Applications of Decision Algorithms for Presburger Arithmetic in Rewrite Automated Deduction

Luiz M. R. Gadelha Jr*and Mauricio Ayala Rincón
Departamento de Matemática, Universidade de Brasília,
70910-900. Brasília, DF, Brasil
E-mail: {gadelha,ayala}@mat.unb.br

Abstract

We present some slight improvements to a semi-decision algorithm for Presburger arithmetic originally developed by Shostak that increase the class of formulas effectively decidable. Furthermore, we show how decision algorithms for Presburger arithmetic may be combined with conditional rewrite techniques for automated deduction in algebraic specifications presented by conditional equational clauses with an arithmetic parameter. In particular we show how Knuth-Bendix completion could be improved by eliminating inconsistent and trivial (modulo the arithmetic parameter) critical pairs. For a further motivation we show some examples of Kapur and Subramaniam [KS96b] on how the cover-set method could be improved by searching solutions for the arithmetic constraints resulting when producing induction schemes.

Keywords: *Automated Deduction, Conditional Rewriting, Algebraic Specification.*

1 Introduction

Rewriting techniques were extended to conditional equationally defined classes of algebras in order to obtain more elegance and expressiveness than with pure equational rewriting. Also, conditional (and unconditional) rewrite specifications were extended including parameters which can be solved by built-in (non-rewriting) algorithms more efficiently than with decision methods based on pure rewriting. Much work has been done in the field of rewriting theory in order to obtain an appropriate combination of rewriting and built-in algorithms. One of the most appropriate built-in theories to be included as parameter of modular rewrite specifications is Presburger arithmetic ($\mathcal{PA}$), since it appears as the basis of almost all deductive systems and cannot be manipulated directly by pure rewriting techniques as shown by Vorobyov [Vor88]. Of course $\mathcal{PA}$ is also appropriate because of the existence of well-known decision algorithms for the theory enlarged with non-interpreted function and predicate symbols [Sho79].

There are a great variety of studies suggesting how to combine built-in algorithms and rewriting techniques appropriately. There are approaches considering the general theory of the built-in parameter as parameter of rewrite specifications (see [Vor89], [Aya93, Aya97]) which allow an operational treatment by rewriting, case-analysis and validity check modulo the built-in parameter theory. Other approaches work with a specific model of the built-in theory giving restrictions and assumptions on the built-in parameter which allow an operational treatment without case-analysis modulo the built-in parameter model (see [DO90], [AB92],[Bec94]).

*Supported by a graduate grant of the Brazilian National Research Council (CNPq).

The first person to formalize a rewriting algorithm for conditional term rewriting systems (CTRSs) with built-in predicates was Vorobyov [Vor89]. He treated CTRSs whose conditions are pure built-in predicates, without admitting standard equational conditions as in [Aya93, Aya97]. In [Aya93] rewriting techniques were combined with built-in algorithms in such a way that it is possible to deduce Horn clauses whose conditions are purely built-in. Pure rewriting is insufficient and it should be refined with satisfiability check of the built-in conditions (matching modulo the built-in theory) and case-analysis guided by the built-in conditions. The language of the built-in theory is considered as a signature over basic sorts and the remaining function symbols of the conditional specification associated with the CTRS range over extended sorts. In this way one guarantees that the whole specification is a conservative extension of the specification of the built-in theory. This restriction appears to be very strong, but it allows reasonable manipulation of many important examples because it occurs often when implementing formal specifications, where new sorts are constructed from the concrete ones.

Dershowitz and Okada [DO90] studied CTRSs with built-in predicates and standard equational conditions as premises. They briefly treated termination, confluence and a critical pair criterion assuming that any basic term can be replaced by an equivalent normal form before rewriting is applied. In a certain sense, this means that the built-in mechanism can also be conceived as a pure rewriting procedure. Avenhaus and Becker [AB92] presented a method for integrating built-in operations that are described by a given built-in algebra into conditional rewriting. They do not allow new sorts in the extended signature and the variables are restricted to range over basic terms. They separate the semantics of the built-in algebra from the syntax of the whole specification by introducing sort hierarchies. Later on Becker [Bec94] presented a rewriting operationalization of clausal specifications where the axioms are positive/negative conditional equations admitting predefined algebras. His approach can be considered as an attempt to provide a general framework of handling specifications with predefined algebras in a semantically clean way. Avenhaus and Becker define the rewrite relation modulo the built-in algebra in such a way that a rule can be applied only if there is an exact matching. Their notion of rewriting can essentially be seen as rewriting of equivalence classes.

In this work we do not treat the question of which is the best form of combining built-in decision algorithms and rewriting deductive methods. As a matter of fact this is necessary and can be done in many ways according to the specific objectives; i.e., to reason in the general theory or in a specific model of the specification. We focus on the analysis of specific situations in which it is necessary to decide about arithmetic formulas of the built-in parameter $\mathcal{PA}$ when rewriting deduction mechanisms are performed. In particular, we show how Knuth-Bendix completion of conditional equational modular specifications with an arithmetic parameter can be improved by eliminating critical pairs whose conditions are $\mathcal{PA}$-inconsistent or which are theorems of $\mathcal{PA}$ enlarged with non-interpreted function and predicate symbols of the specification. We also study how the cover-set method used to generate induction schemes, based on the structure of the definition of function symbols in the conditional equational specifications, can be improved by searching for solutions for the arithmetic constraints resulting when producing induction schemes, as in [KS96b]. Of course, as a semi-decision algorithm for $\mathcal{PA}$ we suggest (and show the suitability of) our improvement of the Shostak method which allows an effective decision of a greater family of formulas than the original method and allows a treatment of arithmetic formulas enlarged with non-interpreted function and predicate symbols, which is the standard case of formulas occurring in modular specifications with an arithmetic parameter [GA96].

This paper is organized as follows: in the second section we present briefly our suggested improvement to the Shostak semi-decision method; in the third section we illustrate how Knuth-Bendix completion of conditional equational specifications with an arithmetic parameter can be improved by eliminating $\mathcal{PA}$-inconsistent and valid critical pairs; in the fourth section we present examples showing the improvement of rewrite inductive proofs by solving arithmetic constraints which appear

as part of induction schemes; finally, we present conclusions and comment on future work.

Notations consistent with the standard ones in the field of rewriting theory are used. Complete surveys on rewriting can be found in [AM90], [DJ90], [Klo92] and [Pla93].

2 Improvement of Shostak semi-decision procedure for $\mathcal{AP}$

The Presburger arithmetic ($\mathcal{PA}$) consists of the structure of integer numbers with the addition operation and an ordering predicate, $\langle \mathbf{Z}, +, \leq \rangle$. The theory of the $\mathcal{PA}$ is very relevant in the history of automated theorem proving since it was the first one to be shown decidable using the method of quantifier elimination [Pre29]. The original Presburger method was further improved by many authors among them Cooper [Coo72] who included algebraic treatment of the formulas including new predicate symbols such as $|$ ($a|x$ abbreviates $\exists y\ ay = x$, where a is an integer constant). Despite the existence of full decision algorithms, these methods present an extremely high complexity $O(2^{2^{2^n}})$ [Opp78]. It turns out to be interesting to study tractable semi-decision algorithms. Shostak [Sho79], presented an algorithm for unquantified formulas of the $\mathcal{PA}$, that uses integer linear programming (ILP), that is fairly more efficient than decision algorithms. It has been implemented in LISP[1] and improved in order to make a greater class of formulas decidable. The algorithm transforms the negation of a formula F in its disjunctive normal form $G_1 \vee G_2 \vee \ldots \vee G_n$, that corresponds to a set of ILP problems. F is valid if and only if none of the conjunctions of inequations, G_i, has integer solutions. The semi-decidability of the algorithm is due to the method used for solving the ILP problems: Bledsoe's SUP-INF method. This method determines, for a given system of inequations, G_i, the maximum and minimum values that its variables may assume, one at a time. If for some variable the corresponding interval contains no integers, then it is concluded that the system has no integer solutions. Shostak proposes sequential application of Bledsoe's method to determine the respective intervals for each variable in the system. When an interval of possible solutions for a variable is found, a real value (integer if possible) is selected randomly. The system is modified replacing this variable with the selected value. The new system is used to set new intervals and values for all the variables in the original system recursively. If the first interval has no integers or if some of the intervals of solutions subsequently calculated is empty then the validity of the original formula is proved. If we get, after the recursive computation, a set of integer values for the variables of the problem, the invalidity of the original formula can be concluded with the set found as a counterexample. Shostak affirms that incompleteness of his ILP method rarely arise in practice, but we verified that some very simple formulas of frequent use in decision processes are not decidable by his algorithm. Take for example the valid $\mathcal{PA}$ formula $(x < 1) \vee (x > 5) \vee (x > 15y) \vee (15y > 2x)$. The algorithm treats the ILP problem related to the negation of the formula, $(1 \leq x \wedge x \leq 5 \wedge x \leq 15y \wedge 15y \leq 2x)$, in the following way: the interval of solutions found for the variable x is $[1, 5]$; a random value is selected for x, for example $x = 1$, the modified formula is $1 \leq 15y \wedge 15y \leq 2$ which determines the interval $[\frac{1}{15}, \frac{2}{15}]$ for y. Despite having no integer solution for y we may not deny the existence of integer solutions due to the random selection made for x, consequently neither validity nor invalidity of the original formula can be concluded by the algorithm. A small improvement that can be done is to include sequential steps before the recursive computation of intervals determining independently intervals for each variable applying the SUP-INF method. The decision process for the previous formula follows. The interval $[1, 5]$ is computed for x and then, without random selection for x, the interval $[\frac{1}{15}, \frac{10}{15}]$ is computed for y[2]. Since there are no integer values possible for y (without the constraints on x) the validity of the formula is concluded.

[1] Available in *http://www.mat.unb.br/~ gadelha*.

[2] Among the admissible values for x, the minimum and maximum values that y can assume are $\frac{1}{15}$ and $\frac{10}{15}$.

Also, simple valid formulas should be algebraically treated, in order to reach their proofs. In particular, validity of formulas of the form: $c_1x_1 + \ldots + c_nx_n \neq c$, where c and the c_i's are integer constants, the x_i's variables and c and $a = gcd\{c_1, \ldots, c_n\}$ are relative primes, cannot be directly detected by Shostak's algorithm. Consider, for example, the valid $\mathcal{PA}$ formula $3x + 12y \neq 7$. The ILP problem corresponding to its negation is $3x + 12y = 7$. Applying non-sequentially Bledsoe's method, it is possible to determine the interval $(-\infty, +\infty)$ for both variables x and y. As these intervals contain integers, validity of the original formula cannot be proved and it can be presented a real counterexample. The solution suggested for these kind of formulas is to make first the algebraic transformation $c_1x_1 + \ldots + c_nx_n = az \rightarrow az \neq c$, where $a = gcd\{c_1, \ldots, c_n\}$. Continuing with the example above, one should consider the formula $x + 4y = z \rightarrow 3z \neq 7$, whose negation corresponds to the ILP problem $x + 4y = z \wedge 3z = 7$. Observe that the interval for z is $[\frac{7}{3}, \frac{7}{3}]$, which does not include integers. In this way the validity of the original formula can be proved.

The sequential steps and the algebraic transformations suggested increase the class of formulas effectively decidable by the algorithm originally proposed by Shostak. The improved algorithm is still a semi-decision algorithm, but it will be very useful in practice, since decision algorithms for $\mathcal{PA}$ have super-exponential complexity [Opp78]. An immediate application of decision algorithms for $\mathcal{PA}$ is their combination with methods for (inductive) theorem proving based on rewriting techniques in theories that are specified by rewrite rules including arithmetic parameters [Aya95]. Shostak's semi-decision algorithm appears to be appropriate for its combination with rewriting based deduction methods since it allows a treatment of unquantified $\mathcal{PA}$ formulas enlarged with predicate and function symbols making immediate implementations of matching and unification algorithms modulo the $\mathcal{PA}$ in the sense mentioned in [DJ90]. This is essential for deduction in rewrite modular specifications with arithmetic parameters. Our improvements of Shostak algorithm can be consulted in [GA96].

3 Completion of CTRSs with arithmetic parameters

The unquantified $\mathcal{PA}$ is an excellent example of a parameter theory for conditional equational modular specifications implemented by rewriting techniques, because arithmetic lies in the very basis of almost all formal systems of reasoning and because the unquantified $\mathcal{PA}$ cannot be easily covered by rewriting systems as shown by Vorobyov [Vor88]. He proved that the unquantified theory of the $\mathcal{PA}$ cannot be axiomatized by a canonical or convergent *context-free* (conditional or unconditional) rewriting system, finite or infinite. By *context-free* it is meant separation between the logical and non-logical part of the system; That means that every *context-free* reduction relation includes a canonical sub-relation for the boolean algebra (for example the one proposed by Hsiang [Hsi85]) and boolean connectives are prohibited in the left-hand sides of non-logical rules (in our case the rules for defining $\mathcal{PA}$.

Here we point out how $\mathcal{PA}$ decision algorithms could be applied during the completion process of conditional equational specifications. We suppose familiarity with notions of conditional rewriting such as conditional critical pairs and the conditional completion process (see for example [Siv89] and [Gan91]).

Example 3.1 Consider the following CTRS specifying the predicate *divides* on natural numbers presented also by rewrite rules. The equational specification that deal with the arithmetic includes the constant zero, 0, the successor constructor, S, addition, $+$, (which is specified as an associative commutative function symbol) and ordering predicate, $<$. Obviously the intended semantics of the

predicate $divides(x, y)$ is "x divides y".

1.	$x + 0 \rightarrow x$	5.	$S(x) < S(y) \rightarrow x < y$
2.	$x + S(y) \rightarrow S(x + y)$	6.	$divides(u, 0) \rightarrow true$
3.	$0 < S(x) \rightarrow true$	7.	$divides(u, v) \rightarrow false \ if \ ((v < u) \ and \ (v \neq 0))$
4.	$x < 0 \rightarrow false$	8.	$divides(u, u + v) \rightarrow divides(u, v)$

Consider the three critical pairs generated during conditional completion of the CTRS that result from overlapping rules 8 with 7, 8 with 1 and 2 with 8 respectively (this can be done with RRL):

$$divides(u, v) = false \ \ if \ (u + v < u) \ and \ (u + v \neq 0)$$
$$divides(u, u) = true$$
$$divides(u, S(u + y)) = divides(u, S(y))$$

Knuth-Bendix conditional completion concludes generating a CTRS "equivalent" to the original one, which includes the three previous critical pairs oriented from left to right. However, if one works with a built-in parameter for $\mathcal{PA}$ by restricted to non-negative integers, which can be made operative with our decision algorithm for $\mathcal{PA}$ by restricting all variables to be non-negative, the first critical pair could be eliminated because of $\mathcal{PA}$-inconsistency of its premise, and the second one too, because it semantically subsumes (w.r.t. $\mathcal{PA}$) rules six and eight; in fact, observe that $divides(u, u)$ is equal to $divides(u, u+0)$ modulo $\mathcal{PA}$. Remember that our decision algorithm for the $\mathcal{PA}$ works for formulas with non-interpreted function and predicate symbols. It allows to verify that $divides(u, u)$ and $divides(u, u + 0)$ are equivalent, without interpreting the predicate $divides$. $\diamond$

The modification of the conditional completion method is described by means of inference rules as usually done. New deletion and simplification inference rules should be added to the set of inference rules for the conditional completion process.

$\frac{\langle E \cup \{s=t \ if \ SC \wedge P\}, R\rangle}{\langle E, R\rangle}$,	if $\mathcal{PA} \models P \Rightarrow s = t$;	*Deletion by $\mathcal{PA}$-validity*
$\frac{\langle E \cup \{s=t \ if \ SC \wedge P\}, R\rangle}{\langle E, R\rangle}$,	if P is $\mathcal{PA}$-inconsistent;	*Deletion by $\mathcal{PA}$-inconsistence*
$\frac{\langle E \cup \{s=t \ if \ SC \wedge P\}, R\rangle}{\langle E \cup \{u=t \ if \ SC \wedge P\}, R\rangle}$,	if $s \equiv_{\mathcal{PA}} s'$ and $s' \rightarrow_R u$	*Simplification modulo $\mathcal{PA}$*

Table 1: Deletion and simplification inference rules of the conditional completion process

Here, $s = t \ if \ SC \wedge P$ stands for a conditional equation, where SC is its standard condition (that is, a conjuntion of non-arithmetic equations) and P its arithmetic condition. At any stage of completion there is a set E_i of conditional equations and a set R_i, of conditional rewrite rules. Initially completion starts with an input set of conditional equations and no rules. A simple completion step may be viewed as applying an inference rule to transform the current pair $\langle E_i, R_i\rangle$ into an "equivalent" pair $\langle E_{i+1}, R_{i+1}\rangle$. This is denoted by $\langle E_i, R_i\rangle \vdash \langle E_{i+1}, R_{i+1}\rangle$.

The deletion by $\mathcal{PA}$-validity inference rule is applied by checking, the formula $P \Rightarrow s = t$ with the decision procedure for $\mathcal{PA}$. The deletion by $\mathcal{PA}$-inconsistence inference rule applies when inconsistence of the arithmetic condition is checked with the decision procedure. Application of the

simplification modulo $\mathcal{PA}$ inference rule is guided by the left-hand sides of the current set of rules, R, in each step of the completion process, that is it applies whenever there is a subterm at some position π of s such that for a rule in $l \to r$ *if* C in R there is a $\mathcal{PA}$-matching substitution σ from l into $s|_\pi$ (i.e., $\mathcal{PA} \models l\sigma = s|_\pi$) and $l \to r$ *if* C applies under substitution σ and $s[r\sigma]_\pi u$. For a finite set of rules R and a term s it is decidable if there is a left-hand side l of a rule in R and a subterm of s for which there exists a $\mathcal{PA}$-matching substitution. This is showed in general for theories with decidable set of unquantified existential-universal formulas [Aya97]. i.e. sentences of the form $\exists x_1 ... \exists x_n \forall y_1 ... \forall y_m \phi(x_1, ..., x_n, y_1, ..., y_m)$ is a quantifier-free formula.

Example 3.2 Continuation of example 3.1. The first critical pair $divides(u, v) = false$ *if* $(u + v < u)$ *and* $(u + v \neq 0)$ is deleted by application of the second inference rule, because $u + v \neq 0 \wedge u + v < u \wedge u \geq 0 \wedge v \geq 0$ is $\mathcal{PA}$-inconsistent. $u \geq 0 \wedge v \geq 0$ is added to the original condition since the intended parameter is restricted to the naturals. The critical pair $divides(u, u) = true$ is simplified by the third inference rule to $divides(u, 0) = true$ by application of the eight rule: observe that the left-hand side of the eight rule, $divides(u, u + v)$, $\mathcal{PA}$-matches with $divides(u, u)$ using substitution $\sigma = \{v \mapsto 0\}$. Subsequently, $divides(u, 0) = true$ simplifies to $true = true$ which can be deleted with the non-presented inference rules of the conditional completion process. $\diamond$

Of course, inclusion of these three additional inference rules in the usual conditional completion process preserves soundness, since each of the additional rules is sound. Inference rules for simplifying standard premises of the conditional equations and rules modulo $\mathcal{PA}$ and for eliminating valid $\mathcal{PA}$ conditions, as is done in [Aya93], could also be included in the usual conditional completion process. Searching for $\mathcal{PA}$-matchings and verifying $\mathcal{PA}$-consistence and $\mathcal{PA}$-validity with the decision algorithm for unquantified $\mathcal{PA}$ enlarged with non-interpreted function and predicate symbols make the completion process extremely inefficient, because of the high complexity of the $\mathcal{PA}$ decision algorithm. However deduction of trivial and inconsistent (modulo $\mathcal{PA}$) critical pairs could give rise to generation of large set of non-applicable rules during completion producing subsequently more non-sense (modulo $\mathcal{PA}$) rules making also inefficient the completion process.

The techniques described here are based on the "Kernel" of the completion procedures detailed in [Gan91] for standard CTRSs and the principal goal here is to illustrate the advantages and differences in the case of our treatment of $\mathcal{PA}$ parameterswith respect to explicit presentation of arithemtical parameters by rewrite rules. It must be clear that the inference rules previously described give rise to a completion algorithm, that is based essentially on the old algorithms for standard CTRSs, and which present slight changes with respect to them.

Of course, since we propose the application of semi-decision algorithms for the arithmetic parameter, only in the case that $\mathcal{PA}$-validity and $\mathcal{PA}$-consistence can be checked critical-pairs will be effectively eliminated, speeding up the completion process. Anyway our improvements on the completion process are independent of the (semi-)decision method used for the arithmetic parameter.

4 Rewrite inductive proofs with $\mathcal{AP}$ decision algorithms

Developing automated procedures for generating induction schemes is interesting since many theorems may not be proved by usual induction over the natural numbers. The main idea behind these methods is to use the structure of function definitions in the rewrite specifications to generate appropriate induction schemes. We select the cover-set method [ZKK88] for generating induction schemes for functions defined by conditional rewriting systems and illustrate which rôle arithmetic decision algorithms play, following suggestions originally presented by Kapur and Subramaniam in [KS96b].

Given a noetherian CTRS defining a function f and a conjecture about f, the cover-set method generates an induction scheme for proving the given conjecture based on the structure of the definition of f. The cover-set method is implemented in the *RRL* theorem proving system [KZ89].

Example 4.1 Consider the following rewriting system defining the greatest common divisor.

$$\begin{array}{llll} 1. & gcd(x,0) \to x & 3. & gcd(x, x+y) \to gcd(x,y) \\ 2. & gcd(0,x) \to x & 4. & gcd(x+y, x) \to gcd(y,x) \end{array}$$

The usual induction scheme for proving the commutativity of the *gcd* consists of an induction basis $gcd(0,y) = gcd(y,0)$ and an induction step $gcd(x,y) = gcd(y,x) \Rightarrow gcd(x+1,y) = gcd(y,x+1)$. The basis of the scheme can be proved by reductions with the first and second rules. But observe that the rewriting system for the *gcd* includes no rule for simplifying the conclusion of the induction step in order to apply the induction hypothesis. Instead, the cover-set method uses the structure of the definition of the *gcd* to generate the scheme: $gcd(0,y) = gcd(y,0)$ and $gcd(x,y) = gcd(y,x) \Rightarrow gcd(x,x+y) = gcd(x+y,x)$, which allows to prove the conjecture, since $gcd(x,x+y) = gcd(x+y,x)$ reduces to the induction hypothesis by applying the third and fourth rules. $\diamond$

Definition 4.1 *[KS96b] Let R be a noetherian CTRS with rules of the form $l \to r$ if C satisfying the usual restriction on variables: $Vars(r) \cup Vars(C) \subset Vars(l)$. Furthermore suppose that R specifies a set of function symbols F and that $f \in F$ is an n-ary function symbol. The* **function symbol definition** *D_f for $f \in F$ is the set of conditional rules in R whose left-hand side have main symbol f. The* **cover-set** *C_f for the function symbol f with function definition D_f, is the set of 3-tuples derived from rewrite rules $l \to r$ if $C \in D_f$ with the following shape: $\langle\langle$arguments of f in $l\rangle, \{$list of arguments of each occurrence of f in $r\}, C\rangle$.*

Observe that the second component of a cover-set triple is of the form $\{\langle s_1^1, \ldots, s_n^1\rangle, \ldots, \langle s_1^k, \ldots, s_n^k\rangle\}$, where each tuple corresponds to arguments of occurrences of f at r. An **induction scheme** for a conjecture $\mathcal{C}$ about $f(t_1, \ldots, t_n)$ occurring at position π in $\mathcal{C}$ consists of a finite set of induction cases of the form $\langle\langle \sigma_\mathcal{C}, cond_\mathcal{C}, repl_\mathcal{C}\rangle, \{\ldots, \langle \theta_i, cond_i, repl_i\rangle, \ldots\}\rangle$ in which the first component is the *induction conclusion* and the second one is the *induction hypothesis*. Each induction scheme is obtained from a cover-set triple in C_f of the form $\langle\langle s_1, \ldots, s_n\rangle, \{\ldots, \langle t_1, \ldots, t_n\rangle, \ldots\}, C\rangle$ by applying the following procedure.

Procedure 4.1 [KS96b] Given a conjecture $\mathcal{C}$ about an n-ary function symbol f occurring at position π in $\mathcal{C}$ as the term $t = f(t_1, ..., t_n)$ and let D_f be the function definition of f w.r.t. a CTRS R. This algorithm computes an induction scheme.

Begin
  Compute the cover-set C_f from D_f.
  For each triple $c = \langle\langle s_1, ..., s_n\rangle, \{\langle s_1^1, ..., s_n^1\rangle, ..., \langle s_1^k, ..., s_n^k\rangle\}, C\rangle$ in C_f **do**
    Let σ be the mgu of t and $s = f(s_1, ..., s_n)$:
      the induction conclusion is $\langle \sigma_\mathcal{C}, cond_\mathcal{C}, repl_\mathcal{C}\rangle$ where $\sigma_\mathcal{C}$ is σ restricted to variables occurring in t, $cond_\mathcal{C} = C\sigma$ and $repl_\mathcal{C} = \{\pi \leftarrow s\sigma\}$;
    Let σ_i be the most general unifier for t and $s' = f(s_1^i, ..., s_n^i)$ for some $1 \le i \le k$:
      the i-th induction hypothesis is: $\langle \theta_i, cond_i, repl_i\rangle$ where θ_i is σ_i restricted to variables occurring in t, $cond_i = C\sigma_i$ and $repl_i = \{\pi \leftarrow s'\sigma_i\}$;
End

Example 4.2 We now show in detail the previous proof of $\mathcal{C} \equiv gcd(m,n) = gcd(n,m)$. $C_{gcd} = \{\langle\langle x, 0\rangle\rangle, \{\}, \{\}\rangle, \langle\langle 0, x\rangle, \{\}, \{\}\rangle, \langle\langle x, x+y\rangle, \{\langle x, y\rangle\}, \{\}\rangle\}, \langle\langle x+y, x\rangle, \{\langle y, x\rangle\}, \{\}\rangle\}$. The induction

scheme for the term $gcd(m,n)$ occurring at position 1 in $\mathcal{C}$ is generated by computing the most general unifiers between $gcd(m,n)$ and the corresponding triples of the cover-set C_{gcd} obtaining:

$\{\ \langle\langle\{m \mapsto x, n \mapsto 0\},\{\},\{1 \leftarrow gcd(x,0)\}\rangle,\{\}\rangle,$
$\langle\langle\{m \mapsto 0, n \mapsto x\},\{\},\{1 \leftarrow gcd(0,x)\}\rangle,\{\}\rangle,$
$\langle\langle\{m \mapsto x, n \mapsto x+y\},\{\},\{1 \leftarrow gcd(x,x+y)\}\rangle,\{\langle\langle\{m \mapsto x, n \mapsto y\},\{\},\{1 \leftarrow gcd(x,y)\}\rangle\}\rangle,$
$\langle\langle\{m \mapsto x+y, n \mapsto x\},\{\},\{1 \leftarrow gcd(x+y,x)\}\rangle,\{\langle\langle\{m \mapsto y, n \mapsto x\},\{\},\{1 \leftarrow gcd(y,x)\}\rangle\}\rangle\}$

The proof of $\mathcal{C}$ following the previous inductive scheme consists of two induction bases corresponding to the first two components of the induction scheme: $gcd(0,x) = gcd(x,0)$ and $gcd(x,0) = gcd(0,x)$ which reduce to true by rules 1 and 2, and two induction steps corresponding to the last two components of the induction scheme: induction hypothesis 1: $gcd(x,y) = gcd(y,x)$ with conclusion 1: $gcd(x,x+y) = gcd(x+y,x)$ and induction hypothesis 2: $gcd(y,x) = gcd(x,y)$ with conclusion 2: $gcd(x+y,x) = gcd(x,x+y)$. Both conclusions of the induction steps reduce to the corresponding induction hypothesis by rules 3 and 4. $\diamond$

Generation of induction schemes depends on successful syntactic unification. But working with modular rewrite specifications with arithmetic parameters it is necessary to perform semantic unification modulo the arithmetic parameter. In [KS96b] it is suggested the use of decision algorithms for $\mathcal{PA}$ to accomplish $\mathcal{PA}$-unification in place of syntactic unification. The following example from [KS96b] illustrates the application of arithmetic decision algorithms when generating induction schemes by the cover-set method.

Example 4.3 Consider rules 6, 7 and 8 of example 3.1 specifying *divides* over the natural numbers presented as parameter of the whole specification. The cover-set of *divides* is used to prove the conjecture $divides(2,x) = not(\ divides(2,S(x)))$, generating the following induction scheme (for the first occurrence of *divides* in the conjecture):

$\{\ \ \langle\langle\{x \mapsto 0\},\{\},\{1 \leftarrow\ divides(2,0)\}\rangle,\{\}\rangle,$
$\langle\langle\{x \mapsto v\},\{v < 2 \wedge v \neq 0\},\{1 \leftarrow\ divides(2,v)\}\rangle,\{\}\rangle,$
$\langle\langle\{x \mapsto 2+v\},\{\},\{1 \leftarrow\ divides(2,2+v)\}\rangle,\{\langle\langle\{x \mapsto v\},\{\},\{1 \leftarrow\ divides(2,v)\}\rangle\}\rangle\ \ \}$

The induction basis given for this scheme consists of two cases. Firstly, it should be proved that $divides(2,0) = not(\ divides(2,S(0)))$, which can be done by reduction with rules 6 and 7. Observe that rule 7 applies since its condition $S(0) < 2 \wedge S(0) \neq 0$ (which is checked with the decision algorithm for $\mathcal{PA}$ restricting all variables to be non-negative) is true. Secondly, it should be proved that $divides(2,v) = not(\ divides(2,S(v)))$ *if* $v < 2 \wedge v \neq 0$. It reduces to $false = not(\ divides(2,S(v)))$ *if* $v < 2 \wedge v \neq 0$ applying rule 7. Observe that the right-hand side of the equality cannot be reduced, but using the decision algorithm for $\mathcal{PA}$ one can find a solution for v. Thus it should be proved that $false = not(\ divides(2,S(1)))$, which reduces to $false = not(\ true\ )$ applying rules 8 and 6. Note that this is done by finding $\mathcal{PA}$-matchings using the decision algorithm. The induction step consists of proving the conclusion hypothesis $divides(2,v+2) = not(\ divides(2,S(v+2)))$ under the induction hypothesis $divides(2,v) = not(\ divides(2,S(v)))$. The conclusion reduces to $divides(2,v) = not(\ divides(2,S(v+2)))$ applying rule 8, which matches with the left-hand side of the equality modulo $\mathcal{PA}$. In order to reduce $divides(2,S(v+2))$ it is also necessary a $\mathcal{PA}$-matching to apply rule 8, i.e., to detect that $S(v+2) = 2+S(v)$. In this way one obtains the induction hypothesis, completing the proof. $\diamond$

Searching for $\mathcal{PA}$-unifiers, $\mathcal{PA}$-matchings and solving arithmetic formulas with the decision algorithm for the unquantified $\mathcal{PA}$ enlarged with non-interpreted function and predicate symbols make extremely inefficient the generation of inductive proofs, however this is necessary and should be done in a rational manner, maintaining the efficiency of rewriting methods as suggested in [KS96b].

5 Conclusions

We have presented an improved version of the semi-decision algorithm for $\mathcal{PA}$ with enlarged function and predicate symbols originally developed by Shostak in [Sho79] and illustrated its applicability and suitability as complementary deduction mechanism when rewriting deduction techniques are applied in modular conditional rewrite specifications with arithmetic parameters. The main contribution of this paper is the improvement of Knuth-Bendix conditional completion process by eliminating critical pairs. Inconsistent and trivial critical pairs, generated during conditional completion, are checked by detecting inconsistency of their arithmetic conditions and by deciding their validity, respectively. Additionally we illustrate applications of arithmetic decision algorithms for proving inductive theorems by rewriting techniques when generating induction schemes by the cover-set method. Combining the decision algorithm for $\mathcal{PA}$ with conditional completion and with inductive deduction by rewriting makes rewriting very expensive, because of the high complexity of decision algorithms for arithmetic. In fact, decision algorithms for arithmetic should be applied in order to search $\mathcal{PA}$-unifiers and $\mathcal{PA}$-matchings, to give arithmetic solutions and to check validity and invalidity. This slows rewriting deduction considerably. As done in our examples, decision arithmetic algorithms should be applied in a rational manner in order to widen the scope of the rewrite-based deduction mechanisms and to maintain the efficiency. Kapur and Subramaniam [KS96b] have presented in detail how decision algorithms for $\mathcal{PA}$ can be judiciously combined with rewrite-based induction heuristics such as the cover-set method and generalization. With respect to the completion process, in [Aya93] a conditional procedure for conditional rewrite specifications with general decidable built-in parameters is presented, which can be adapted to arithmetic parameters as suggested in this work.

One of the most interesting potential applications of built-in decision algorithms for $\mathcal{PA}$ in rewrite-based deduction provers is the generation of intermediate lemmas necessary to conclude proofs of a given (inductive) conjecture. In [KS96a], an approach for speculating about intermediate lemmas is presented, whose scope can be enlarged by considerations over arithmetic parameters.

References

[AB92] J. Avenhaus and K. Becker. Conditional rewriting modulo a built-in algebra. SEKI-Report SR-92-11, Fachbereich Informatik, Universität Kaiserslautern, Postfach 3049, D-67653 Kaiserslautern (Germany), 1992.

[AM90] J. Avenhaus and K. Madlener. Term Rewriting and Equational Reasoning. In R. B. Banerji, editor, *Formal Techniques in Artificial Intelligence*, chapter 1, pages 1–43. Elsevier Science Publishers B. V. (North-Holland), 1990.

[Aya93] M. Ayala. *Expressiveness of Conditional Equational Systems with Built-in Predicates.* PhD thesis, Universität Kaiserslautern, Kaiserslautern (Germany), December 1993.

[Aya95] M. Ayala. A Deductive Calculus for Conditional Equational Systems with Built-in Predicates as Premises — Extended Abstract —. In *XV International Conference of the Chilean Computer Science Society, Arica, Chile*, pages 25–36, November 1995.

[Aya97] M. Ayala. A Decision Procedure for Conditional Rewriting Systems with Built-in Predicates. In *Anais XXIV Seminário Integrado de Software e Hardware, Brasília, Brazil*, pages 387–398, August 1997.

[Bec94] K. Becker. *Rewrite Operationalization of Clausal Specifications with Predefined Structures.* PhD thesis, Universität Kaiserslautern, Kaiserslautern (Germany), April 1994.

[Coo72] D. C. Cooper. Theorem Proving in Arithmetic without Multiplication. *Machine Intelligence*, 7:91–99, 1972.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 2, chapter 6, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990.

[DO90] N. Dershowitz and M. Okada. A Rationale for Conditional Equational Programming. *Theoretical Computer Science*, 75:111–138, 1990.

[GA96] L. M. R. Gadelha and M. Ayala. Aplicação de Métodos de Programação Linear Inteira para Decisão na Teoria da Aritmética de Presburger. In *Anáis do XIX Congresso Nacional de Matemática Aplicada e Computacional, Goiânia, Brazil, September, 1996*, pages 400–401, 1996. Extended version available in *http://www.mat.unb.br/~gadelha*. In Portuguese.

[Gan91] H. Ganzinger. A Completion Procedure for Conditional Equations. *Journal of Symbolic Computation*, 11:51–81, 1991.

[Hsi85] J. Hsiang. Refutational Theorem Proving Using Term-Rewriting Systems. *Artificial Intelligence*, 25(2):255–300, 1985.

[Klo92] J. W. Klop. Term Rewriting Systems. In S. Abramski, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, chapter 1, pages 1–116. Oxford Science Publications, 1992.

[KS96a] D. Kapur and M. Subramaniam. Lemma Discovery in Automating Induction. In M. A. McRobbie and J. K. Slaney, editors, *Proc. of the* 13^{th} *Int. Conference on Automated Deduction CADE-13, New Brunswick, NJ, USA*, volume 1104 of *LNCS*, pages 538–552. Springer, July 1996.

[KS96b] D. Kapur and M. Subramaniam. New Uses of Linear Arithmetic in Automated Theorem Proving by Induction . *Journal of Automated Reasoning*, 16(1/2), 1996.

[KZ89] D. Kapur and H. Zhang. An overview of Rewrite Rule Laboratory (RRL). In N. Dershowitz, editor, *Proc. Third Int. Conf. on Rewriting techniques and Applications, Chapel-Hill, NC*, volume 355 of *LNCS*. Springer, April 1989.

[Opp78] D. C. Oppen. A $2^{2^{2^{pn}}}$ Upper Bound on the Complexity of Presburger Arithmetic. *Journal of Computer and System Sciences*, 16:323–332, 1978.

[Pla93] D. Plaisted. Equational Reasoning and Term Rewriting Systems. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*. Oxford University Press, 1993.

[Pre29] M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *1. Kongres matematyków krajow slowiańskich, Warsaw*, pages 92–101, 1929. In German.

[Sho79] R. E. Shostak. A Practical Decision Procedure for Arithmetic with Function Symbols. *Journal of the Association for Computing Machinery*, 26(2):351–360, April 1979.

[Siv89] G. Sivakumar. *Proofs and Computations in Conditional Equational Theories*. PhD thesis, University of Illinois at Urbana-Champaign, 1989.

[Vor88] S. G. Vorobyov. On the Arithmetic Inexpressiveness of Term Rewriting Systems. In *Third Symp. on Logics in Computer Sciences, Edinburgh Scotland*, pages 212–217, July 1988.

[Vor89] S. G. Vorobyov. Conditional rewrite rule systems with Built-in Arithmetic and Induction. In N. Dershowitz, editor, *Proc. Third Int. Conf. on Rewriting techniques and Applications, Chapel-Hill, NC*, volume 355 of *LNCS*, pages 492–512. Springer, April 1989.

[ZKK88] H. Zhang, D. Kapur, and M. S. Krishnamoorthy. A Mechanizable Induction Principle for Equational Specifications. In *Proc. of the* 9^{th} *Int. Conference on Automated Deduction CADE-9* , volume 310 of *LNCS*, pages 250–265. Springer, 1988.