free linear arithmetic formula ξ with free variables $x_1, \cdots, x_n$, specifying the values over which f is defined; true if $\mathcal{C}_f$ is complete.
- **Method** :

```
1. Initialize : Initialize $\xi$ to be $false$.
2. Linear Solve : For each cover set triple
   $c_i = \langle\langle s_1, \cdots, s_n\rangle, \{\cdots, \langle s'_1, \cdots s'_n\rangle, \cdots\}, cond\rangle$ do:
     a) Set up constraint equations $CE = \{x_i = s_i \mid 1 \leq i \leq n\}$.
     b) Solve $CE$ by $LA$ with context $cond$. If solvable, let $c_c$ be a quantifier-
        free formula over $x_i$'s under which solutions exist. If no solution exists
        then $c_c$ is assumed to be false.
     c) $\xi := \xi \vee c_c$.
3. Linear Simplify : Using $LA$, check whether $\xi$ is valid. If so then return
   $true$. Otherwise, return the simplified form of $\xi$.
```

As the following theorem shows, if the above algorithm returns $true$, then the definition of f is complete, and hence, the associated cover set is also complete. If the algorithm returns a formula different from $true$, then the definition of f may or may not be complete depending upon whether the rewrite rules defining f are confluent or not. However, the associated cover set is incomplete.

THEOREM 6. *An LA-based cover set $\mathcal{C}_f$ of a function $f(x_1, \cdots, x_n)$ is complete iff the above algorithm returns $true$.*

Proof. If an LA-based cover set $\mathcal{C}_f$ of an n-ary function f is complete, for any n-tuple $\langle e_1, \cdots, e_n\rangle$ of numbers, there exist a substitution σ and a cover set triple $\mathrm{c}_i = \langle\langle s_1, \cdots, s_n\rangle, \{\cdots, \langle s'_1, \cdots, s'_n\rangle \cdots\}, cond\rangle$ such that $\sigma(s_i) =_{LA} e_i$, $1 \leq i \leq n$, and $\sigma(cond) = true$. So the formula c_c generated by the algorithm for that cover set tuple at step $2(b)$ when instantiated with $\{x_1 \rightarrow e_1, \cdots, x_n \rightarrow e_n\}$ is $true$, making $\xi[x_1 \rightarrow e_1, \cdots, x_n \rightarrow e_n]$ to be true. This implies that the quantifier-free formula ξ with free variables $x_1, \cdots, x_n$ generated by the algorithm is equivalent to $true$.

If $\mathcal{C}_f$ is not complete, there exists an n-tuple $\langle e_1, \cdots, e_n\rangle$ that is not covered by any cover set triple, i.e. for each cover set triple c_i, the set of equations $CE = \{e_i = s_i\}$ in step $2(a)$ of the above algorithm does not have a solution under $cond$, i.e. the corresponding $c_c[x_1 \rightarrow e_1, \cdots x_n \rightarrow e_n]$ is false. Therefore, $\xi[x_1 \rightarrow e_1, \cdots, x_n \rightarrow e_n]$ is false, implying that ξ cannot be equivalent to $true$. ■

4.2. Examples

– As the first example, consider the definition of gcd given earlier.

1. *Initialize*: Let $\xi = false$ and $\mathcal{C}_{gcd}$ be the cover set of $gcd(x_1, x_2)$.
2. *Linear Solve*: For the first two cover set triples of $\mathcal{C}_{gcd}$ the constraint equations are: $\{x_1 = x, 0 = x_2\}$ and $\{x_1 = 0, x_2 = x\}$ respectively. Hence ξ is updated to be: $\xi = (x_1 = 0) \vee (x_2 = 0)$.
   For the third cover set triple, the constraint equations are: $\{\ x + y = x_1,\ y = x_2\ \}$ with context $x + y > x$. c_c is $x_1 \geq x_2 \wedge x_2 > 0$. Adding these conditions : $\xi = (x_1 = 0) \vee (x_2 = 0) \vee (x_1 \geq x_2 \wedge x_2 > 0)$.
   For the last cover set triple by symmetry, c_c is $x_2 \geq x_1 \wedge x_1 > 0$. Adding these two conditions to ξ, we obtain
   $$\xi = (x_1 = 0) \vee (x_2 = 0) \vee (x_1 \geq x_2 \wedge x_2 > 0) \vee (x_2 \geq x_1 \wedge x_1 > 0).$$
3. *Linear Simplify* : Since LA reduces ξ to $true$, the definition of gcd is complete.

– *Integer Divisibility* : Consider the predicate *divides* defined in the example discussed in the introduction. We observed that $divides(x_1, x_2)$ is defined only if its first argument is non-zero or both its arguments are 0. This can be established using the above algorithm as follows.

1. *Initialize*: Let $\xi = false$. and $\mathcal{C}_{divides}$ be the cover set for *divides*.
2. *Linear solve* : For the first cover set triple, the constraint equations are: $\{x_1 = u, x_2 = 0\}$. c_c is $\{x_2 = 0\}$. So, $\xi = (x_2 = 0)$.
   Using the second cover set triple $\langle\langle u, v\rangle, \{\}, \{v < u, v \neq 0\}\rangle$ the constraint equations are: $\{x_1 = u, x_2 = v\}$, with context $\{v < u, v \neq 0\}$. c_c is $x_2 < x_1 \wedge x_2 > 0$, so $\xi = (x_2 = 0) \vee (x_2 < x_1 \wedge x_2 > 0)$.

For the third cover set triple $\langle\langle u, u+v\rangle\{\langle u, v\rangle\}, \{u+v>v\}\rangle$, the constraint equations are $\{x_1 = u, x_2 = u+v\}$ with context $u+v>v$. c_c is $x_2 \geq x_1 \wedge x_1 > 0$. The updated ξ is:

$$\xi = (x_2 = 0) \vee (x_2 < x_1 \wedge x_2 > 0) \vee (x_2 \geq x_1 \wedge x_1 > 0).$$

3. *Linear Simplify* : Since ξ is not a valid formula, LA does not reduce it to $true$. The simplified form of ξ, $x_2 = 0 \vee x_1 > 0$, is output by the algorithm which corresponds to our initial observation that *divides* is defined only if when its first argument is non-zero or the second argument is zero.

4.2.1. *Incomplete Cover Sets*

In order to draw sound conclusions, it is important to use a complete cover set. Otherwise, unsound conclusions can be made. For example, if the cover set based on the definition of *divides* given in the introduction is used without caring about its completeness, it is possible to wrongly conclude that

$$divides(x, y+y) \quad if \quad divides(x, y),$$

even though the above formula does not hold for the case when $x = 0$. Since *divides* is defined only if $x > 0 \vee y = 0$, the above formula holds only under this condition. Thus,

$$divides(x, y+y) \quad if \quad divides(x, y) \wedge (x > 0 \vee y = 0)$$

is a theorem. Similarly, in the case of the second conjecture,

$$divides(2, x) = not(divides(2, s(x))) \quad if \quad (2 > 0 \vee x = 0)$$

is a theorem, which is the same as

$$divides(2, x) = not(divides(2, s(x))),$$

since the condition in the conditional formula is true.

Incomplete cover sets, however, can be useful in certain cases, such as proving $divides(2, x) = not(divides(2, s(x)))$, an incomplete cover set of *divides* can be used since the cover set is incomplete only if the first argument of *divides* is 0. We can relativize the completeness of a cover set with respect to a formula.

DEFINITION. *Relatively Complete LA-based Cover Sets:* An LA-based cover set $\mathcal{C}_f$ of $f(x_1, \cdots, x_n)$ is *complete with respect to* a linear arithmetic quantifier-free formula ξ with free variables $x_1, \cdots, x_n$ iff for any n-tuple $\langle e_1, \cdots, e_n\rangle$ of numbers either one of the following conditions hold.

– $\xi[x_1 \rightarrow e_1, \cdots, x_n \rightarrow e_n] = false$,

– there exists a cover set triple $c_i = \langle\langle s_1, \cdots, s_n\rangle, \{\cdots, \langle s'_1, \cdots, s'_n\rangle \cdots\}, cond\rangle$ and a substitution σ such that $\sigma(s_i) =_{LA} e_i$ and $\sigma(cond) = true$.

A complete LA-based cover set $\mathcal{C}_f$ is complete with respect to the formula *true*. A cover set $\mathcal{C}_f$ associated with $f(x_1, \cdots, x_n)$ can be used to establish an inductive property C *if* $\xi(x_1, \cdots, x_n)$ if $\mathcal{C}_f$ is complete with respect to $\xi(x_1, \cdots, x_n)$.

THEOREM 7. *Given an LA-based cover set* $\mathcal{C}_f$ *of* $t = f(x_1, \cdots, x_n)$ *that is complete with respect to* $\xi(x_1, \cdots, x_n)$*, an induction scheme generated from* $\mathcal{C}_f$ *is sound and complete for a conjecture* $C(\cdots, t, \cdots)$ *if* $\xi(x_1, \cdots, x_n)$*.*

Proof. By contradiction. Assume that $C(\cdots, t, \cdots)$ *if* $\xi(x_1, \cdots, x_n)$ is not true, but for each induction case of a scheme ψ obtained from $\mathcal{C}_f$, $C(\cdots, t, \cdots)$ *if* $\xi(x_1, \cdots, x_n)$ is true. Let $\langle e_1, \cdots, e_n\rangle$ be the smallest n-tuple of numbers with respect to the termination ordering $\succ_{LA}$ used to prove termination of the definition of f, that is a counterexample to $C(\cdots, t, \cdots)$ *if* $\xi(x_1, \cdots, x_n)$. That is $C(\cdots, f(x_1, \cdots, x_n), \cdots)[x_1 \to e_1, \cdots, x_n \to e_n]$ *if* $\xi[x_1 \to e_1, \cdots, x_n \to e_n] = false$. Equivalently, $\xi[x_1 \to e_1, \cdots, x_n \to e_n] = true$ and $C(\cdots, f(x_1, \cdots, x_n), \cdots)$ $[x_1 \to e_1, \cdots, x_n \to e_n] = false$.

Since $\mathcal{C}_f$ is complete with respect to $\xi(x_1, \cdots, x_n)$ and $\xi[x_1 \to e_1, \cdots, x_n \to e_n] = true$, there exists a cover set triple, $c_i = \langle\langle s_1, \cdots, s_n\rangle, \{\cdots, \langle s_1^k, \cdots, s_m^k\rangle, \cdots\}, cond\rangle$ and a substitution σ such that $\sigma(s_i) =_{LA} e_i$, $1 \le i \le n$, and $\sigma(cond) = true$. Corresponding to c_i, there exists an induction case ψ_j of the form, $\langle\langle\{x_1 \to s_1, \cdots, x_n \to s_n\}, cond, \{\}\rangle, \{\cdots, \langle\{x_1 \to s_1^k, \cdots, x_n \to s_n^k\}, cond, \{\}\rangle, \cdots\}\rangle$, by theorem 2.6. The induction subgoal corresponding to ψ_j, $(\theta(C)$ *if* $\theta(\xi))$ *if cond* assuming $\{\bigwedge_k(\theta_k(C)$ *if* $\theta_k(\xi))$ *if* $cond\}$ is *true*, where $\theta = \{x_1 \to s_1, \cdots, x_n \to s_n\}$ and $\theta_k = \{x_1 \to s_1^k, \cdots, x_n \to s_n^k\}$. Since $\sigma(\theta(C)$ *if* $\theta(\xi))$ is *false* and $\sigma(cond) = true$, one of $\sigma((\theta_k(C)$ *if* $\theta_k(\xi))$ *if* $cond)\}$ is false. $\sigma(cond)$ being *true* implies that there is a smaller counterexample to C *if* ξ since $\sigma(\langle s_1^k, \cdots, s_n^k\rangle)$ is lower in the termination ordering $\succ_{LA}$ than $\langle e_1, \cdots, e_n\rangle$. (by the definition of an LA-based cover set, $\langle s_1, \cdots, s_n\rangle \succ_{LA} \langle s_1^k, \cdots, s_n^k\rangle$ for any k, and since $\succ_{LA}$ is preserved under $=_{LA}$, $\langle\sigma(s_1), \cdots, \sigma(s_n)\rangle \succ_{LA} \langle\sigma(s_1^k), \cdots, \sigma(s_n^k)\rangle$). And, this is a contradiction. ∎

5. Generalization

While attempting proofs by induction, intermediate conjectures are generated which may be difficult to prove automatically. In some situations, however, it is possible to prove a generalization of a conjecture from which the conjecture follows. Most induction theorem provers support heuristics for generalizing conjectures. In *RRL*, at least two kinds of generalizations are performed: (i) abstracting a nonvariable subterm appearing in a conjecture to be a variable, if the nonvariable subterm appears more than once in the conjecture (in the left side as well as the right side and/or the condition), and (ii) dropping an assumption from a conditional conjecture. Semantic analysis can be useful in the implementation of the generalization heuristic since a subterm may have multiple occurrences semantically but it may not appear to have multiple occurrences syntactically. In this section, we illustrate the use of the linear arithmetic procedure for improving the generalization heuristic focusing on the first kind of generalization of abstracting a nonvariable subterm with multiple occurrences by a variable. We first briefly review how this generalization heuristic is performed in *RRL* based on syntactic properties without using *LA*, and then we discuss how this heuristic can be improved using *LA*.

Given a conjecture C of the form $l = r\ if\ cond$, we look for a maximal nonvariable subterm s occurring in at least two of l, r and *cond*. Since there may be many such maximal subterms, they are collected in a list as possible candidates for generalization. Abstracting any nonempty subset of a set of candidates would lead to a generalized version of the conjecture. For each such subset of candidates, a *generalization template* is generated in which the first component is a list of *abstraction pairs* of the form $\langle s, u\rangle$, where s is a candidate subterm and u is a new variable used to replace s, and the second component is a list of triples of the form $\langle p, t_1, t_2\rangle$ indicating that subterm t_1 at position p in C is replaced by t_2 using the abstraction pairs (obviously, t_2 is generated from t_1 by abstracting using the first component). A generalization template is used to generate a generalized version C_g of C by simultaneously replacing the subterm t_1 at position p in C by t_2 for every triple $\langle p, t_1, t_2\rangle$ in its second component. It is obvious that if C_g can be proved, then C can be proved.

The main steps of the generalization procedure in *RRL* are to (i) identify maximal nonvariable subterms that occur in a conjecture, (ii) generate different possible generalization templates by considering all possible nonempty lists of abstraction pairs, and finally (iii) use each generalization template, one by one, to generate a generalized conjecture, until a proof is obtained. Generation of candidate subterms for generalization and the order of replacement of candidate subterms by new variables is arbitrary and has been fine-tuned in *RRL* based on experimentation.

5.1. Using Linear Arithmetic for determining Subterms

A drawback of the above heuristic, as was the case with other heuristics discussed earlier, is that it is based on syntactic considerations only. A subterm may not have multiple occurrences syntactically in a conjecture, but semantically equivalent subterms may occur in the conjecture. For instance, consider the following conjecture about the *gcd* function:

$$gcd(x + y + 1, 2 * x + 2 * y + 2) = x + y + 1.$$

The first argument of *gcd* appears to be quite different from the second argument. If no semantic analysis is performed, *RRL* we would generate

$$gcd(u, 2 * x + 2 * y + 2) = u,$$

as a generalized version of the conjecture by abstracting occurrences of $x+y+1$ in the left and right side by a variable u since the second argument of gcd syntactically appears not to include $x+y+1$ as a subterm. This generalization is not a valid formula since there are counterexamples. Even if the above subterms are expressed in some normal form, say $s(x+y)$ and $s(s(x+x+y+y))$, respectively, the first does not occur in the second. However, from a semantic standpoint, the second argument of gcd is closely related to its first argument. If LA is used to look for equivalent subterms, then this relationship can be identified, and it can be found that the first argument of gcd appears twice in its second argument.

Given a conjecture of the form $l = r\ if\ cond$, for some maximal nonvariable subterm s of l, we check whether it appears in r and/or $cond$ also. If s is not a linear term, syntactic subterm check is performed as before. Otherwise, we check whether other linear subterms include occurrences of s. This check can be performed using LA by querying whether a linear subterm $t \geq s$. If the answer is no, then s does not occur in t; otherwise, we find the number of times s appears in t (this can be done by repeated query and subtraction from t until the result becomes smaller than s). Let $t = k * s + tr$, where k is a positive integer and tr is a linear term smaller than s. In this case, s can be abstracted to be a new variable u giving the abstraction pair $\langle s, u\rangle$, and the subterm t in the conjecture at position p can be replaced by $k * u + tr$.

Once candidate subterms in a conjecture C for generalization have been identified, generalization templates can be generated by considering all possible subsets of candidates for generalization as before.

In the above conjecture about gcd, we look for occurrences of the linear term $x+y+1$ in other linear terms in the conjecture. LA is queried to compute the number of occurrences of $x+y+1$ in the other linear term $2*x+2*y+2$ in the conjecture. By repeated subtraction the number of occurrences is computed to be 2 with a remainder 0. Similarly, the right side of the conjecture also has an occurrence of $x+y+1$. And, that is the only maximal subterm in the conjecture that can be generalized, since subterm $2*x+2*y+2$ does not occur in the other linear subterm $x+y+1$. By abstracting $x+y+1 = u$, the template formed is

$$\langle\{(x+y+1, u)\}, \{\langle 1.1, x+y+1, u\rangle, \langle 1.2, 2*x+2*y+2, 2*u\rangle,$$
$$\langle 2, x+y+1, u\rangle\}\rangle.$$

From this, the conjecture is generalized to: $gcd(u, 2*u) = u$. This conjecture can be rewritten using the fourth rule of gcd to give $gcd(u,u) = u$, which can be proved using LA based induction scheme generation procedure, as shown earlier. A complete description of the extended generalization procedure using LA follows.

- **Input**: A conjecture C of the form $l = r\ if\ cond$.
- **Output**: A set of generalized versions of conjecture C.
- **Method** : For each maximal nonvariable subterm s of C at position p do:

1. *Generate Templates*:
   a) Let (s, x) , $x \notin Vars(C)$ be the abstraction pair associated with s.
   b) *Compute the replacement triples*: Let the set of triples $S = \{\}$. For each nonvariable subterm t of C at a position q other than p do:
      i) If s is not a linear term and $t = s$ then $S := S \cup \{\langle q, s, x\rangle\}$.
      ii) If s is a linear term, then compute the number of occurrences, n of s in t, by using LA by repeated subtraction of s from t until $t \geq s$ with rt being the remainder, if any.
      iii) If $n = 0$ then no replacement triple is generated. Otherwise, $S := S \cup \{\langle q, t, nx + rt\rangle\}$.
2. Templates whose second components do not replace terms occurring in at least two of l, r or *cond* are discarded.
3. For each nonempty subset of templates, output the generalized version of C obtained by simultaneously replacing for each of the triples $\langle p, t_1, t_2\rangle$, the subterm t_1 at position p by t_2 in C.

6. Related Work

The seminal work on mechanizing induction is that of Boyer and Moore [1, 2]. In [1], Boyer and Moore discussed recursion analysis of definitions and the formulation of induction schemes for a given conjecture. Many other heuristics which manipulate induction schemes towards the choice of the most appropriate induction scheme for a given conjecture are also developed there. These methods have been implemented in their theorem prover *Nqthm*. All of these methods are purely syntactic and do not exploit the semantic information associated with the functions in any manner. For instance, in formulating an induction scheme, for a given conjecture based on a term $f(t_1, \cdots, t_n)$ they write [1][pp :185 - 186]

"If the t_i occupying the argument positions measured by the induction template we have in mind are all distinct variables, then we can use the template to construct a sound induction scheme. And in general, if any of the measured t_i is not a variable or is the same variable as another measured t_i we cannot. .....Let *changeables* be those t_i that occurred in measured positions that are also sometimes changed in the recursion. Let the *unchangeables* be the variables occurring in those t_i occupying measured positions that never change in recursion. A template *applies* to a term if the changeables are all distinct variables and none of them is among the unchangeables."

In almost all the examples discussed in this paper, the methods given in [1] would be inapplicable. For instance, in the definition of *gcd* both the arguments of *gcd* are required to justify the termination of the definition and hence are in measured positions. Further, both of them are changed in the recursive calls to *gcd* in the definition and are therefore, *changeables* in the sense of [1]. Thus given a conjecture of the form $gcd(m + m, 2) = 2$, an induction scheme cannot be formulated corresponding to the term $gcd(m + m, 2)$ since the arguments to *gcd* are nonvariables. As yet another example, in the conjecture such as $gcd(m, m) = m$, an induction scheme cannot be formulated corresponding to the term $gcd(m, m)$ also, since the arguments in changeable positions of gcd are not distinct variables. Similarly, in our motivating example, $divides(2, x) = not(divides(2, s(x)))$, $s(x)$ is a nonvariable in a changeable position of $divides$ and is not syntactically compatible with $+$ used in defining $divides$.

$Nqthm$ provides a mechanism for performing user-directed induction using the so-called *hint* directive [2]. A user can specify a function definition to be used for generating an induction scheme. The variables in the conjecture and additional dummy variables, if required are provided as arguments to the function being specified. For the conjecture (P_2), $divides(2, x) = not(divides(2, s(x)))$, the scheme generated using LA can be obtained in $Nqthm$ using the hint directive:

$$divides(2, x) = not(divides(2, s(x)))\ if\ induct(even(m)).$$ [13]

This shows the power of the LA based induction scheme generation procedure. Some induction schemes which must otherwise be specified using hint, can be automatically generated using the linear arithmetic procedure.

The description of merging schemes in this paper follows along the lines of [1]. In [1] whenever two schemes are merged, the basis case of the merged scheme is obtained by negating the conditions of the merged induction cases. This is not the case in RRL where the basis cases are generated from the rules which do not have recursive calls on the right hand side. Consequently, for soundness all induction cases have to be considered while merging.

7. Concluding Remarks

We have shown how a decision procedure such as Presburger arithmetic which embodies knowledge about numbers and their two related representations using $0, s, +$, can be effectively used to further automate and make more effective, certain heuristics in mechanizing induction. The focus of this paper has been on generating induction schemes using the decision procedure. We have briefly reviewed how the decision procedure can be used for checking completeness of definitions and hence the associated cover sets and induction schemes. This extension of the cover set method which does semantic analysis with the help of the linear arithmetic decision procedure, enables the automation of proofs of many theorems which otherwise, require human guidance and hints. We have manually exercised this extension quite a bit on a number of examples and the results are extremely encouraging. Since RRL already supports a decision procedure for linear arithmetic [9], we plan to extend the procedure to handle the various applications discussed in this paper. We believe that the use of the linear arithmetic procedure for performing semantic analysis will also be useful in the proof by consistency (also popularly known as the inductionless induction) approach.

[13] In Nqthm corpus induction scheme based on the definition of the function *odd* is used instead.

The proposed approach suggests a new direction for enhancing heuristics for mechanizing induction based on semantic information about data structures. We believe that in data structures such as finite sets and multisets, constructors such as the null element, inserting a single element and the union operation play a role similar to the constructors 0, s and + for numbers. So it should be possible to design decision procedures and heuristics for converting between different representations of values of the data structure. This might be the case for other data structures also such as lists and sequences. There is a need to study and investigate developing decision procedures for these data structures, especially focusing on aspects related to convertibility among representations.

References

1. R.S. Boyer and J S. Moore, *A Computational Logic.* ACM Monographs in Computer Science, 1979.
2. R.S. Boyer and J S. Moore, *A Computational Logic Handbook.* New York: Academic Press, 1988.
3. R.S. Boyer and J S. Moore, "Integrating decision procedures into heuristic theorem provers: A case study of linear arithmetic," *Machine Intelligence* 11 (1988) 83–157.
4. N. Dershowitz, "Termination of rewriting," *J. of Symbolic Computation* 3, 69-116, 1987.
5. J.-P. Jouannaud and E. Kounalis, "Automatic proofs by induction in theories without constructors," *Information and Computation* 82, 1-33, 1989.
6. D. Kapur, "An automated tool for analyzing completeness of equational specifications," Proc. of *International Symposium on Software Testing and Analysis (ISSTA),* Seattle, August 1994, 28-43.
7. D. Kapur, D.R. Musser, and X. Nie, "An Overview of the Tecton Proof System," *Theoretical Computer Science* Journal, special issue on *Formal Methods in Databases and Software Engineering,* (ed. V. Alagar), Vol. 133, October, 1994, 307-339.
8. D. Kapur, P. Narendran, D. Rosenkrantz, H. Zhang., "Sufficient-completeness, quasi-reducibility and their complexity," *Acta Informatica*, 28, 1991, 311-350.
9. D. Kapur and X. Nie, "Reasoning about numbers in Tecton", Proceedings of the *8th International Symposium on Methodologies for Intelligent Systems, (ISMIS'94)*, Charlotte, North Carolina, October 1994, 57-70.
10. D. Kapur and H. Zhang, "An Overview of Rewrite Rule Laboratory (RRL)," *J. of Computer and Mathematics with Applications,* 29, 2, 1995, 91-114.
11. C. Walther, "Combining Induction Axioms By Machine", Proc. of *Twelfth International Joint Conference on Artificial Intelligence*, Chambery, France, 1993.
12. H. Zhang, *Reduction, superposition and induction: automated reasoning in an equational logic.* Ph.D. Thesis, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, 1988.
13. H. Zhang, D. Kapur, and M.S. Krishnamoorthy, "A mechanizable induction principle for equational specifications," *Proc. of Ninth International Conference on Automated Deduction (CADE-9)*, Argonne, IL. Springer-Verlag LNCS 310, 250-265, 1988.