Folding Transformation Rules for Constraint Logic Programs

Valerio Senni[1], Alberto Pettorossi[1], and Maurizio Proietti[2]

(1) DISP, University of Roma Tor Vergata, Via del Politecnico 1, I-00133 Roma, Italy
`{senni,pettorossi}@disp.uniroma2.it`
(2) IASI-CNR, Viale Manzoni 30, I-00185 Roma, Italy
`proietti@iasi.rm.cnr.it`

Abstract. We consider the folding transformation rule for constraint logic programs. We propose an algorithm for applying the folding rule in the case where the constraints are linear equations and inequations over the rational or the real numbers. Basically, our algorithm consists in reducing a rule application to the solution of one or more systems of linear equations and inequations. We also introduce two variants of the folding transformation rule. The first variant combines the folding rule with the clause splitting rule, and the second variant eliminates the existential variables of a clause, that is, those variables which occur in the body of the clause and not in its head. Finally, we present the algorithms for applying these variants of the folding rule.

1 Introduction

Rule-based program transformation is a program development technique which has been first proposed by Burstall and Darlington in the context of functional programming [4], and then it has been extended to logic programming [19] and to other programming paradigms as well (see [13] for an overview).

In this paper we consider constraint logic programs [8] and the unfold/fold transformation rules presented in [3,5,7,10]. In particular, we focus our investigations on the folding rule, which can be defined (in a declarative style) as follows.

Let (i) H and K be atoms, (ii) c and d be constraints, and (iii) G and B be goals (that is, conjunctions of literals). Given a clause γ: $H \leftarrow c \wedge G$ and a clause δ: $K \leftarrow d \wedge B$, if there exist a constraint e, a substitution ϑ, and a goal R such that $H \leftarrow c \wedge G$ is equivalent (in a given theory of constraints) to $H \leftarrow e \wedge (d \wedge B)\vartheta \wedge R$, then γ is folded into the clause η: $H \leftarrow e \wedge K\vartheta \wedge R$.

In the literature, the folding rule is presented with respect to a generic theory of constraints and no algorithm is provided to determine whether or not the suitable e, ϑ, and R needed for applying that rule exist. In this paper we assume that the constraints are linear equations and inequations over the rational numbers (but the techniques we will present are valid without significant changes also when the equations and inequations are over the real numbers), and we propose an algorithm based on linear algebra and term rewriting techniques for computing e, ϑ, and R, if they exist. For instance, let us consider the clauses:

γ: $p(X,Y) \leftarrow X>1 \ \wedge \ X>T \ \wedge \ q([\,],T) \ \wedge \ r(Y)$

δ: $s(U,V,A) \leftarrow U>1 \ \wedge \ V>0 \ \wedge \ U>W \ \wedge \ q(A,W)$

and suppose that we want to fold γ using δ. Our folding algorithm computes: (i) the constraint e: $X>1 \wedge X \geq U$, (ii) the substitution ϑ: $\{A/[\,], W/T\}$, and (iii) the goal R: $r(Y)$, and the clause derived by folding γ using δ is:

η: $p(X,Y) \leftarrow X>1 \ \wedge \ X \geq U \ \wedge \ s(U,V,[\,]) \ \wedge \ r(Y)$

(The correctness of folding will be stated in Section 3. At this point the reader can check it by unfolding the atom $s(U,V,[\,])$ in the clause η using δ and, indeed, that unfolding returns clause γ, modulo equivalence of constraints.)

In general, there may be zero or more triples $\langle e, \vartheta, R\rangle$ that satisfy the conditions for the applicability of the folding rule. For this reason, our folding algorithm is nondeterministic and thus, given the clauses γ and δ, it may return, in different computations, different folded clauses.

In this paper we also introduce two variants of the standard folding rule presented above, which are often very useful for transforming programs, and we propose two algorithms for their application.

The first variant of the folding rule corresponds to some applications of the *clause splitting* rule [7] followed by some applications of the standard folding rule. Clause splitting consists in replacing a clause $H \leftarrow c \wedge G$ by two clauses of the form: $H \leftarrow c \wedge d_1 \wedge G$ and $H \leftarrow c \wedge d_2 \wedge G$ such that in the given theory of constraints, the universal closure of $d_1 \vee d_2$ is equivalent to *true*. This variant of the folding rule combines in a single rule application: (i) several applications of the clause splitting rule, by which from clause γ we derive the n clauses $\gamma_1, \ldots, \gamma_n$, and (ii) n subsequent applications of the standard folding rule, by which we fold the clauses $\gamma_1, \ldots, \gamma_n$ using clause δ and we derive the n clauses $\eta_1, \ldots, \eta_n$. In the paper we will also propose an algorithm for determining the suitable applications of the clause splitting rule which allow the subsequent applications of the standard folding rule.

The second variant of the standard folding rule eliminates the *existential variables* which occur in the clause to be folded [14,15]. Recall that the existential variables are those variables which occur in the body of a clause but not in its head.

The paper is structured as follows. In Section 2 we recall some basic definitions concerning constraint logic programs. In Sections 3, 4, and 5 we introduce the standard folding rule [3,5,7,10] and two variants of this rule. We present the three algorithms for applying the folding rule and its two variants. For these algorithms we also prove the soundness and completeness results with respect to the declarative specifications of the folding rules. Finally, in Section 6 we discuss the related work and we indicate some directions for future investigation.

2 Preliminary Definitions

In this section we recall some basic definitions concerning constraint logic programs, where the constraints are conjunctions of linear equations and inequations over the rational numbers. As already mentioned, the results we will present in the following sections, are valid also when the constraints are conjunctions of linear equations and inequations over the real numbers. For notions not defined here the reader may refer to [8,9].

Let us consider a first order language $\mathcal{L}$ given by a set *Var* of variables, a set *Fun* of function symbols, and a set *Pred* of predicate symbols. We assume that the function symbols $+$ and $\cdot$ denoting addition and multiplication, respectively, belong to *Fun*, and every rational number in $\mathbb{Q}$ is a constant symbol (that is, a 0-ary function symbol) belonging to *Fun*. We also assume that the predicate symbols $\geq$, $>$, and $=$ denoting inequality, strict inequality, and equality, respectively, belong to *Pred*.

In order to distinguish terms representing rational numbers from other terms, we assume that $\mathcal{L}$ is a typed language [9]. We have two basic types: `rat`, that is, the type of rational numbers, and `tree`, that is, the type of finite trees. We also consider types constructed from basic types by using the type constructors $\times$ and $\rightarrow$. A variable $X \in \mathit{Var}$ has either type `rat` or `tree`. We denote by $\mathit{Var}_{\texttt{rat}}$ and $\mathit{Var}_{\texttt{tree}}$ the set of variables of type `rat` and `tree`, respectively. A predicate symbol of arity n and a function symbol of arity n in $\mathcal{L}$ have types of the form $\tau_1 \times \ldots \times \tau_n$ and $\tau_1 \times \ldots \times \tau_n \rightarrow \tau_{n+1}$, respectively, for some types $\tau_1, \ldots, \tau_n, \tau_{n+1} \in \{\texttt{rat}, \texttt{tree}\}$. In particular, the predicate symbols $\geq$, $>$, and $=$ have type $\texttt{rat} \times \texttt{rat}$, the function symbols $+$ and $\cdot$ have type $\texttt{rat} \times \texttt{rat} \rightarrow \texttt{rat}$, and the rational numbers have type `rat`. The function symbols in $\{+, \cdot\} \cup \mathbb{Q}$ are the only symbols whose type is $\tau_1 \times \ldots \times \tau_n \rightarrow \texttt{rat}$, for some types $\tau_1, \ldots, \tau_n$.
A *term of type* `rat` is a *linear polynomial* and has the following syntax:

$$p ::= \ a \mid X \mid a{\cdot}X \mid p_1 + p_2$$

where a is a rational number and X is a variable in $\mathit{Var}_{\texttt{rat}}$. We will also write the term $a{\cdot}X$ as aX. A *term of type* `tree` has the following syntax:

$$t ::= \ X \mid f(t_1, \ldots, t_n)$$

where X is a variable in $\mathit{Var}_{\texttt{tree}}$, f is a function symbol of type $\tau_1 \times \ldots \times \tau_n \rightarrow \texttt{tree}$, and $t_1, \ldots, t_n$ are terms of type $\tau_1, \ldots, \tau_n$, respectively. A *term* is either a term of type `rat` or a term of type `tree`.

A *constraint* is a finite conjunction of *atomic constraints*, which are linear equations and inequations over the rational numbers. Constraints have the following syntax:

$$c ::= \ p_1 \geq p_2 \mid p_1 > p_2 \mid p_1 = p_2 \mid c_1 \wedge \ldots \wedge c_n$$

When $n = 0$ we write $c_1 \wedge \ldots \wedge c_n$ as *true*. We denote by $\mathit{LIN}_{\mathbb{Q}}$ the set of all constraints.

An *atom* is of the form $r(t_1, \ldots, t_n)$, where r is a predicate symbol of type $\tau_1 \times \ldots \times \tau_n$ not in $\{\geq, >, =\}$, and $t_1, \ldots, t_n$ are terms of type $\tau_1, \ldots, \tau_n$, respectively. A *literal* is either an atom or a negated atom. An atom is also called a *positive literal* and a negated atom is also called a *negative literal.* A *goal* is a conjunction $L_1 \wedge \ldots \wedge L_n$ of literals, with $n \geq 0$. Similarly to the case of constraints, we will denote by *true* the conjunction of 0 literals. A *constrained goal* is a conjunction $c \wedge G$, where c is a constraint and G is a goal. A *clause* is of the form $H \leftarrow c \wedge G$, where H is an atom and $c \wedge G$ is a constrained goal. A *constraint logic program* is a set of clauses. A *formula* of the language $\mathcal{L}$ is constructed as usual in first order logic from the symbols of $\mathcal{L}$ by using the logical connectives $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftarrow$, $\leftrightarrow$, and the quantifiers $\exists$, $\forall$.

If e is a term or a formula then by $\mathit{Vars}_{\mathtt{rat}}(e)$ and $\mathit{Vars}_{\mathtt{tree}}(e)$ we denote, respectively, the set of variables of type `rat` and of type `tree` occurring in e. By $\mathit{Vars}(e)$ we denote the set $\mathit{Vars}_{\mathtt{rat}}(e) \cup \mathit{Vars}_{\mathtt{tree}}(e)$. By $\mathit{Vars}(e_1, e_2)$ we denote the set $\mathit{Vars}(e_1) \cup \mathit{Vars}(e_2)$. By $\mathit{FVars}(e)$ we denote the set of free variables of e. Given a constraint c occurring in the body of a clause γ, a variable which occurs in c and not elsewhere in γ, is said to be a *local variable of* c *in* γ. Given a clause γ: $H \leftarrow c \wedge G$, by $\mathit{EVars}(\gamma)$ we denote the set $\mathit{Vars}(c \wedge G) - \mathit{Vars}(H)$ of the *existential variables* of γ. Given a set $X = \{X_1, \ldots, X_n\}$ of variables and a formula φ, by $\forall X\, \varphi$ we denote the formula $\forall X_1 \ldots \forall X_n\, \varphi$ and by $\exists X\, \varphi$ we denote the formula $\exists X_1 \ldots \exists X_n\, \varphi$. By $\forall(\varphi)$ and $\exists(\varphi)$ we denote the *universal closure* and the *existential closure* of φ, respectively.

In the following we will use the notion of substitution as defined in [9] with the following extra condition: for any substitution $\{X_1/t_1, \ldots, X_n/t_n\}$, for $i = 1, \ldots, n$, the type of X_i is equal to the type of t_i.

Let $\mathcal{L}_{\mathtt{rat}}$ denote the sublanguage of $\mathcal{L}$ given by the set $\mathit{Var}_{\mathtt{rat}}$ of variables, the set $\{+, \cdot\} \cup \mathbb{Q}$ of function symbols, and the set $\{\geq, >, =\}$ of predicate symbols. The formulas of $\mathcal{L}_{\mathtt{rat}}$ are the formulas of $\mathcal{L}$ where all variables, function symbols, and predicate symbols belong to $\mathcal{L}_{\mathtt{rat}}$. The interpretation $\mathcal{Q}$ for $\mathcal{L}_{\mathtt{rat}}$ is defined as follows: (i) $\mathcal{Q}$ assigns to the function symbols $+$ and $\cdot$ the usual addition and multiplication operations in $\mathbb{Q}$, (ii) $\mathcal{Q}$ assigns to every $a \in \mathbb{Q}$ the element a itself, and (iii) $\mathcal{Q}$ assigns to the predicate symbols $\geq$, $>$, and $=$ the usual inequality, strict inequality, and equality relations on $\mathbb{Q}$, respectively. For a formula φ of $\mathcal{L}_{\mathtt{rat}}$ (in particular, for a constraint), the satisfaction relation $\mathcal{Q} \models \varphi$ is defined as usual in first order logic. Recall that the problem of checking, for any formula φ of $\mathcal{L}_{\mathtt{rat}}$, whether or not $\mathcal{Q} \models \varphi$ holds, is decidable.

By $D_{\mathtt{rat}}$ we denote the set $\mathbb{Q}$ of the rational numbers and by $D_{\mathtt{tree}}$ we denote the set $\mathbb{H} - \mathbb{Q}$, where $\mathbb{H}$ is the set of ground terms constructed from the function symbols in $\mathit{Fun} - \{+, \cdot\}$. A *$\mathcal{Q}$-interpretation* is an interpretation I for the typed language $\mathcal{L}$ defined as follows. For $\tau_1, \ldots, \tau_n, \tau_{n+1} \in \{\mathtt{rat}, \mathtt{tree}\}$, (i) I assigns to a function symbol of type $\tau_1 \times \ldots \times \tau_n \rightarrow \tau_{n+1}$ a function from $D_{\tau_1} \times \ldots \times D_{\tau_n}$ to $D_{\tau_{n+1}}$. In particular, (i.1) to any function symbol in $\{+, \cdot\} \cup \mathbb{Q}$, I assigns the same function as $\mathcal{Q}$, and (i.2) to any function symbol f of type $\tau_1 \times \ldots \times \tau_n \rightarrow \mathtt{tree}$, I assigns the function that maps $\langle d_1, \ldots, d_n \rangle \in D_{\tau_1} \times \ldots \times D_{\tau_n}$ to $f(d_1, \ldots, d_n) \in D_{\mathtt{tree}}$. (ii) I assigns to any predicate symbol of type $\tau_1 \times \ldots \times \tau_n$ a relation on $D_{\tau_1} \times \ldots \times D_{\tau_n}$. In particular, to the symbols $\geq$, $>$, and $=$, I assigns the same relation as $\mathcal{Q}$. For any formula φ of $\mathcal{L}$ (and thus, for a constraint logic program), the satisfaction relation $I \models \varphi$ is defined as usual in typed first order logic. For any $\mathcal{Q}$-interpretation I and formula φ of $\mathcal{L}_{\texttt{rat}}$, we have that $I \models \varphi$ iff $\mathcal{Q} \models \varphi$. We say that a $\mathcal{Q}$-interpretation I is a *$\mathcal{Q}$-model* of a program P if for every clause $\gamma \in P$ we have that $I \models \forall(\gamma)$. Similarly to the case of logic programs, we can define *stratified* constraint logic programs and we can show that every stratified program P has a *perfect* $\mathcal{Q}$-model [7,8,10], denoted by $M(P)$.

A *solution* of a set C of constraints is a ground substitution σ of the form $\{X_1/a_1, \ldots, X_n/a_n\}$, where $\{X_1, \ldots, X_n\} = \mathit{Vars}(C)$ and $a_1, \ldots, a_n \in \mathbb{Q}$, such that $\mathcal{Q} \models c\,\sigma$ for every $c \in C$. A set of constraints is said to be *satisfiable* if it has a solution. We assume that we are given a function *solve* that takes a set C of constraints in $\mathit{LIN}_{\mathbb{Q}}$ as input and returns a solution σ of C, if C is satisfiable, and **fail** otherwise. The function *solve* can be implemented by using, for instance, the Fourier-Motzkin algorithm or the Khachiyan algorithm [17].

We assume that we are also given a function *project* such that for every constraint $c, d \in \mathit{LIN}_{\mathbb{Q}}$ and for every finite set of variables $X \subseteq \mathit{Var}_{\texttt{rat}}$, if $\mathit{project}(c, X) = d$ then $\mathcal{Q} \models \forall X\,((\exists Y\,c) \leftrightarrow d)$, where $Y = \mathit{Vars}(c) - X$ and $\mathit{Vars}(d) \subseteq X$. Also the *project* function can be implemented by using the Fourier-Motzkin algorithm.

A clause $\gamma : H \leftarrow c \wedge G$ is said to be in *normal form* if (i) the terms of type `rat` occurring in G are distinct existential variables, and (ii) c has no local variables in γ. It is always possible to transform any clause γ into a clause γ_1 in normal form such that γ and γ_1 have the same $\mathcal{Q}$-models. (In particular, given a clause γ, all local variables of a constraint in γ can be eliminated by applying the *project* function.) The clause γ_1 is called *a normal form of γ*.

Since every clause can be transformed into an equivalent clause in normal form, when presenting the folding rule and the corresponding algorithm for its application we will assume, without loss of generality, that the clauses are in normal form.

Given two clauses γ_1 and γ_2, we write $\gamma_1 \cong \gamma_2$ if there exist a normal form $H \leftarrow c_1 \wedge B_1$ of γ_1, a normal form $H \leftarrow c_2 \wedge B_2$ of γ_2, and a variable renaming ρ such that: (1) $H = H\rho$, (2) $B_1 =_{AC} B_2\rho$, and (3) $\mathcal{Q} \models \forall\ (c_1 \leftrightarrow c_2\rho)$, where $=_{AC}$ denotes equality modulo the equational theory of associativity and commutativity of conjunction. We refer to this theory as the *$AC_\wedge$ theory* [1].

Proposition 1. *(i) $\cong$ is an equivalence relation.*
(ii) If $\gamma_1 \cong \gamma_2$ then, for every $\mathcal{Q}$-interpretation I, $I \models \gamma_1$ iff $I \models \gamma_2$.
(iii) If γ_2 is a normal form of γ_1 then $\gamma_1 \cong \gamma_2$.
(iv) Let γ_1 be $H \leftarrow c \wedge B$ and γ_2 be $H \leftarrow d \wedge B$, then $\gamma_1 \cong \gamma_2$ iff $\mathcal{Q} \models \forall\,(\exists V_1\, c \leftrightarrow \exists V_2\, d)$, where V_1 is the set of local variables of c in γ_1 and V_2 is the set of local variables of d in γ_2.

The $\cong$ equivalence relation can be extended to sets of clauses as follows. Let $\{\gamma_1, \ldots, \gamma_m\}$ and $\{\delta_1, \ldots, \delta_n\}$ be two sets of clauses. We write $\{\gamma_1, \ldots, \gamma_m\} \cong \{\delta_1, \ldots, \delta_n\}$ if there exist $H, B_1, B_2, c_1, \ldots, c_m, d_1, \ldots, d_n$ such that, for $i = 1, \ldots, m$, $H \leftarrow c_i \wedge B_1$ is a normal form of γ_i, for $j = 1, \ldots, n$, $H \leftarrow d_j \wedge B_2$ is a normal form of δ_j, and there exists a variable renaming ρ such that: (1) $H = H\rho$, (2) $B_1 =_{AC} B_2\rho$, and (3) $\mathcal{Q} \models \forall\ ((c_1 \vee \ldots \vee c_m) \leftrightarrow (d_1 \vee \ldots \vee d_n)\rho)$.

3 The Standard Folding Rule

In the literature there are several, slightly different versions of the folding transformation rule for constraint logic programs [3,5,7,10]. These versions differ on the applicability conditions and on the number of clauses that can be folded by a single rule application. In this section we introduce the rule $\mathbf{Fold_1}$, whose applicability conditions are the ones given in [7]. However, unlike the folding rule presented in [7], the rule $\mathbf{Fold_1}$ can be applied to one clause at a time, and in this respect it is similar to the folding rule considered in [3,5]. The general case, where $n(\geq 1)$ clauses can be folded at once, can be addressed as a straightforward generalization of the methods we will present here.

Definition 1 (Rule $\mathbf{Fold_1}$). Let γ and δ be clauses of the form

$\gamma\colon\ H \leftarrow c \wedge G$
$\delta\colon\ K \leftarrow d \wedge B$

such that γ and δ are in normal form and without variables in common. Suppose that there exist a constraint e, a substitution ϑ, and a goal R such that:
(1) $\gamma\ \cong\ H \leftarrow e \wedge d\vartheta \wedge B\vartheta \wedge R$;
(2) for every variable X in $EVars(\delta)$, the following conditions hold: (2.i) $X\vartheta$ is a variable not occurring in $\{H, e, R\}$, and (2.ii) $X\vartheta$ does not occur in the term $Y\vartheta$, for every variable Y occurring in $d \wedge B$ and different from X.
By *folding clause γ using clause δ* we derive the clause η: $H \leftarrow e \wedge K\vartheta \wedge R$.

In Theorem 1 below we establish the correctness of the folding rule $\mathbf{Fold_1}$ w.r.t. the perfect model semantics. That result follows immediately from [18].

A *transformation sequence* is a sequence $P_0, \ldots, P_n$ of programs such that, for $k = 0, \ldots, n-1$, program P_{k+1} is derived from program P_k by an application of one of the following transformation rules: *definition*, *unfolding* (w.r.t. *positive* literals), *folding* (Here we refer to the definition rule and the unfolding rule as they are presented in [7]). By $Defs_n$ we denote the set of clauses introduced by applying the definition rule during the construction of $P_0, \ldots, P_n$.

Program P_{k+1} is derived from program P_k by an application of the folding rule $\mathbf{Fold_1}$ if $P_{k+1} = (P_k - \{\gamma\}) \cup \{\eta\}$, where γ is a clause in P_k, δ is a clause in $Defs_k$, and η is the clause derived by folding γ using δ as described in Definition 1.

Theorem 1. *Let P_0 be a stratified program and let $P_0, \ldots, P_n$ be a transformation sequence. Suppose that, for $k = 0, \ldots, n-1$, if P_{k+1} is derived from P_k by folding clause γ using clause δ in $Defs_k$, then there exists j, with $0 < j < n$, such that $\delta \in P_j$ and program P_{j+1} is derived from P_j by unfolding δ w.r.t. a positive literal in its body. Then $P_0 \cup Defs_n$ and P_n are stratified and $M(P_0 \cup Defs_n) = M(P_n)$.*

Now we will present an algorithm for determining whether or not a clause $\gamma : H \leftarrow c \wedge G$ can be folded using a clause $\delta : K \leftarrow d \wedge B$, according to Definition 1. Our folding algorithm finds, if they exist, a substitution ϑ, a constraint e, and a goal R that satisfy the clause equivalence $H \leftarrow c \wedge G\ \cong\ H \leftarrow e \wedge d\vartheta \wedge B\vartheta \wedge R$

of Point (1) of Definition 1 with the extra conditions of Point (2) of that same definition. If it is not possible to fold clause γ using clause δ, our algorithm returns **fail**.

Note that since Definition 1 does not determine in a unique way ϑ, e, and R, our folding algorithm is nondeterministic.

Our algorithm can be viewed as a solver of a matching problem [20] and it consists of two procedures which we will present below: (i) the *goal matching procedure*, called **GM**, which matches goals modulo the AC$_\wedge$ theory [1], and (ii) the *constraint matching procedure*, called $\mathbf{CM}_1$, which matches constraints modulo the theory of constraints.

Now let us present the goal matching procedure **GM**, which given two clauses $\gamma\colon H \leftarrow c \wedge G$ and $\delta\colon K \leftarrow d \wedge B$, either computes, if they exist, two new clauses $\gamma'\colon H \leftarrow c \wedge B\vartheta_1 \wedge R$ and $\delta'\colon K\vartheta_1 \leftarrow d\vartheta_1 \wedge B\vartheta_1$ such that G is equal to $B\vartheta_1 \wedge R$ modulo the AC$_\wedge$ theory, for some substitution ϑ_1 and goal R, or returns **fail**, otherwise.

Goal Matching Procedure: GM

Input: two clauses $\gamma\colon H \leftarrow c \wedge G$ and $\delta\colon K \leftarrow d \wedge B$ in normal form without variables in common.

Output: (A) two clauses $\gamma'\colon H \leftarrow c \wedge B\vartheta_1 \wedge R$ and $\delta'\colon K\vartheta_1 \leftarrow d\vartheta_1 \wedge B\vartheta_1$ in normal form such that: (1) $\gamma \cong \gamma'$ and (2) for every variable X in $EVars(\delta)$, the following conditions hold: (2.i) $X\vartheta_1$ is a variable not occurring in $\{H, R\}$, and (2.ii) $X\vartheta_1$ does not occur in the term $Y\vartheta_1$, for every variable Y occurring in $d \wedge B$ and different from X, if such two clauses γ' and δ' exist, and (B) **fail**, otherwise.

Step 1. Compute an injection μ from the multiset $\overline{B}$ of the literals occurring in B, to the multiset $\overline{G}$ of the literals occurring in G such that: (i) μ is different from any previously computed injection from $\overline{B}$ to $\overline{G}$, if any, and (ii) for all $L \in \overline{B}$, L and $\mu(L)$ are either both positive or both negative literals, and have the same predicate symbol with the same arity. If there exists no such an injection μ then return **fail**.

Step 2. Consider the set of equations $\overline{S} = \{L = \mu(L) \mid L \in \overline{B}\}$. Transform the set $\overline{S}$ by applying as long as possible the following rewriting rules:

(i) $\{\neg A_1 = \neg A_2\} \cup \overline{S} \Longrightarrow \{A_1 = A_2\} \cup \overline{S}$;
(ii) $\{a(s_1, \ldots, s_n) = a(t_1, \ldots, t_n)\} \cup \overline{S} \Longrightarrow \{s_1 = t_1, \ldots, s_n = t_n\} \cup \overline{S}$;
(iii) $\{a(s_1, \ldots, s_m) = b(t_1, \ldots, t_n)\} \cup \overline{S} \Longrightarrow$ **fail**, if a is different from b;
(iv) $\{a(s_1, \ldots, s_n) = X\} \cup \overline{S} \Longrightarrow$ **fail**, if X is a variable;
(v) $\{X = t_1\} \cup \overline{S} \Longrightarrow$ **fail**, if X is a variable and in the current set $\overline{S}$ of equations there is an equation $X = t_2$ such that t_1 is different from t_2.

Step 3. Let R be the conjunction of all literals in $\overline{G} - \{\mu(L) \mid L \in \overline{B}\}$ and S be the final set of equations (possibly, **fail**) which is derived at the end of Step 2.

IF S is not **fail** and for every equation $X = t$ in S such that $X \in EVars(\delta)$ we have that: (i) the term t is a variable not occurring in $\{H, R\}$ and (ii) there is no $Y \in Vars(d \wedge B)$ different from X such that $Y = r$ is an equation in

S and $t \in \mathit{Vars}(r)$ THEN return the clauses $\gamma' : H \leftarrow c \wedge B\vartheta_1 \wedge R$ and $\delta' : K\vartheta_1 \leftarrow d\vartheta_1 \wedge B\vartheta_1$, where ϑ_1 is the substitution $\{X/t \mid X = t \in S\} \cup \{V_1/s_1, \ldots, V_\ell/s_\ell\}$, where $\{V_1, \ldots, V_\ell\} = \mathit{Vars}_{\mathtt{tree}}(K) - \mathit{Vars}(B)$ and $s_1, \ldots, s_\ell$ are arbitrary ground terms of type `tree` ELSE go to Step 1.

Note that the goal matching procedure **GM** is nondeterministic because, in general, more than one injection μ may lead to the construction of a matching substitution ϑ_1 and different mappings μ may determine different matching substitutions.

Moreover, the goal matching procedure **GM** has a 'generate-and-test' structure in the sense that at Step 1 it looks for an injection μ and then at Step 3 it tests whether or not that injection μ allows the construction of the matching substitution ϑ_1. It could be desirable to avoid the inefficiency due to that 'generate-and-test' structure, but one should recall that the problem of matching modulo the equational theory $\mathrm{AC}_\wedge$ has been shown to be NP-complete [1,2].

Note also that, since the input clauses γ and δ are in normal form, the substitution ϑ_1 computed by the goal matching procedure **GM** is a renaming substitution when restricted to the variables of type `rat`.

Finally, note that the arbitrary ground substitution $\{V_1/s_1, \ldots, V_\ell/s_\ell\}$ for the variables of $\mathit{Vars}_{\mathtt{tree}}(K) - \mathit{Vars}(B)$ can be chosen before executing the **GM** procedure, so that **GM** does *not* compute distinct substitutions ϑ_1 differing only on the choice of $\{V_1/s_1, \ldots, V_\ell/s_\ell\}$.

The following theorem states that the goal matching procedure **GM** is sound and complete, that is, **GM** finds a suitable matching substitution ϑ_1 and a goal R if and only if they exist.

Theorem 2 (Termination, Soundness, and Completeness of the Goal Matching Procedure). *Let $\gamma : H \leftarrow c \wedge G$ and $\delta : K \leftarrow d \wedge B$ be two clauses in normal form and without variables in common, given in input to the goal matching procedure* **GM**. *Then the following properties hold:*
(i) **GM** *terminates;*
(ii) If the output of **GM** *is the pair of clauses $\gamma' : H \leftarrow c \wedge B\vartheta_1 \wedge R$ and $\delta' : K\vartheta_1 \leftarrow d\vartheta_1 \wedge B\vartheta_1$, then*

1. *γ' and δ' are in normal form,*
2. *$\gamma \cong \gamma'$, and*
3. *the substitution ϑ_1 is such that, for every variable X in $\mathit{EVars}(\delta)$, the following conditions hold: (i) $X\vartheta_1$ is a variable not occurring in $\{H, R\}$, and (ii) $X\vartheta_1$ does not occur in the term $Y\vartheta_1$, for every variable Y occurring in $d \wedge B$ and different from X;*

(iii) If there exist a substitution ϑ_1 and a goal R such that:

1. *$\gamma \cong H \leftarrow c \wedge B\vartheta_1 \wedge R$, and*
2. *for every variable X in $\mathit{EVars}(\delta)$, the following conditions hold: (i) $X\vartheta_1$ is a variable not occurring in $\{H, R\}$, and (ii) $X\vartheta_1$ does not occur in the term $Y\vartheta_1$, for every variable Y occurring in $d \wedge B$ and different from X,*

then the output of **GM** *is not* **fail**.

Now we present the constraint matching procedure $\mathbf{CM}_1$ which takes in input the two clauses γ' and δ' which are produced in output by the goal matching procedure. When presenting the procedure $\mathbf{CM}_1$ we will assume that the clauses γ' and δ' are of the following form:

$\gamma' : H \leftarrow c \wedge B' \wedge R$

$\delta' : K' \leftarrow d' \wedge B'$.

The constraint matching procedure $\mathbf{CM}_1$ returns either a clause γ'': $H \leftarrow e \wedge d' \wedge B' \wedge R$ such that $\gamma' \cong \gamma''$ and $EVars(\delta') \cap Vars(e) = \emptyset$ (see Condition 2.i of Definition 1), or **fail**.

Note that, by Point (iv) of Proposition 1, the equivalence $\gamma' \cong \gamma''$ holds iff $\mathcal{Q} \models \forall(\exists W(e \wedge d') \leftrightarrow c)$, where W is the set of the local variables of $e \wedge d'$ in γ''. As a consequence of this fact and the fact that, without loss of generality, we may assume that the constraint e has no local variables in γ'', one can show that $Vars(e) = Vars(c, d') - (Vars(c) \cap Vars(d'))$.

Now let us consider the formula $\widehat{e}$ defined as follows.

Definition 2. *Let γ' and δ' be the output clauses of the goal matching procedure* $\mathbf{GM}$. *By $\widehat{e}$ we denote the formula $\forall Z\,(d' \rightarrow c)$, where: (i) c and d' are the constraints occurring in the clauses γ' and δ', respectively, and (ii) Z is the set of variables $Vars(c) \cap Vars(d')$.*

From Definition 2, we immediately get that $FVars(\widehat{e}) = Vars(c, d') - (Vars(c) \cap Vars(d'))$. In the following Lemma 1 we show that, among all formulas e in $\mathcal{L}_{\texttt{rat}}$ such that $\mathcal{Q} \models \forall((\exists W(e \wedge d')) \leftrightarrow c)$, where W is the set of the free variables of $e \wedge d'$ not occurring in $\{H, B' \wedge R\}$, the formula $\widehat{e}$ is the most general one in the sense that $\mathcal{Q} \models \forall(e \rightarrow \widehat{e})$.

Lemma 1. *Let $\gamma' : H \leftarrow c \wedge B' \wedge R$ and $\delta' : K' \leftarrow d' \wedge B'$ be the two clauses in normal form which are the output clauses of the goal matching procedure* $\mathbf{GM}$. *Let Y be $FVars(\widehat{e} \wedge d') - Vars(H, B' \wedge R)$. The following properties hold:*

1. *$\mathcal{Q} \models \forall((\exists Y(\widehat{e} \wedge d')) \rightarrow c)$;*
2. *For all formulas e in $\mathcal{L}_{\texttt{rat}}$ such that $FVars(e) = FVars(\widehat{e})$,*
   if $\mathcal{Q} \models \forall((\exists Y(e \wedge d')) \rightarrow c)$ then $\mathcal{Q} \models \forall(e \rightarrow \widehat{e})$ and
   if $\mathcal{Q} \models \forall((\exists Y(e \wedge d')) \leftrightarrow c)$ then $\mathcal{Q} \models \forall((\exists Y(\widehat{e} \wedge d')) \leftrightarrow c)$.

Note that, in general, the formula $\widehat{e}$ is not a constraint and, therefore, the constraint matching procedure cannot return $H \leftarrow \widehat{e} \wedge d' \wedge B' \wedge R$. Thus, starting from the formula $\widehat{e}$, we need to construct a constraint e such that $FVars(e) = FVars(\widehat{e})$ and $\mathcal{Q} \models \forall(\exists Y(e \wedge d') \leftrightarrow \exists Y(\widehat{e} \wedge d'))$ where $Y = FVars(\widehat{e} \wedge d') - Vars(H, B' \wedge R)$. Note also that it is always possible to rewrite the formula $\widehat{e}$ into a finite disjunction $e_1 \vee \ldots \vee e_m$ of constraints such that $\mathcal{Q} \models \forall(\widehat{e} \leftrightarrow (e_1 \vee \ldots \vee e_m))$ and $FVars(\widehat{e}) = Vars(e_1 \vee \ldots \vee e_m)$. Thus, we have the following property.

Lemma 2. *Let $\widehat{e}$ be the formula introduced in Definition 2 and let $e_1, \ldots, e_m$ be constraints such that $\mathcal{Q} \models \forall(\widehat{e} \leftrightarrow (e_1 \vee \ldots \vee e_m))$. Let Y be $FVars(\widehat{e} \wedge d') - Vars(H, B' \wedge R)$. Then for $i = 1, \ldots, m$, we have that $\mathcal{Q} \models \forall((\exists Y(e_i \wedge d')) \rightarrow c)$.*

Now we present the constraint matching procedure $\mathbf{CM_1}$ which is a nondeterministic procedure whose soundness follows from Proposition 1 and from Lemma 2. At Step 1 of the $\mathbf{CM_1}$ procedure the formula $\widehat{e}$ is replaced by an equivalent finite disjunction $e_1 \vee \ldots \vee e_n$ of constraints. At Step 2, the procedure tests whether or not there exists a constraint e_i such that $\mathcal{Q} \models \forall(c \rightarrow \exists Y\,(e_i \wedge d'))$ and, thus, by Lemma 2, it can be taken to be the constraint e we want to construct. Note that this technique for constructing e is very simple but, unfortunately, it leads to the incompleteness of the constraint matching procedure $\mathbf{CM_1}$ because, in general, there may be a constraint e such that $\mathcal{Q} \models \forall(c \leftrightarrow \exists Y\,(e \wedge d'))$, and yet, it does not exists e_i in $\{e_1, \ldots, e_m\}$ such that $\mathcal{Q} \models \forall(c \rightarrow \exists Y\,(e_i \wedge d'))$.

Constraint Matching Procedure: $\mathbf{CM_1}$

Input: two clauses $\gamma' \colon H \leftarrow c \wedge B' \wedge R$ and $\delta' \colon K' \leftarrow d' \wedge B'$ in normal form. Suppose that they are an output of the goal matching procedure **GM**.

Output: either a clause $\gamma'' : H \leftarrow e \wedge d' \wedge B' \wedge R$ such that $\gamma' \cong \gamma''$ and $EVars(\delta') \cap Vars(e) = \emptyset$, or **fail**.

Step 1. Transform the formula $\widehat{e} \colon \forall Z\ (d' \rightarrow c)$, where $Z = Vars(c) \cap Vars(d')$, into a finite disjunction of constraints as follows:

1.1 Transform the formula into $\neg\exists Z \neg\ (d' \rightarrow c)$.

1.2 Eliminate the negation from the formula $\neg\ (d' \rightarrow c)$ by eliminating $\rightarrow$ in favor of $\vee$ and $\neg$, by pushing $\neg$ inward as much as possible, and by replacing $\neg(p_1 \geq p_2)$ by $p_2 > p_1$, $\neg(p_1 > p_2)$ by $p_2 \geq p_1$, and $\neg(p_1 = p_2)$ by $p_1 > p_2 \vee p_2 > p_1$. A formula $f_1 \vee \ldots \vee f_k$, where $f_1, \ldots, f_k$ are constraints, is obtained.

1.3 Distribute $\exists Z$ over $\vee$, eliminate the existential variables of the set Z of variables from each disjunct f_i using the function *project*, and obtain the formula $g_1 \vee \ldots \vee g_k$, where for $i = 1, \ldots, k$, the disjunct g_i is the constraint *project*$(f_i,\ Vars(f_i) - Z)$.

1.4 Eliminate the negation from the formula $\neg\ (g_1 \vee \ldots \vee g_k)$ by pushing $\neg$ inward, and obtain a formula $e_1 \vee \ldots \vee e_m$, where $e_1, \ldots, e_m$ are constraints.

Step 2. IF there exists e_i in $\{e_1, \ldots, e_m\}$ such that $\mathcal{Q} \models \forall(c \rightarrow \exists Y (e_i \wedge d'))$, where Y is the set $FVars(e_i \wedge d') - Vars(H, B' \wedge R)$ THEN return $\gamma'' \colon H \leftarrow e_i \wedge d' \wedge B' \wedge R$ ELSE return **fail**.

In the following theorem we prove that the constraint matching procedure $\mathbf{CM_1}$ terminates and returns a correct output in the sense that if $\mathbf{CM_1}$ constructs a constraint e then the clauses $\gamma' : H \leftarrow c \wedge B' \wedge R$ and $\gamma'' : H \leftarrow e \wedge d' \wedge B' \wedge R$ are equivalent.

Theorem 3 (Termination and Soundness of the Constraint Matching Procedure $\mathbf{CM_1}$). *Let the two clauses $\gamma' : H \leftarrow c \wedge B' \wedge R$ and $\delta' : K' \leftarrow d' \wedge B'$ in normal form be the input of the constraint matching procedure $\mathbf{CM_1}$. Suppose that γ' and δ' are an output of* **GM**. *Then: (i) $\mathbf{CM_1}$ terminates, and (ii) if $\mathbf{CM_1}$ returns the clause $\gamma'' : H \leftarrow e \wedge d' \wedge B' \wedge R$, then $\gamma' \cong \gamma''$ and $EVars(\delta') \cap Vars(e) = \emptyset$.*

Now we introduce the folding algorithm $\mathbf{FA_1}$, which makes use of the goal matching procedure **GM** and the constraint matching procedure $\mathbf{CM_1}$. In order to fold a clause γ using a clause δ, the folding algorithm $\mathbf{FA_1}$: (i) applies the goal matching procedure **GM** and computes a matching substitution ϑ and a goal R, and then (ii) it applies the constraint matching procedure $\mathbf{CM_1}$ and looks for a constraint e such that Conditions (1) and (2) of Definition 1 are satisfied. If $\mathbf{CM_1}$ fails to compute the desired constraint e, the folding algorithm executes again the **GM** procedure so that new, alternative outputs ϑ and R are generated, until the $\mathbf{CM_1}$ procedure computes the desired constraint e. The folding algorithm $\mathbf{FA_1}$ returns **fail** whenever **GM** fails to compute a substitution ϑ and a goal R, and whenever, for all ϑ and R computed by **GM**, the $\mathbf{CM_1}$ procedure fails to compute a constraint e such that Conditions (1) and (2) of Definition 1 are satisfied.

Folding Algorithm: $\mathbf{FA_1}$

Input: two clauses $\gamma : H \leftarrow c \wedge G$ and $\delta : K \leftarrow d \wedge B$ in normal form and without variables in common.

Output: either a clause η: $H \leftarrow e \wedge K\vartheta \wedge R$ such that η is the result of folding γ using δ according to Definition 1, or **fail**.

IF there exist two clauses of the form $\gamma' : H \leftarrow c \wedge B\vartheta \wedge R$ and $\delta' : K\vartheta \leftarrow d\vartheta \wedge B\vartheta$, which are the output of an execution of the **GM** procedure when the clauses γ and δ are given in input
AND there exists a clause γ'': $H \leftarrow e \wedge d\vartheta \wedge B\vartheta \wedge R$ which is the output of an execution of the $\mathbf{CM_1}$ procedure when the clauses γ' and δ' are given in input
THEN return the clause η: $H \leftarrow e \wedge K\vartheta \wedge R$
ELSE return **fail**.

By using Theorems 2 and 3 we can prove termination and soundness of the folding algorithm $\mathbf{FA_1}$.

Theorem 4 (Termination and Soundness of the Folding Algorithm $\mathbf{FA_1}$). *Let γ and δ two clauses in normal form and without variables in common, which are the input of Algorithm $\mathbf{FA_1}$. Then: (i) $\mathbf{FA_1}$ terminates, and (ii) if the output of $\mathbf{FA_1}$ is a clause η, then η can be derived by folding γ using δ according to Definition 1.*

Let us now illustrate the folding algorithm $\mathbf{FA_1}$ by considering the example presented in the Introduction. The clauses γ and δ are already in normal form and do not have variable in common. Let p be a predicate symbol of type `rat` $\times$ `tree`, s be of type `rat` $\times$ `rat` $\times$ `tree`, q be of type `tree` $\times$ `rat`, and r be of type `tree`. If we apply the goal matching procedure **GM** to the clauses γ and δ we get the substitution ϑ: $\{A/[\,], W/T\}$ and the goal R: $r(Y)$. Thus, we get the following clauses:

γ': $p(X,Y) \leftarrow X>1 \ \wedge\ X>T \ \wedge\ q([\,],T) \ \wedge\ r(Y)$

δ': $s(U,V,[\,]) \leftarrow U>1 \ \wedge\ V>0 \ \wedge\ U>T \ \wedge\ q([\,],T)$

Then these clauses are given in input to the constraint matching procedure $\mathbf{CM_1}$. Since in that procedure we have assumed that γ' is of the form: $H \leftarrow c \wedge B' \wedge R$ and δ' is of the form: $K' \leftarrow d' \wedge B'$, we have that:

c is $X>1 \wedge X>T$ and d' is $U>1 \wedge V>0 \wedge U>T$.

We also have that $\mathit{Vars}(c) \cap \mathit{Vars}(d') = \{T\}$ and thus, we have to perform Step 1 starting from the formula $\forall T(d' \rightarrow c)$. At the end of Step 1 we obtain the following new formula:

$(1 \geq U) \ \vee \ (0 \geq V) \ \vee \ (X>1 \wedge X \geq U)$

which is the disjunction of the following three constraints: $e_1 \equiv 1 \geq U$, $e_2 \equiv 0 \geq V$, and $e_3 \equiv X>1 \wedge X \geq U$. Then, at Step 2 we get that:

(i) $\mathcal{Q} \not\models \forall(c \rightarrow \exists U \exists V\, (1 \geq U \wedge d'))$,
(ii) $\mathcal{Q} \not\models \forall(c \rightarrow \exists U \exists V\, (0 \geq V \wedge d'))$, and
(iii) $\mathcal{Q} \models \forall(c \rightarrow \exists U \exists V\, (X>1 \wedge X \geq U \wedge d'))$.

Thus, the constraint matching procedure $\mathbf{CM_1}$ returns the clause:

γ'': $p(X,Y) \leftarrow X>1 \ \wedge \ X \geq U \ \wedge \ d' \ \wedge \ q(T) \ \wedge \ r(Y)$

which after folding, that is, after the replacement of $d' \wedge q(T)$ by $s(U,V)$, becomes, as indicated in the Introduction:

η: $p(X,Y) \leftarrow X>1 \ \wedge \ X \geq U \ \wedge \ s(U,V) \ \wedge \ r(Y)$.

Due to the the fact that in general the constraint matching procedure $\mathbf{CM_1}$ is not complete, the folding algorithm $\mathbf{FA_1}$ is not complete either. Indeed, for some input clauses γ and δ there exists a triple $\langle e, \vartheta, R\rangle$ such that Conditions (1) and (2) of Definition 1 are satisfied, and yet $\mathbf{FA_1}$ returns **fail**.

In the next two sections, we will present two more folding algorithms for applying the standard folding rule as introduced in Definition 1, and we will provide the conditions under which they are complete. In particular, the first algorithm can be used when there is the extra requirement that no existential variable should be present in the folded clause, while the second algorithm can be used when, before applying the folding rule, preliminary applications of the clause splitting rule are possible.

4 A Folding Rule Combined with Clause Splitting

In this section we present a rule, called $\mathbf{Fold_2}$, which is an extension of the folding rule $\mathbf{Fold_1}$ presented in Section 3. An application of the rule $\mathbf{Fold_2}$ is equivalent to zero or more applications of the clause splitting rule [6] followed by some applications of the folding rule $\mathbf{Fold_1}$.

Let us motivate the introduction of the rule $\mathbf{Fold_2}$ by means of an example. Let us consider the following two clauses:

γ: $p(X) \leftarrow 2 \geq X \wedge X \geq 1 \wedge Z \geq 1 \wedge 2 \geq Z \wedge q(Z)$
δ: $s(Y) \leftarrow W + \frac{3}{4} \geq Y \wedge Y \geq W - \frac{3}{4} \wedge 4 - W \geq Y \wedge Y \geq 2 - W \wedge q(W)$

where the predicate symbols p, q, and s are of type `rat`. It is not possible to apply the rule $\mathbf{Fold_1}$ and fold clause γ using δ, because no constrained subgoal of clause γ is an instance of the body of clause δ. In order to allow folding, we can use the clause splitting transformation rule. In particular, given the property $\mathcal{Q} \models \forall Z\,(Z \geq \frac{5}{4} \vee \frac{7}{4} \geq Z)$, we can split clause γ into two clauses:

γ_1: $p(X) \leftarrow 2 \geq X \wedge X \geq 1 \wedge Z \geq 1 \wedge 2 \geq Z \wedge Z \geq \frac{5}{4} \wedge q(Z)$
γ_2: $p(X) \leftarrow 2 \geq X \wedge X \geq 1 \wedge Z \geq 1 \wedge 2 \geq Z \wedge \frac{7}{4} \geq Z \wedge q(Z)$

Now it is possible to fold both clause γ_1 and clause γ_2 using δ.

In order to fold γ_1 using δ (by applying the rule $\mathbf{Fold_1}$), we introduce: (i) the constraint e_1: $2 \geq X \wedge X \geq 1 \wedge 1 \geq Y$, (ii) the substitution ϑ: $\{W/Z\}$, and (iii) the empty conjunction R. In order to fold γ_2 using δ we introduce: (i) the constraint e_2: $2 \geq X \wedge X \geq 1 \wedge Y \geq 2$, (ii) the substitution ϑ: $\{W/Z\}$, and (iii) the empty conjunction R. The clauses η_1 and η_2 derived by folding γ_1 and γ_2 using δ are:

η_1: $p(X) \leftarrow 2 \geq X \wedge X \geq 1 \wedge 1 \geq Y \wedge s(Y)$
η_2: $p(X) \leftarrow 2 \geq X \wedge X \geq 1 \wedge Y \geq 2 \wedge s(Y)$

Note that in general it may be difficult to find a suitable formula (such as $\forall Z\,(Z \geq \frac{5}{4} \vee \frac{7}{4} \geq Z)$ in our example above) which after clause splitting will allow folding.

Definition 3 below introduces the folding rule $\mathbf{Fold_2}$ by which a clause $\gamma : H \leftarrow c \wedge G$ can be folded using a clause $\delta : K \leftarrow d \wedge B$ if: (i) there exist n constraints $c_1, \ldots, c_n$ such that $\mathcal{Q} \models \forall(c \leftrightarrow c_1 \vee \ldots \vee c_n)$ holds, and (ii) for $i = 1, \ldots, n$, by applying the folding rule $\mathbf{Fold_1}$, the clause $H \leftarrow c_i \wedge G$ can be folded using δ, thereby deriving $H \leftarrow e_i \wedge K\vartheta \wedge R$. Note that in Definition 3 the constraints $c_1, \ldots, c_n$ are not mentioned, and we will only require to find the clauses $\eta_1, \ldots, \eta_n$, or equivalently, the constraints $e_1, \ldots, e_n$, the substitution ϑ, and the goal R. Indeed, in the algorithm $\mathbf{FA_2}$ for applying the rule $\mathbf{Fold_2}$ we will introduce below, we directly compute, if they exist, the clauses $\eta_1, \ldots, \eta_n$, without finding the constraints $c_1, \ldots, c_n$.

Definition 3 (Rule $\mathbf{Fold_2}$). Let γ and δ be clauses of the form

γ: $H \leftarrow c \wedge G$
δ: $K \leftarrow d \wedge B$

such that γ and δ are in normal form and without variables in common. Suppose that there exist some constraints $e_1, \ldots, e_n$, a substitution ϑ, and a goal R such that:
(1) $\{\gamma\} \;\cong\; \{H \leftarrow e_1 \wedge d\vartheta \wedge B\vartheta \wedge R, \ldots, H \leftarrow e_n \wedge d\vartheta \wedge B\vartheta \wedge R\}$;
(2) for every variable X in $EVars(\delta)$, the following conditions hold: (i) $X\vartheta$ is a variable not occurring in $\{H, e_1, \ldots, e_n, R\}$, and (ii) $X\vartheta$ does not occur in the term $Y\vartheta$, for every variable Y occurring in $d \wedge B$ and different from X.
By *folding clause γ using clause δ* we derive the clauses η_1: $H \leftarrow e_1 \wedge K\vartheta \wedge R, \ldots,$ η_n: $H \leftarrow e_n \wedge K\vartheta \wedge R$.

Now we provide the algorithm $\mathbf{FA_2}$ for applying rule $\mathbf{Fold_2}$. Given two input clauses γ and δ, the output of the algorithm is: (i) a set of clauses derived by folding γ using δ as specified in Definition 3, if it is possible to fold, and (ii) **fail**, otherwise. The $\mathbf{FA_2}$ algorithm looks for suitable constraints $e_1, \ldots, e_n$, substitution ϑ, and goal R such that Conditions (1) and (2) of Definition 3 hold.

Similarly to Algorithm $\mathbf{FA_1}$ presented in Section 3, Algorithm $\mathbf{FA_2}$ makes use of two procedures: the goal matching procedure **GM**, which is the one used in $\mathbf{FA_1}$, and the constraint matching procedure $\mathbf{CM_2}$, which is the following variant of $\mathbf{CM_1}$.

Constraint Matching Procedure: $\mathbf{CM_2}$

Input: two clauses $\gamma'\colon H \leftarrow c \wedge B' \wedge R$ and $\delta'\colon K' \leftarrow d' \wedge B'$ in normal form. Suppose that they are an output of the goal matching procedure **GM**.

Output: (A) a set $\{\gamma''_1, \ldots, \gamma''_n\}$ of clauses, with $n \geq 0$, such that: (i) for $i = 1, \ldots, n$, γ''_i is of the form $H \leftarrow e_i \wedge d' \wedge B' \wedge R$, (ii) $\{\gamma'\} \cong \{\gamma''_1, \ldots, \gamma''_n\}$, and (iii) $\mathit{EVars}(\delta') \cap \mathit{Vars}(e_1, \ldots, e_n) = \emptyset$, if such set exists, and (B) **fail**, otherwise.

Step 1. Starting from $\widehat{e} = \forall Z(d' \rightarrow c)$ compute the disjunction $h_1 \vee \ldots \vee h_m$ of constraints in the same way as it has been done at Step 1 of the procedure $\mathbf{CM_1}$ for computing $e_1 \vee \ldots \vee e_m$ starting from $\widehat{e} = \forall Z(d' \rightarrow c)$.

Step 2. Let $\{e_1, \ldots, e_n\}$, with $n \leq m$, be the set of those h_i, with $1 \leq i \leq m$, such that $\mathcal{Q} \models \exists(h_i \wedge d')$. IF $\mathcal{Q} \models \forall(c \rightarrow \exists Y\, ((e_1 \vee \ldots \vee e_n) \wedge d'))$, where $Y = \mathit{FVars}((e_1 \vee \ldots \vee e_n) \wedge d') - \mathit{Vars}(H, B \wedge R)$ THEN return the set $\{H \leftarrow e_i \wedge d' \wedge B' \wedge R \mid i = 1, \ldots, n\}$ of clauses ELSE return **fail**.

It can be shown that the constraint matching procedure $\mathbf{CM_2}$ always terminates and it is sound and complete with respect to its specification. Thus, we slightly modify the folding algorithm $\mathbf{FA_1}$ by using $\mathbf{CM_2}$, instead of $\mathbf{CM_1}$, and we get a terminating, sound, complete folding algorithm $\mathbf{FA_2}$ to compute the result of applying rule $\mathbf{Fold_2}$ to clause γ using clause δ.

Folding Algorithm: $\mathbf{FA_2}$

Input: two clauses $\gamma\colon H \leftarrow c \wedge G$ and $\delta\colon K \leftarrow d \wedge B$ in normal form and with no variables in common.

Output: n clauses $\eta_1\colon H \leftarrow e_1 \wedge K\vartheta \wedge R,\ \ldots,\ \eta_n\colon H \leftarrow e_n \wedge K\vartheta \wedge R$, with $n \geq 0$, if it is possible to fold γ using δ according to Definition 3, and **fail**, otherwise.

IF c is unsatisfiable, that is, $\mathcal{Q} \models \neg\exists(c)$ THEN return the empty set of clauses
ELSE IF there exist two clauses of the form $\gamma' : H \leftarrow c \wedge B\vartheta \wedge R$ and $\delta' : K\vartheta \leftarrow d\vartheta \wedge B\vartheta$, which are the output of an execution of the **GM** procedure when given in input clauses γ and δ
AND there exists a set $\{H \leftarrow e_1 \wedge d\vartheta \wedge B\vartheta \wedge R,\ \ldots,\ H \leftarrow e_n \wedge d\vartheta \wedge B\vartheta \wedge R\}$ of clauses which is the output of an execution of the $\mathbf{CM_2}$ procedure when given in input clauses γ' and δ'
THEN return the clauses $\eta_1\colon H \leftarrow e_1 \wedge K\vartheta \wedge R,\ \ldots,\ \eta_n\colon H \leftarrow e_n \wedge K\vartheta \wedge R$
ELSE return **fail**.

We leave it to the reader to check that the clauses η_1 and η_2 considered in the example at the beginning of this section can be computed by applying the folding algorithm $\mathbf{FA_2}$ with clauses γ and δ as input.

Theorem 5 (Termination, Soundness, and Completeness of the Folding Algorithm $\mathbf{FA_2}$). *Let two clauses γ and δ, in normal form and with no variables in common, be the input of Algorithm* $\mathbf{FA_2}$. *Then:*
(i) $\mathbf{FA_2}$ *terminates,*
(ii) if $\mathbf{FA_2}$ *returns the clauses $\eta_1, \ldots, \eta_n$, then $\eta_1, \ldots, \eta_n$ can be derived by folding γ using δ according to Definition 3, and*
(iii) if it is possible to fold γ using δ according to Definition 3, then $\mathbf{FA_2}$ *does not return* **fail**.

5 A Folding Rule for the Elimination of Existential Variables

In this section we introduce a folding rule, called $\mathbf{Fold_3}$, which is a variant of the rule $\mathbf{Fold_1}$. When we apply the rule $\mathbf{Fold_3}$ to the clauses $\gamma\colon H \leftarrow c \wedge G$ and $\delta\colon K \leftarrow d \wedge B$, we replace a subgoal of $c \wedge G$, where some existential variables may occur, by an atom $K\vartheta$ thereby getting a new clause η in which $K\vartheta$ has no existential variables.

Definition 4 (Rule $\mathbf{Fold_3}$). Let γ and δ be clauses of the form

$\gamma\colon\ H \leftarrow c \wedge G$
$\delta\colon\ K \leftarrow d \wedge B$

such that γ and δ are in normal form and without variables in common. Suppose that there exist a constraint e, a substitution ϑ, and goal R such that:
(1) $\gamma\ \cong\ H \leftarrow e \wedge d\vartheta \wedge B\vartheta \wedge R$;
(2) for every variable X in $EVars(\delta)$, the following conditions hold: (i) $X\vartheta$ is a variable not occurring in $\{H, e, R\}$, and (ii) $X\vartheta$ does not occur in the term $Y\vartheta$, for every variable Y occurring in $d \wedge B$ and different from X;
(3) $Vars(K\vartheta) \subseteq Vars(H)$.
By *folding clause γ using clause δ* we derive the clause $\eta\colon H \leftarrow e \wedge K\vartheta \wedge R$.

Condition (3) ensures that no existential variable in the body of η occurs in $K\vartheta$. However, it may happen that in the folded clause η there are still some existential variables occurring in e or R, which could be eliminated by further folding steps using clause δ again or using other clauses.

Now we will present an algorithm, called $\mathbf{FA_3}$, for applying rule $\mathbf{Fold_3}$. The $\mathbf{FA_3}$ algorithm is a variant of the $\mathbf{FA_1}$ algorithm for applying rule $\mathbf{Fold_1}$ presented in Section 3. Given two clauses γ and δ, the $\mathbf{FA_3}$ algorithm returns a clause obtained by folding, if folding is possible, and it returns **fail**, otherwise. The $\mathbf{FA_3}$ algorithm makes use of: (i) the goal matching procedure **GM** presented in Section 3 with the following additional rewriting rule:

(vi) $\{X = t\} \cup \overline{S} \Longrightarrow$ **fail** if $X \in Vars_{\mathtt{tree}}(K)$ and $Vars(t) \not\subseteq Vars_{\mathtt{tree}}(H)$

and (ii) a constraint matching procedure, called $\mathbf{CM_3}$, we will present below.

The rewrite rule (vi) ensures that $Vars_{\mathtt{tree}}(K\vartheta) \subseteq Vars_{\mathtt{tree}}(H)$ and this fact will be used in the proof of Theorem 7 below for showing that Point (3) of Definition 4 holds and, thus, existential variables are eliminated.

The $\mathbf{CM_3}$ procedure takes as input the two clauses $\gamma' : H \leftarrow c \wedge B' \wedge R$ and $\delta' : K' \leftarrow d' \wedge B'$, which are the output of the goal matching procedure. If there exist a constraint e and a substitution ϑ_2 such that: (i) $\gamma' \cong H \leftarrow e \wedge d'\vartheta_2 \wedge B' \wedge R$, (ii) $B'\vartheta_2 = B'$, (iii) $\mathit{Vars}(K'\vartheta_2) \subseteq \mathit{Vars}(H)$, and (iv) $\mathit{Vars}(e) \subseteq \mathit{Vars}(H, R)$, then the constraint matching procedure $\mathbf{CM_3}$ returns a clause $\gamma'' : H \leftarrow e \wedge d'\vartheta_2 \wedge B'\vartheta_2 \wedge R$, else it returns **fail**.

Let $\widetilde{e}$ denote the constraint $\mathit{project}(c, X)$, where $X = \mathit{Vars}(c) - \mathit{Vars}(B')$ (see Section 2 for the definition of the *project* function). Lemma 3 below shows that, for any substitution ϑ_2, if there exists a constraint e satisfying Conditions (i)–(iv) above, then we can always take e to be the constraint $\widetilde{e}$. Thus, by Lemma 3 the procedure $\mathbf{CM_3}$ should only search for a substitution ϑ_2 such that $\mathcal{Q} \models \forall(c \leftrightarrow (\widetilde{e} \wedge d'\vartheta_2))$.

Lemma 3. *Let $\gamma' : H \leftarrow c \wedge B' \wedge R$ and $\delta' : K' \leftarrow d' \wedge B'$ be the input clauses of the constraint matching procedure. For every substitution ϑ_2, there exists a constraint e such that:* (i) $\gamma' \cong H \leftarrow e \wedge d'\vartheta_2 \wedge B' \wedge R$, (ii) $B'\vartheta_2 = B'$, (iii) $\mathit{Vars}(K'\vartheta_2) \subseteq \mathit{Vars}(H)$, *and* (iv) $\mathit{Vars}(e) \subseteq \mathit{Vars}(\{H, R\})$ *iff* $\mathcal{Q} \models \forall(c \leftrightarrow (\widetilde{e} \wedge d'\vartheta_2))$ *and Conditions* (ii) *and* (iii) *hold.*

Now we introduce some notions and we state some properties (see Lemma 4 and Theorem 6) which will be exploited by the constraint matching procedure $\mathbf{CM_3}$ for reducing the equivalence between c and $\widetilde{e} \wedge d'\vartheta_2$, for a suitable ϑ_2, to a set of equivalences between the atomic constraints occurring in c and $\widetilde{e} \wedge d'\vartheta_2$.

A conjunction $a_1 \wedge \ldots \wedge a_m$ of (not necessarily distinct) atomic constraints is said to be *redundant* if there exists i, with $0 \leq i \leq m$, such that $\mathcal{Q} \models \forall((a_1 \wedge \ldots \wedge a_{i-1} \wedge a_{i+1} \wedge \ldots \wedge a_m) \rightarrow a_i)$. In this case we also say that a_i is redundant in $a_1 \wedge \ldots \wedge a_m$. Thus, the empty conjunction *true* is non-redundant and an atomic constraint a is redundant iff $\mathcal{Q} \models \forall(a)$. Given a redundant constraint c, we can always derive a non-redundant constraint c' which is equivalent to c, that is, $\mathcal{Q} \models \forall(c \leftrightarrow c')$, by repeatedly eliminating from the constraint at hand an atomic constraint which is redundant in that constraint.

Without loss of generality we can assume that any given constraint c is of the form $p_1\, \rho_1\, 0 \wedge \ldots \wedge p_m\, \rho_m\, 0$, where $m \geq 0$ and $\rho_1, \ldots, \rho_m \in \{\geq, >\}$. We define the *interior* of c, denoted $\mathit{interior}(c)$, to be the constraint $p_1 > 0 \wedge \ldots \wedge p_m > 0$. A constraint c is said to be *admissible* if both c and $\mathit{interior}(c)$ are satisfiable and non-redundant. For instance, the constraint $c_1 : X - Y \geq 0 \wedge Y \geq 0$ is admissible, while the constraint $c_2 : X - Y \geq 0 \wedge Y \geq 0 \wedge X > 0$ is not admissible (indeed, c_2 is non-redundant and $\mathit{interior}(c_2) : X - Y > 0 \wedge Y > 0 \wedge X > 0$ is redundant). The following Lemma 4 characterizes the equivalence of two constraints when one of them is admissible.

Lemma 4. *Let us consider an admissible constraint a of the form $a_1 \wedge \ldots \wedge a_m$ and a constraint b of the form $b_1 \wedge \ldots \wedge b_n$, where $a_1, \ldots, a_m, b_1, \ldots, b_n$ are atomic constraints* (*in particular, they are not equalities*)*. We have that $\mathcal{Q} \models \forall\,(a \leftrightarrow b)$ holds iff there exists an injection $\mu : \{1, \ldots, m\} \rightarrow \{1, \ldots, n\}$ such that for $i = 1, \ldots, m$, $\mathcal{Q} \models \forall\,(a_i \leftrightarrow b_{\mu(i)})$ and for $j = 1, \ldots, n$, if $j \notin \{\mu(i) \mid 1 \leq i \leq m\}$, then $\mathcal{Q} \models \forall\,(a \rightarrow b_j)$.*

In order to see that admissibility is a needed hypothesis for Lemma 4, let us consider the non-admissible constraint $c_3 : X - Y \geq 0 \wedge Y \geq 0 \wedge X + Y > 0$. We have that $\mathcal{Q} \models \forall(c_2 \leftrightarrow c_3)$ and yet there is no injection which has the properties stated in Lemma 4.

Lemma 4 will be used to show that if there exists a substitution ϑ_2 such that $\mathcal{Q} \models \forall(c \leftrightarrow (\widetilde{e} \wedge d'\vartheta_2))$, where c is an admissible constraint and $\widetilde{e}$ is defined as in Lemma 3, then $\mathbf{CM_3}$ computes such a substitution ϑ_2. Indeed, given the constraint c, of the form $a_1 \wedge \ldots \wedge a_m$, and the constraint $\widetilde{e} \wedge d'$, of the form $b_1 \wedge \ldots \wedge b_n$, $\mathbf{CM_3}$ computes: (1) an injection μ from $\{1, \ldots, m\}$ to $\{1, \ldots, n\}$, and (2) a substitution ϑ_2 such that: (2.i) for $i = 1, \ldots, m$, $\mathcal{Q} \models \forall(a_i \leftrightarrow b_{\mu(i)}\vartheta_2)$, and (2.ii) for $j = 1, \ldots, n$, if $j \notin \{\mu(i) \mid 1 \leq i \leq m\}$, then $\mathcal{Q} \models \forall(c \rightarrow b_j\vartheta_2)$.

In order to compute ϑ_2 satisfying the property of Point (2.i), we make use of the following Property $P1$: given the satisfiable, non-redundant constraints $p > 0$ and $q > 0$, we have that $\mathcal{Q} \models \forall(p > 0 \leftrightarrow q > 0)$ holds iff there exists a rational number $k > 0$ such that $\mathcal{Q} \models \forall(kp - q = 0)$ holds. Property $P1$ holds also if we replace $p > 0$ and $q > 0$ by $p \geq 0$ and $q \geq 0$, respectively.

Finally, in order to compute ϑ_2 satisfying the property of Point (2.ii), we make use of the following Theorem 6 which is a generalization of the above Property $P1$ and it is an extension of Farkas' Lemma to the case of systems of weak and strict inequalities [17].

Theorem 6. *Suppose that* $p_1\, \rho_1\, 0, \ldots, p_m\, \rho_m\, 0, p_{m+1}\, \rho_{m+1}\, 0$ *are atomic constraints such that, for* $i = 1, \ldots, m+1$*,* $\rho_i \in \{\geq, >\}$ *and* $\mathcal{Q} \models \exists(p_1\, \rho_1\, 0 \wedge \ldots \wedge p_m\, \rho_m\, 0)$*. Then* $\mathcal{Q} \models \forall(p_1\, \rho_1\, 0 \wedge \ldots \wedge p_m\, \rho_m\, 0 \rightarrow p_{m+1}\, \rho_{m+1}\, 0)$ *iff there exist* $k_1 \geq 0, \ldots, k_{m+1} \geq 0$ *such that: (i)* $\mathcal{Q} \models \forall\,(k_1 p_1 + \cdots + k_m p_m + k_{m+1} = p_{m+1})$*, and (ii) if* ρ_{m+1} *is* $>$ *then* $(\sum_{i \in I} k_i) > 0$*, where* $I = \{i \mid 1 \leq i \leq m+1,\ \rho_i \text{ is } >\}$*.*

As we will see below, the constraint matching procedure $\mathbf{CM_3}$ may generate *bilinear* polynomials (see rules (i)–(iii)), that is, non-linear polynomials of a particular form, which we now define. Let p be a polynomial and $\langle P_1, P_2 \rangle$ be a partition of a (proper or not) superset of $\mathit{Vars}(p)$. The polynomial p is said to be *bilinear in the partition* $\langle P_1, P_2 \rangle$ if the monomials of p are of the form: *either* (i) $k\,XY$, where k is a rational number, $X \in P_1$, and $Y \in P_2$, *or* (ii) $k\,X$, where k is a rational number and X is a variable, *or* (iii) k, where k is a rational number. Let us consider a polynomial p which is bilinear in the partition $\langle P_1, P_2 \rangle$ where $P_2 = \{Y_1, \ldots, Y_m\}$. The *normal form* of p, denoted $\mathit{nf}(p)$, *w.r.t. a given ordering* $Y_1, \ldots, Y_m$ *of the variables in* P_2, is a bilinear polynomial which is derived by: (i) computing the bilinear polynomial $p_1 Y_1 + \cdots + p_m Y_m + p_{m+1}$ such that $\mathcal{Q} \models \forall\,(p_1 Y_1 + \cdots + p_m Y_m + p_{m+1} = p)$, and (ii) erasing from that bilinear polynomial every summand $p_i Y_i$ such that $\mathcal{Q} \models \forall\,(p_i = 0)$.

Constraint Matching Procedure: $\mathbf{CM_3}$

Input: two clauses $\gamma' : H \leftarrow c \wedge B' \wedge R$ and $\delta' : K' \leftarrow d' \wedge B'$ in normal form.

Output: a clause $\gamma'' : H \leftarrow e \wedge d'\vartheta_2 \wedge B'\vartheta_2 \wedge R$ such that: (i) $\gamma' \cong H \leftarrow e \wedge d'\vartheta_2 \wedge B' \wedge R$, (ii) $B'\vartheta_2 = B'$, (iii) $\mathit{Vars}(K'\vartheta_2) \subseteq \mathit{Vars}(H)$, and (iv) $\mathit{Vars}(e) \subseteq \mathit{Vars}(H, R)$. If such clause γ'' does not exist, then **fail**.

Let X be the set $\mathit{Vars}(c) - \mathit{Vars}(B')$, Y be the set $\mathit{Vars}(d') - \mathit{Vars}(B')$, and Z be the set $\mathit{Vars}_{\texttt{rat}}(B')$. Let e be the constraint $\mathit{project}(c, X)$. Without loss of generality, we may assume that: (i) c is a constraint of the form $p_1\, \rho_1\, 0 \wedge \ldots \wedge p_m\, \rho_m\, 0$, where for $i = 1, \ldots, m$, p_i is a linear polynomial and $\rho_i \in \{\geq, >\}$, and (ii) $e \wedge d'$ is a constraint of the form $q_1\, \pi_1\, 0 \wedge \ldots \wedge q_n\, \pi_n\, 0$, where for $j = 1, \ldots, n$, q_i is a linear polynomial and $\pi_i \in \{\geq, >\}$.

Let us consider the following rewrite rules (i)–(v) which are all of the form:

$$\langle f_1 \leftrightarrow g_1,\ S_1,\ \sigma_1\rangle \Longrightarrow \langle f_2 \leftrightarrow g_2,\ S_2,\ \sigma_2\rangle$$

where: (1) f_1, g_1, f_2, and g_2 are constraints, (2) S_1 and S_2 are sets of constraints, and (3) σ_1 and σ_2 are substitutions. In the rewrite rules (i)–(v) below, whenever S_1 is written as $A \cup B$, we assume that $A \cap B = \emptyset$.

(i) $\langle p\, \rho\, 0 \wedge f \leftrightarrow g_1 \wedge q\, \rho\, 0 \wedge g_2,\ S,\ \sigma\rangle \Longrightarrow$
$\quad\langle f \leftrightarrow g_1 \wedge g_2,\ \{\mathit{nf}(Vp - q) = 0, V > 0\} \cup S,\ \sigma\rangle$
where V is a new variable and $\rho \in \{\geq, >\}$;

(ii) $\langle \mathit{true} \leftrightarrow q \geq 0 \wedge g,\ S,\ \sigma\rangle \Longrightarrow$
$\quad\langle \mathit{true} \leftrightarrow g,\ \{\mathit{nf}(V_1 p_1 + \ldots + V_m p_m + V_{m+1} - q) = 0,$
$\quad\quad V_1 \geq 0, \ldots, V_{m+1} \geq 0\} \cup S,\ \sigma\rangle$
where $V_1, \ldots, V_{m+1}$ are new variables;

(iii) $\langle \mathit{true} \leftrightarrow q > 0 \wedge g,\ S,\ \sigma\rangle \Longrightarrow$
$\quad\langle \mathit{true} \leftrightarrow g,\ \{\mathit{nf}(V_1 p_1 + \ldots + V_m p_m + V_{m+1} - q) = 0,$
$\quad\quad V_1 \geq 0, \ldots, V_{m+1} \geq 0,\ (\sum_{i \in I} V_i) > 0\} \cup S,\ \sigma\rangle$
where $V_1, \ldots, V_{m+1}$ are new variables and $I = \{i \mid 1 \leq i \leq m+1,\ \rho_i \text{ is } >\}$;

(iv) $\langle f \leftrightarrow g,\ \{p\, U + q = 0\} \cup S,\ \sigma\rangle \Longrightarrow \langle f \leftrightarrow g,\ \{p = 0, q = 0\} \cup S,\ \sigma\rangle$
if $U \in X \cup Z$;

(v) $\langle f \leftrightarrow g,\ \{a\, U + q = 0\} \cup S,\ \sigma\rangle \Longrightarrow$
$\quad\langle f \leftrightarrow (g\{U/-\frac{q}{a}\}),\ \{\mathit{nf}(p\{U/-\frac{q}{a}\})\rho 0 \mid p\, \rho\, 0 \in S\},\ \sigma\{U/-\frac{q}{a}\}\rangle$
if $U \in Y$, $\mathit{Vars}(q) \cap \mathit{Vars}(R) = \emptyset$, and $a \in (\mathbb{Q} - \{0\})$;

IF $\mathcal{Q} \models \neg\exists(e)$ THEN return clause $\gamma''\colon H \leftarrow e \wedge d'\vartheta_2 \wedge B'\vartheta_2 \wedge R$, where ϑ_2 is an arbitrary substitution of the form $\{U_1/a_1, \ldots, U_s/a_s\}$, with $\{U_1, \ldots, U_s\} = \mathit{Vars}_{\texttt{rat}}(K') - \mathit{Vars}(H)$, and $a_1, \ldots, a_s \in \mathbb{Q}$;

ELSE IF there exists a set C of constraints and a substitution σ_Y such that:

1. $\langle c \leftrightarrow e \wedge d',\ \emptyset,\ \emptyset\rangle \Longrightarrow^* \langle \mathit{true} \leftrightarrow \mathit{true},\ C,\ \sigma_Y\rangle$,
2. there is no triple T such that $\langle \mathit{true} \leftrightarrow \mathit{true},\ C,\ \sigma_Y\rangle \Longrightarrow T$,
3. for all $f \in C$, $\mathit{Vars}(f) \subseteq W$, where W is the set of the new variables introduced when applying rules (i)–(v) at Step 1,
4. C is satisfiable and $\mathit{solve}(C) = \sigma_W$,

THEN let σ_G be an arbitrary substitution of the form $\{U_1/a_1, \ldots, U_s/a_s\}$, where $\{U_1, \ldots, U_s\} = \mathit{Vars}_{\texttt{rat}}(K'\sigma_Y\sigma_W) - \mathit{Vars}(H)$ and $a_1, \ldots, a_s \in \mathbb{Q}$;
let ϑ_2 be $\sigma_Y\sigma_W\sigma_G$;
return clause $\gamma''\colon H \leftarrow e \wedge d'\vartheta_2 \wedge B'\vartheta_2 \wedge R$

ELSE return **fail**.

Note that in order to apply rules (iv) and (v), $p\,U$ and $a\,U$, respectively, should be the leftmost monomials. The procedure $\mathbf{CM_3}$ is nondeterministic (see rule (i)). By induction on the number of rule applications, we can show that the polynomials occurring in the second components of the triples are all bilinear in the partition $\langle W, X \cup Y \cup Z\rangle$, where W is the set of the new variables introduced during the application of the rewrite rules. The normal forms of the bilinear polynomials which occur in the rewrite rules are all computed w.r.t. the fixed variable ordering $Z_1, \ldots, Z_h,\ Y_1, \ldots, Y_k,\ X_1, \ldots, X_l$, where $\{Z_1, \ldots, Z_h\} = Z$, $\{Y_1, \ldots, Y_k\} = Y$, and $\{X_1, \ldots, X_l\} = X$.

It can be shown that the constraint matching procedure $\mathbf{CM_3}$ always terminates and it returns a clause γ'' which satisfies the *Output* conditions of Procedure $\mathbf{CM}_3$. Moreover, it can be shown that if the constraint c occurring in the input clause γ' (and γ) is either unsatisfiable or admissible then the constraint matching procedure $\mathbf{CM_3}$ does not return **fail**. Thus, by using $\mathbf{CM_3}$, instead of $\mathbf{CM_1}$, we get a terminating, sound, and (under the above mentioned condition on the constraint c) complete folding algorithm $\mathbf{FA_3}$ for applying the rule $\mathbf{Fold_3}$.

Folding Algorithm: $\mathbf{FA_3}$

Input: two clauses $\gamma : H \leftarrow c \wedge G$ and $\delta : K \leftarrow d \wedge B$ in normal form and without variables in common.

Output: a clause $\eta : H \leftarrow e \wedge K\vartheta \wedge R$, if it is possible to fold γ using δ according to Definition 4, and **fail**, otherwise.

```
IF there exist two clauses of the form $\gamma' : H \leftarrow c \wedge B\vartheta_1 \wedge R$ and $\delta' : K\vartheta_1 \leftarrow d\vartheta_1 \wedge B\vartheta_1$, which are the output of an execution of the GM procedure when the clauses $\gamma$ and $\delta$ are given in input
AND there exists a clause $\gamma'' : H \leftarrow e \wedge d\vartheta_1\vartheta_2 \wedge B\vartheta_1\vartheta_2 \wedge R$ which is the output of an execution of the CM_3 procedure when the clauses $\gamma'$ and $\delta'$ are given in input
THEN return the clause $\eta : H \leftarrow e \wedge K\vartheta_1\vartheta_2 \wedge R$
ELSE return fail.
```

Theorem 7 (Termination, Soundness, and Completeness of the Folding Algorithm $\mathbf{FA_3}$). *Let two clauses γ and δ, in normal form and without variables in common, be the input of Algorithm* $\mathbf{FA_3}$. *Then:*
(i) $\mathbf{FA_3}$ *terminates,*
(ii) if $\mathbf{FA_3}$ *returns a clause η, then η can be derived by folding γ using δ according to Definition 4, and*
(iii) if it is possible to fold γ using δ according to Definition 4 and the constraint occurring in γ is either unsatisfiable or admissible, then $\mathbf{FA_3}$ *does not return* **fail**.

Let us now present an example of application of the folding algorithm $\mathbf{FA_3}$. Suppose we are given the following two clauses:

γ: $p(X_1, X_2) \leftarrow Z > 0 \wedge X_1 - Z - 1 \geq 0 \wedge X_2 - X_1 > 0 \wedge s(Z, f(X_3))$
δ: $q(Y_1, Y_2) \leftarrow U > 0 \wedge Y_1 - 3 - 2U \geq 0 \wedge 2Y_2 - Y_1 - 1 > 0 \wedge s(U, f(Y_3))$

where p and q are predicate symbols of type `rat`$\times$`rat`, s of type `rat`$\times$`tree`, and we want to fold γ using δ. These clauses are in normal form and do not have variables in common. The output of the goal matching procedure applied to the clauses γ and δ is the following pair of clauses:

γ': $p(X_1,X_2) \leftarrow Z > 0 \wedge X_1 - Z{-}1 \geq 0 \wedge X_2 - X_1 > 0 \wedge s(Z, f(X_3))$
δ': $q(Y_1,Y_2) \leftarrow Z > 0 \wedge Y_1{-}3 - 2Z \geq 0 \wedge 2Y_2 - Y_1{-}1 > 0 \wedge s(Z, f(X_3))$

which are the input of the constraint matching procedure $\mathbf{CM_3}$. The $\mathbf{CM_3}$ procedure starts off by computing the following constraint e which is defined as $\mathit{project}(Z > 0 \wedge X_1 \geq Z{+}1 \wedge X_2 > X_1,\ \{X_1, X_2\})$:

e: $X_1{-}1 > 0 \wedge X_2{-}X_1 > 0$

Now, starting from the triple:

$\langle (Z > 0 \wedge X_1 - Z{-}1 \geq 0 \wedge X_2 - X_1 > 0) \leftrightarrow$
$(X_1{-}1 > 0 \wedge X_2{-}X_1 > 0 \wedge Z > 0 \wedge Y_1{-}3 - 2Z \geq 0 \wedge 2Y_2 - Y_1{-}1 > 0),\ \emptyset,\ \emptyset\rangle$

and applying the rewrite rules (i)–(v), we derive the following set of constraints:

C: $\{W_1 - 1 = 0,\ W_1 > 0,\ 2 - W_2 = 0,\ W_2 > 0,\ W_3 > 0\}$

together with the following substitution:

σ_Y: $\{Y_1/W_2X_1 + 3 - W_2,\ Y_2/(\frac{W_2}{2} - \frac{W_3}{2})X_1 + \frac{W_3}{2}X_2 + 2 - \frac{W_2}{2}\}$

The solution $\mathit{solve}(C)$ is the ground substitution $\sigma_W = \{W_1/1,\ W_2/2,\ W_3/2\}$. Since $\mathit{Vars}_{\mathtt{rat}}(q(Y_1,Y_2)\sigma_Y\sigma_W) \subseteq \mathit{Vars}(p(X_1,X_2))$, we have that σ_G is the identity substitution $\emptyset$. Thus, the substitution ϑ_2 is equal to $\sigma_Y\sigma_W$. Since we have to compute $q(Y_1,Y_2)\,\vartheta_2$, we can restrict ϑ_2 to the set $\{Y_1,Y_2\}$ and we get:

ϑ_2: $\{Y_1/2X_1 + 1,\ Y_2/X_2 + 1\}$

Thus, an output of the folding algorithm $\mathbf{FA_3}$ is the clause $p(X_1,X_2) \leftarrow e \wedge q(Y_1,Y_2)\vartheta_2$, that is:

η: $p(X_1,X_2) \leftarrow X_1 > 1 \wedge X_2 > X_1 \wedge q(2X_1{+}1, X_2{+}1)$

Clause η has no existential variables.

When applying the rewrite rule (i), the nondeterministic procedure $\mathbf{CM_3}$ may also compute a different substitution, which we call ϑ_2', by selecting a different atomic constraint $q\,\rho\,0$. In particular, the procedure $\mathbf{CM_3}$ may compute the following alternative set of constraints:

C': $\{W_1 + W_2 + W_3 > 0,\ W_1 \geq 0,\ W_2 \geq 0,\ W_3 \geq 0\}$

together with the substitution:

σ_Y': $\{Y_1/2X_1 + 1,\ Y_2/\frac{W_1+1}{2}X_1 + \frac{W_2}{2}X_2 + 1 + \frac{W_3}{2}\}$

The solution $\mathit{solve}(C')$ is the ground substitution $\sigma_W' = \{W_1/1,\ W_2/0,\ W_3/1\}$. Since $\mathit{Vars}_{\mathtt{rat}}(q(Y_1,Y_2)\sigma_Y'\sigma_W') \subseteq \mathit{Vars}(p(X_1,X_2))$, we have that σ_G' is the identity substitution $\emptyset$. Therefore, the substitution ϑ_2' is equal to $\sigma_Y'\sigma_W'$. Since we have to compute $q(Y_1,Y_2)\,\vartheta_2'$, we can restrict ϑ_2' to the set $\{Y_1,Y_2\}$ and we get:

ϑ_2': $\{Y_1/2X_1 + 1,\ Y_2/X_1 + \frac{3}{2}\}$

Thus, an alternative output of the folding algorithm $\mathbf{FA_3}$ is the clause:

η': $p(X_1,X_2) \leftarrow X_1 > 1 \wedge X_2 > X_1 \wedge q(2X_1{+}1, X_2{+}\frac{3}{2})$

Also this clause η' has no existential variables.

The reader can check that the $\mathbf{CM_3}$ procedure cannot generate other sets of constraints besides the two sets C and C' we have indicated above.

6 Related Work and Conclusions

The folding rule has been often considered in the papers that deal with the transformation rules for logic programs and constraints logic programs [3,5,7,10,18,19]. Folding is a crucial transformation rule because, besides other reasons, it allows beneficial changes of the recursive structure of the programs and, when using program transformation for inductive proofs [14], its application is basically equivalent to the use of the inductive hypothesis.

In the literature the folding rule is specified in a declarative way and no algorithm is provided to determine whether or not, given a clause γ to be folded and a clause δ for folding, one can actually fold γ using δ.

In this paper we have considered constraint logic programs with constraints that are conjunctions of linear equations and inequations over the rational numbers (or the real numbers) and we have proposed an algorithm, based on linear algebra and term rewriting techniques, for applying the folding rule.

We have also introduced two variants of the folding rule and we have presented two algorithms for applying these variants. The first variant combines the folding rule with the clause splitting rule and the second variant can be applied for eliminating the existential variables of a clause, that is, the variables which occur in the body of a clause and not in the head.

The problem of checking whether or not, given two clauses, say γ and δ, clause γ can be folded using clause δ, is similar to the problem of matching two terms modulo an equational theory [1,20], but the matching problem for deciding the applicability of folding has an extra difficulty that is due to the presence of existential variables (these variables are not taken into account in the equational theories considered in [1,20]).

In the future we plan to investigate in more detail the connections between the problem of folding and the problem of matching modulo an equational theory, by looking, in particular, at those techniques which deal with combinations of equational theories (see, for instance, [16]).

We also plan to adapt of our folding algorithms to other constraint domains, such as the linear equations and inequations over the integer numbers.

One more aspect that need to be addressed is the analysis of the computational complexity of our algorithms for folding. Since our algorithms make use of the Fourier-Motzkin variable elimination procedure, they take superexponential time in the number of variables occurring in the input clauses.

Finally, an implementation of our folding algorithms is under development in the MAP transformation system [11]. This implementation will allow us to evaluate in practice the efficiency of the folding algorithms, as well as the usefulness of the various versions of the folding rule in various program derivations.

References

1. F. Baader and W. Snyder. Unification theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, pages 445–532. Elsevier Science, 2001.

2. D. Benanav, D. Kapur, and P. Narendran. Complexity of matching problems. *Journal of Symbolic Computation*, 3(1-2):203–216, 1987.
3. N. Bensaou and I. Guessarian. Transforming constraint logic programs. *Theoretical Computer Science*, 206:81–125, 1998.
4. R. M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, January 1977.
5. S. Etalle and M. Gabbrielli. Transformations of CLP modules. *Theoretical Computer Science*, 166:101–146, 1996.
6. F. Fioravanti, A. Pettorossi, and M. Proietti. Specialization with clause splitting for deriving deterministic constraint logic programs. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Hammamet (Tunisia)*. IEEE Computer Society Press, 2002.
7. F. Fioravanti, A. Pettorossi, and M. Proietti. Transformation rules for locally stratified constraint logic programs. In K.-K. Lau and M. Bruynooghe, editors, *Program Development in Computational Logic*, Lecture Notes in Computer Science 3049, pages 292–340. Springer-Verlag, 2004.
8. J. Jaffar and M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.
9. J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1987. Second Edition.
10. M. J. Maher. A transformation system for deductive database modules with perfect model semantics. *Theoretical Computer Science*, 110:377–403, 1993.
11. MAP. The MAP transformation system. Available from `http://www.iasi.rm.cnr.it/~proietti/system.html`, 1995–2008.
12. E. D. Nering. *Linear Algebra and Matrix Theory.* John Wiley & Sons, 2nd edition, 1963.
13. A. Pettorossi and M. Proietti. Rules and strategies for transforming functional and logic programs. *ACM Computing Surveys*, 28(2):360–414, 1996.
14. A. Pettorossi, M. Proietti, and V. Senni. Proving properties of constraint logic programs by eliminating existential variables. In S. Etalle and M. Truszczynski, editors, *Proceedings of the 22nd International Conference on Logic Programming (ICLP '06)*, volume 4079 of *Lecture Notes in Computer Science*, pages 179–195. Springer-Verlag, 2006.
15. M. Proietti and A. Pettorossi. Unfolding-definition-folding, in this order, for avoiding unnecessary variables in logic programs. *Theoretical Computer Science*, 142(1):89–124, 1995.
16. C. Ringeissen. Matching in a class of combined non-disjoint theories. In F. Baader, editor, *Proceedings of the 19th International Conference on Automated Deduction (CADE-19), Miami Beach, FL, USA, July 28 - August 2, 2003*, volume 2741 of *Lecture Notes in Computer Science*, pages 212–227. Springer, 2003.
17. A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, 1986.
18. H. Seki. Unfold/fold transformation of stratified programs. *Theoretical Computer Science*, 86:107–139, 1991.
19. H. Tamaki and T. Sato. Unfold/fold transformation of logic programs. In S.-Å. Tärnlund, editor, *Proceedings of the Second International Conference on Logic Programming*, pages 127–138, Uppsala, Sweden, 1984. Uppsala University.
20. Terese. *Term Rewriting Systems.* Cambridge University Press, 2003.