PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT.

Output in CLP($\mathcal{R}$)

JOXAN JAFFAR*
PETER J. STUCKEY†
MICHAEL J. MAHER*
ROLAND H.C. YAP*

Abstract

An important issue in Constraint Logic Programming (CLP) systems is how to output constraints in a usable form. Typically, only a small subset $\bar{x}$ of the variables in constraints is of interest, and so an informal statement of the problem at hand is: given a conjunction $c(\bar{x}, \bar{y})$ of constraints, express $\exists \bar{y}\ c(\bar{x}, \bar{y})$ in the simplest form. In this paper, we consider the constraints of the CLP($\mathcal{R}$) system and describe the essential features of its output module. In the main, we focus on the well-known problem of projection in *linear* arithmetic constraints. We start with a classical algorithm and augment it with a procedure for eliminating redundant constraints generated by the algorithm. The rest of the paper discusses the remaining kinds of constraints, equations over trees and nonlinear equations, and clarifies how they are output together with linear constraints.

1 Introduction

In its simplest description, the output of a constraint logic programming (CLP) [Jaffar and Lassez 1986] program is the collection of all constraints accumulated along a successful execution path. However such a collection is, in general, extremely complex because it is very large and contains many intermediate variables of no intrinsic interest to the user. Therefore, we can informally state that the problem at hand is: given a set $\bar{x}$ of *target* variables and a conjunction $\mathcal{C}(\bar{x}, \bar{y})$ of constraints, express $\exists \bar{y}\ \mathcal{C}(\bar{x}, \bar{y})$ in the most usable form. While we cannot define usability formally, it typically means both conciseness and readability. In this paper we consider the constraints of the CLP($\mathcal{R}$) system [Jaffar *et al.* 1990] and discuss the several issues and techniques that arose in implementing the output module for CLP($\mathcal{R}$).

*IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA.
†Dept. of Computer Science, Univ. of Melbourne, Parkville, Victoria 3052, Australia.

Consider some examples. Where $\{x, y\}$ are the target variables: (a) the constraints $x = f(z, z),\ z = g(y, w)$ can be output as $x = f(g(y, \_1), g(y, \_1))$[1]; (b) the constraints $x = z + 1, y = 2 * z$ can be output as $x = 0.5 * y + 1$; (c) the constraints $x < z,\ z \leq y,\ z \leq y + 1$ can be output as $x < y$; (d) the constraints $x = \sin z * \sin z + \cos z * \cos z + y$ can be output as $x = 1 + y$.

We can classify the simplification of constraints in three directions:

(I) the elimination of auxiliary variables (as in (a), (b) and (c));
(II) the elimination of redundant constraints (as in (c)), and
(III) the replacement of expressions by simpler equivalent ones (as in (d)).

The problem (I) of expressing $\exists \tilde{y}\ \mathcal{C}(\tilde{x}, \tilde{y})$ as a formula involving only the variables $\tilde{x}$ is known variously as projection, variable elimination and quantifier elimination. Full variable elimination is not always possible in CLP($\mathcal{R}$), for example, in (a) above. However we note that it is theoretically possible to eliminate all auxiliary variables from purely *arithmetic* constraints [Collins 1982, Tarski 1951]. We will see later that additional requirements restrain us from always achieving this goal. Eliminating redundant constraints (II) is in general very difficult, often more so than the problem of determining constraint satisfiability. Discovering simpler equivalent expressions (III) is also difficult in general; in this paper, it affects us only in the nonlinear constraints.

A traditional approach to constraint simplification, is to use a notion of *canonical form* equipped with an efficient algorithm for its computation. Informally, such a form represents the information content of the original constraints in a minimal manner w.r.t. the target variables $\tilde{x}$. For example, in PROLOG (equations over trees), constraints can be represented by their mgu, and written in the form $\tilde{x} = t(\tilde{y})$ where $\tilde{y}$ are distinct from $\tilde{x}$ and

[1] Underscore notation is used to emphasize auxiliary variables.

t is a tuple of terms. For linear equations, a well known canonical form is called parametric form where equations are represented in the form $\tilde{x} = t(\tilde{y})$ where $\tilde{y}$ are distinct from $\tilde{x}$ and t is a tuple of linear expressions. If linear inequalities are also considered, there is still a natural notion of canonical form [Lassez and McAloon 1988]; however, it is not clear if there exists an efficient algorithm. Finally, if nonlinear equations (including functions like $sin()$ and $abs()$) are also included, then it is not clear what a desired canonical form is, much less if there is an algorithm[2] at all.

In the context of CLP languages, the constraint simplification problem is compounded by other difficulties. One difficulty concerns backtracking: for efficiency the output module operates directly on the run-time structures representing the constraints, and consequently these operations need to be undone. Another difficulty is that constraints are represented in a form designed for testing satisfiability; this form is often unsuitable for computing output.

After a brief outline of the CLP($\mathcal{R}$) system, we focus on the classical problem of projection in *linear* arithmetic constraints. The core element here is a Fourier-based algorithm for eliminating non-target variables. The original Fourier algorithm [Fourier 1824] has the fundamental problem of generating too many redundant constraints, and the systematic removal of all such constraints is prohibitive. A major advance due to Černikov [Černikov 1963] made the Fourier algorithm plausibly practical by using an efficient, but partial, redundancy removal method. Combining the Černikov method with further redundancy removal is, unfortunately, unsound in general. The main technical result of this paper shows that augmenting the Černikov algorithm with strict redundancy removal is in fact sound.

The rest of the paper discusses the remaining kinds of constraints: equations over trees and nonlinear equations. Functor equations are straightforward. For nonlinear constraints, many possible simplifications are not performed because a nonlinear constraint solver is not employed. However, we do employ a general heuristic which is both efficient and effective. Finally, the various sub-algorithms are put together in a specific order, together with a substitution mechanism, to obtain the complete algorithm.

2 CLP($\mathcal{R}$)

Real constants and real variables are both *arithmetic terms*. If t, t_1 and t_2 are arithmetic terms, then so are $(t_1 + t_2)$, $(t_1 - t_2)$, $(t_1 * t_2)$, (t_1/t_2), $abs(t)$, $max(t_1, t_2)$, $min(t_1, t_2)$, $sin(t)$, $cos(t)$ and $pow(t_1, t_2)$. Uninterpreted constants and functors are like those in PROLOG. Uninterpreted constants and arithmetic terms are *terms*, and so is any variable. If f is an n-ary uninterpreted functor, $n \geq 0$, and $t_1, ..., t_n$ are terms, then $f(t_1, ..., t_n)$ is a term. If t_1 and t_2 are arithmetic terms, then $t_1 = t_2$, $t_1 < t_2$ and $t_1 \leq t_2$ are all arithmetic *constraints*. If at least one of t_1 and t_2 is not an arithmetic term, then only the expression $t_1 = t_2$ is a constraint.

An *atom* is of the form $p(t_1, t_2, \ldots, t_n)$ where p is a predicate symbol distinct from $=$, $<$, and $\leq$, and $t_1, ..., t_n$ are terms. A CLP($\mathcal{R}$) program is defined to be a finite collection of *rules* of the form $A_0 :- \alpha_1, \alpha_2, \ldots, \alpha_k$ where each α_i, $0 \leq i \leq k$, is either a constraint or an atom. A CLP($\mathcal{R}$) *goal* consists of a set of *current constraints*, and a *goal body*. These constraints must be *linear consistent*, that is, it can be partitioned into a functor component, a linear component and a nonlinear component such that the conjunction of the first two components is satisfiable; the goal body is the same as a rule body. In an initial goal, the set of current constraints is empty.

Let the goal G be $\Psi ?- \alpha_1, \ldots, \alpha_k$ where Ψ denotes the current constraints. A *derivation step* from G is defined over two cases: if α_1 is a constraint, then G derives $\Psi \cup \alpha_1 ?- \alpha_2, \ldots, \alpha_k$ providing $\Psi \cup \alpha_1$ is linear consistent; if α_1 is an atom A and if there is a rule $A' :- \beta_1, \ldots, \beta_m$, then G derives $\Psi \cup A = A' ?- \beta_1, \ldots, \beta_m, \alpha_2, \ldots, \alpha_k$ providing $\Psi \cup A = A'$ is linear consistent. When no derivation step is possible, execution "backtracks" to a point where an alternate choice of matching rule is available.

A *derivation sequence* is a possibly infinite sequence of goals such that there is a derivation step to each goal from the preceding goal. A derivation sequence is *conditionally successful* if it is finite and the body in the last goal is empty. If, however, the current constraints in this last goal (sometimes called *answer constraints*) has an empty nonlinear component, we say that the derivation sequence is *successful*. Producing appropriate output from these constraints, given as target variables the variables appearing in the original goal, is the subject of this paper.

In the CLP($\mathcal{R}$) system the user can also call, anywhere in the computation, the predefined predicate $dump([x_1, \ldots, x_N])$; this invokes the output module on the current constraint set with target variables $x_1, \ldots, x_N$. For more details of the CLP($\mathcal{R}$) system and its implementation we refer the reader to [Jaffar *et al.* 1990].

3 Linear Constraints

Let C denote the collection of linear constraints at hand, let $x_1, x_2, \ldots, x_N$ (abbreviated $\tilde{x}$) denote the tar-

[2] The satisfiability problem here is in fact undecidable.

```
for (i = 1; i ≤ N; i = i + 1) {
    if (x_i is a parameter) continue;
    let E denote the equation x_i = r.h.s.(x_i) at hand;
    if (r.h.s.(x_i) contains a variable z of lower priority than x_i) {
        choose the z of lowest priority;
        rewrite the equation E into the form z = t;
        if (z is a target variable) mark the equation E as final;
        substitute t for z in the other linear equations and inequalities;
    } else mark the equation E as final;
}
return all final equations;
```

Figure 1: *Linear equations*

get variables within C, and let $y_1, y_2, \ldots, y_M$ (abbreviated $\tilde{y}$) denote the remaining *auxiliary* variables. The linear solver in CLP($\mathcal{R}$) is partitioned into the equation solver and the inequality solver for efficiency reasons.

In this section, we describe an algorithm which outputs $\exists \tilde{y}\, C(\tilde{x}, \tilde{y})$, where C is linear, in terms of target variables only, treating first the equations then the inequalities. The algorithm may produce an output not containing a particular target variable x which appears in C – for example, when eliminating y from $\exists y\, x = y + 2$ – or may produce an untyped equation – for example, producing $x = z$ from $\exists y\, x = y + 1 \wedge z = y + 1$. For such x, we add to the output the special constraint $real(x)$ restricting x to real number values.

3.1 Linear Equations

Equations are maintained in *parametric form*, that is, in the form $\tilde{u} = t(\tilde{v})$ where $\tilde{u}$, called the *object* variables are distinct from $\tilde{v}$, called the *parameters*. For each object variable z we write $r.h.s.(z)$ to denote the linear expression (of parameters) that z is equated to. Inequalities are always written in terms of parameters alone (in addition to other restrictions which do not concern us here).

The algorithm, essentially a form of Gaussian elimination, is described in Figure 1. It assumes there is a priority π among the variables, $\pi(x_1) > \cdots > \pi(x_N) > \pi(y_1) > \cdots > \pi(y_M)$, expressing the relative importance of each variable in the output[3]. The algorithm ensures that lower priority variables are represented in terms of higher priority variables. (We will see later how π is used in the context of functor and nonlinear equations to minimize the number of variables occurring in the output.)

A crucial point for efficiency is that the main loop in Figure 1 iterates N times, and $N \ll M$ in general, that is, the number of target variables is often far smaller than the total number of variables in the system.

Note that the order of variables in the predefined predicate $dump([x_1, \ldots, x_N])$ determines the priority relation over these variables. Hence the user can influence the output representation of the constraints.

3.2 Linear Inequalities

The constraint solver stores the linear inequalities in a Simplex tableau. (See [Jaffar *et al.* 1990] for details.) Each linear inequality is expressed internally as an equality by introducing a *slack* variable, one whose value is restricted to be either nonnegative or positive. Our first job, therefore, is the elimination of such slack variables. This is achieved by pivoting the inequality tableau to make all the slack variables basic so that each appears in exactly one equation. Hence each row can be viewed as $s = exp$ where $s \geq 0$ or $s > 0$, and this equation can now easily be rewritten into the appropriate inequality $exp \geq 0$ or $exp > 0$.

The remainder of this section deals with the problem of eliminating non-target variables which occur in these inequalities. We use a method based on Fourier's algorithm [Fourier 1824]. It is well-known that the direct application of this algorithm is impractical because it generates many redundant constraints. Attempting to eliminate all redundancy at every step is also impractical [Lassez *et al.* 1989]. Adaptations of Fourier's algorithm due to Černikov [Černikov 1963] substantially improve the performance. We show how to incorporate other redundancy elimination methods with those of Černikov to obtain a more practical algorithm for eliminating variables from linear inequalities.

In some circumstances, especially when constraints

[3]The priority among the y_i's is arbitrary.

when written as a matrix is dense, algorithms not based on Fourier such as [Huynh *et al.* 1990] can be more efficient; however, typical CLP($\mathcal{R}$) programs produce sparse matrices. In general, the size of projection can grow exponentially in the number of variables eliminated, even when all redundancy is eliminated. Fourier-based methods have the advantage over other methods that we can stop eliminating variables at any time, thus computing a partial projection.

3.2.1 Fourier-based Methods

We begin with some necessary definitions. A *labeled constraint* is a linear inequality labeled by a set of constraint names. We say that c is *label-subsumed* by c' if $label(c') \subseteq label(c)$. To simplify the explanation, we will not consider strict inequalities. We assume all constraints are written in the form $\sum_{i=1}^{m} \alpha_i x_i \leq \beta$.

We shall be using some algebraic manipulation of constraints. Let c_j be the constraint $\sum_{i=1}^{m} \alpha_{j,i} x_i \leq \beta_j$, for $j = 1, \ldots, n$. Then $\gamma * c_j$ (or γc_j) denotes the constraint $\sum_{i=1}^{m} \gamma\alpha_{j,i} x_i \leq \gamma\beta_j$ where γ is a real number, and $c_j + c_k$ denotes the constraint $\sum_{i=1}^{m} (\alpha_{j,i} + \alpha_{k,i}) x_i \leq \beta_j + \beta_k$. Similarly, $\sum_{j=1}^{n} c_j$ denotes an iterated sum of constraints. We consider constraints c and c' equal, $c = c'$, if $c \equiv \gamma * c'$ for some $\gamma > 0$.

Let $\mathcal{C}$ be a set of labeled constraints. Given a variable x_i, we divide $\mathcal{C}$ into three subsets: $\mathcal{C}^+_{x_i}$, those constraints in which x_i has a positive coefficient (i.e. c_j such that $\alpha_{j,i} > 0$); $\mathcal{C}^-_{x_i}$, those constraints in which x_i has a negative coefficient; and $\mathcal{C}^0_{x_i}$, those constraints in which the coefficient of x_i is zero. We omit the subscript when the given variable is clear from the context.

Let $c_k \in \mathcal{C}^+$ and $c_l \in \mathcal{C}^-$ and let $d = 1/\alpha_{k,i} * c_k + -1/\alpha_{l,i} * c_l$. Then, by construction, x_i does not occur in the constraint d. If $\mathcal{S}_k$ ($\mathcal{S}_l$) is the label of c_k (c_l) then d has label $\mathcal{S}_k \cup \mathcal{S}_l$. Let $\mathcal{D}$ be the collection of all such d. Then $\mathcal{D} \cup \mathcal{C}^0$ is the result of a *Fourier step eliminating* x_i. We write $fourier_i(\mathcal{C}) = \mathcal{D} \cup \mathcal{C}^0$. When both $\mathcal{C}^+$ and $\mathcal{C}^-$ are non-empty then $\mathcal{D}$ is non-empty, and the step is called an *active* variable elimination. After eliminating x_i the total number of constraints in $\mathcal{C}$ increases (possibly decreasing) by $measure(x_i, \mathcal{C}) = |\mathcal{C}^+| \times |\mathcal{C}^-| - |\mathcal{C}^+| - |\mathcal{C}^-|$.

Let $\mathcal{A}$ be a set of constraints where each constraint is labeled by its own name. Define $\mathcal{F}_0 = \mathcal{A}$ and $\mathcal{F}_{i+1} = fourier_{i+1}(\mathcal{F}_i)$. Then $\{\mathcal{F}_i\}_{i=0,1,\ldots}$ is the sequence of constraint sets obtained by Fourier's method, eliminating, in order, $y_1, y_2, \ldots$. We write $\mathbf{F}_i$ for $\{\mathcal{F}_j\}_{j=0,1,\ldots,i}$. It is straightforward (see [Lassez and Maher 1988], for example) that, if $m < n$, $\mathcal{F}_n \leftrightarrow \exists y_{m+1}, y_{m+2}, \ldots, y_n \; \mathcal{F}_m$. In particular, $\mathcal{F}_n \leftrightarrow \exists y_1, y_2, \ldots, y_n \; \mathcal{A}$. Thus Fourier's algorithm computes projections.

However Fourier's algorithm generates many redundant constraints and has doubly-exponential worst-case behavior. Černikov [Černikov 1963] (and later Kohler [Kohler 1967]; see also [Duffin 1974]) proposed modifications which allow some redundant constraints to be eliminated during a Fourier step, and address this problem. The first method eliminates all constraints generated at the n'th active step which have a label of cardinality $n+2$ or greater, for every n. A second method retains, at each step, a set S of constraints such that every constraint generated at this step is label-subsumed by a constraint in S [4]. The first method eliminates a subset of the constraints eliminated by the second. These methods are *correct* in the following sense: If $\{\mathcal{C}_i\}_{i=0,1,\ldots}$ is the sequence generated by such a method, then $\mathcal{C}_i \leftrightarrow \exists y_1 \ldots y_i \; \mathcal{A}$, for every i.

Although it appears that the Černikov modifications to Fourier's algorithm could be augmented by deleting additional redundant constraints after each step, this is incorrect in general [Huynh *et al.* 1990]. The following example highlights this point by showing that the first Černikov algorithm, augmented by the simplest kind of redundancy removal, removal of *duplicate* constraints, is unsound.

3.2.2 An Example

Let $\mathcal{A}$ denote the following labeled constraints. Labels appear to the left of the constraints. It can be verified that $\mathcal{A}$ contains no redundancy.

$$\begin{array}{lrrrrrrrrl}
\{1\} & & w & + & x & + & y & + & z & \leq 1 \\
\{2\} & & w & - & x & + & y & + & z & \leq 1 \\
\{3\} & & -w & + & x & + & y & + & z & \leq 1 \\
\{4\} & & -w & - & x & + & y & + & z & \leq 1 \\
\{5\} & v & & & & - & y & & & \leq 0 \\
\{6\} & -v & & & & & & & & \leq 0
\end{array}$$

Upon eliminating v (by adding the last two constraints), we obtain in the first (Fourier) step:

$$\begin{array}{lrrrrrrrrl}
\{1\} & & w & + & x & + & y & + & z & \leq 1 \\
\{2\} & & w & - & x & + & y & + & z & \leq 1 \\
\{3\} & & -w & + & x & + & y & + & z & \leq 1 \\
\{4\} & & -w & - & x & + & y & + & z & \leq 1 \\
\{5,6\} & & & & & & -y & & & \leq 0
\end{array}$$

Next we eliminate w obtaining:

$$\begin{array}{lrrrrrrrl}
\{1,3\} & & x & + & y & + & z & \leq 1 \\
\{1,4\} & & & & y & + & z & \leq 1 \\
\{2,3\} & & & & y & + & z & \leq 1 \\
\{2,4\} & & -x & + & y & + & z & \leq 1 \\
\{5,6\} & & & & -y & & & \leq 0
\end{array}$$

[4] In the English translation of [Černikov 1963], this is misstated.

Observe that Černikov's criterion does not allow us to delete any constraints. Since the second and third constraints are duplicates, we could delete one. However, we choose not to in this step. Next x is eliminated to obtain:

$$\begin{array}{ll} \{1,2,3,4\} & y + z \leq 1 \\ \{1,4\} & y + z \leq 1 \\ \{2,3\} & y + z \leq 1 \\ \{5,6\} & -y \leq 0 \end{array}$$

The first three constraints are identical, and now we choose to delete the second and third, obtaining:

$$\begin{array}{ll} \{1,2,3,4\} & y + z \leq 1 \\ \{5,6\} & -y \leq 0 \end{array}$$

In the final Fourier step, we eliminate y to obtain:

$$\{1,2,3,4,5,6\} \qquad z \leq 1$$

Černikov's criterion allows us to delete this constraint, and so we finally obtain an empty set of constraints. This outcome is incorrect since it implies $\exists v, w, x, y\ \mathcal{A}$ is true for all values of z, and it is straightforward to verify that, in fact, $\exists v, w, x, y\ \mathcal{A} \leftrightarrow (z \leq 1)$. Observe that we could have achieved the same incorrect outcome if, after eliminating w, one of the duplicate constraints was deleted.

3.2.3 Combining Fourier-based Methods with Strict Redundancy Elimination

Given a set $\mathcal{C}$ of constraints, $c \in \mathcal{C}$ is *redundant* in $\mathcal{C}$ if $\mathcal{C} \leftrightarrow \mathcal{C} - \{c\}$. A subset $\mathcal{R}$ of $\mathcal{C}$ is redundant if $\mathcal{C} \leftrightarrow \mathcal{C} - \mathcal{R}$. We define $\mathcal{C} \twoheadrightarrow c$ iff, for some constraint c', $\mathcal{C} \rightarrow c'$ and $c' \rightarrow c$ but $c \not\rightarrow c'$. Equivalently, (if we are dealing with only non-strict inequalities) $\mathcal{C} \twoheadrightarrow c$ means $\mathcal{C} \rightarrow c'$ where $c = c' + (0 \leq \epsilon)$ for some constraint c' and some $\epsilon > 0$. (Recall that $c' + (0 \leq \epsilon)$ denotes the sum of the constraint c' and the constraint $(0 \leq \epsilon)$.) If also $c \in \mathcal{C}$ then c is said to be *strictly redundant* in $\mathcal{C}$. Geometrically, a strictly redundant constraint c determines a hyperplane which does not intersect the volume defined by $\mathcal{C}$. We write $\mathcal{C} \twoheadrightarrow \mathcal{C}'$ if $\mathcal{C} \twoheadrightarrow c$ for every $c \in \mathcal{C}'$. A constraint $c \in \mathcal{C}$ is said to be *quasi-syntactic redundant* [Lassez *et al.* 1989] if, for some $c' \in \mathcal{C}$ and some $\epsilon > 0$, $c = c' + (0 \leq \epsilon)$. Clearly quasi-syntactic redundancy is one kind of strict redundancy.

We capture Černikov's modifications of Fourier's algorithm and others in the following definition. Let r be a constraint deletion procedure which, at step i, determines a redundant subset of $\mathcal{F}_i$ as a function of the sequence $\mathbf{F}_i$. Let a *Fourier-based* algorithm be one which generates a sequence of constraint sets $\{\mathcal{C}_i\}_{i=0,1,\ldots}$ where $\mathcal{C}_0 = \mathcal{A}$ and $\mathcal{C}_{i+1} = fourier_{i+1}(\mathcal{C}_i) - r(\mathbf{F}_{i+1})$. It is important to note that, in general, it is not necessary for a Fourier-based method to compute the sequence $\{\mathcal{F}_j\}_{j=0,1,\ldots}$. Indeed, these methods are valuable to the extent that they *do not* compute $\mathbf{F}_i$. All that is required is that $\mathcal{C}_i$ can be viewed as being computed using a function of this sequence.

Let r be the function associated with a Fourier-based method, and let s map every constraint set to a subset obtained by deleting some strictly redundant constraints. The sequence of constraint sets $\{\mathcal{K}_i\}_{i=0,1,\ldots}$ where $\mathcal{K}_0 = \mathcal{A}$ and $\mathcal{K}_{i+1} = s(fourier_{i+1}(\mathcal{K}_i) - r(\mathbf{F}_{i+1}))$ is the result of augmenting the Fourier-based method with the deletion of (some) strictly redundant constraints.

We let $\mathcal{D}_i$ denote the set of constraints deleted by s at step i, that is, $\mathcal{D}_i = (fourier_i(\mathcal{K}_{i-1}) - r(\mathbf{F}_i)) - s(fourier_i(\mathcal{K}_{i-1}) - r(\mathbf{F}_i))$. A *removed constraint* in $\mathcal{C}_i$ is defined to be a constraint in $\mathcal{C}_i$ which uses some constraint in $\mathcal{D}_j, j \leq i$ during generation. That is, for some $j \leq i$, if we view generation of $\mathcal{C}_i$ as starting from $\mathcal{C}_j$ (instead of $\mathcal{C}_0$) then c is generated using constraints from $\mathcal{D}_j$. $\mathcal{R}_i$ denotes the set of all removed constraints in $\mathcal{C}_i$. Clearly $\mathcal{F}_i \supseteq \mathcal{C}_i \supseteq \mathcal{K}_i$, $\mathcal{K}_i = \mathcal{C}_i - \mathcal{R}_i$, and $\mathcal{D}_i \subseteq \mathcal{R}_i$.

Numbers denoted by λ's, μ's, ν's and ϵ's are non-negative throughout. Thus $(0 \leq \epsilon)$ is a tautologous constraint. The notation $var(c)$ denotes the set of variables with non-zero coefficient in constraint c.

The following theorem from the folklore underlies all the work below.

Theorem 1 *Let $\mathcal{C} = \{c_0, c_1, \ldots, c_k\}$ be a consistent set of constraints and let c be a constraint. $\mathcal{C} \rightarrow c$ iff $c = \sum_{i=0}^{k} \lambda_i c_i + (0 \leq \epsilon)$ where the λ's and ϵ are non-negative.* □

The next lemma shows that all strictly redundant constraints can be deleted simultaneously from a consistent set of constraints. Consequently it is meaningful to speak of a strictly redundant subset of $\mathcal{C}$. It also shows that a set of redundant constraints can be deleted simultaneously with a set of strictly redundant constraints. The corresponding results for the class of all redundant constraints do not hold.

Lemma 2 *Let $\mathcal{C}$ be a consistent set of constraints.*

1. *If $\mathcal{D} \subseteq \mathcal{C}$ and each $c \in \mathcal{D}$ is strictly redundant in $\mathcal{C}$ then $\mathcal{C} \leftrightarrow \mathcal{C} - \mathcal{D}$.*
2. *If $\mathcal{S}$ is strictly redundant in $\mathcal{C}$ and $\mathcal{R}$ is redundant in $\mathcal{C}$ then $\mathcal{S} \cup \mathcal{R}$ is redundant in $\mathcal{C}$.* □

We now prove that elimination of strict redundancy does not affect correctness of Fourier-based methods.

```
C = initial set of inequalities (after linear substitutions);
label(c) = {c} for each c ∈ C;
n = 0;
while (there exists an auxiliary variable x in C) {
    choose a variable x with minimal measure(x, C);
    D = C_x^0;
    if (|C_x^+| > 0 and |C_x^-| > 0 ) {
        n = n + 1; /* count active eliminations */
        for (each pair c_k ∈ C_x^+, c_l ∈ C_x^-)
            if (|label(c_k) ∪ label(c_l)| ≥ n + 2) continue; /* first Černikov method */
            d is the constraint obtained from c_k and c_l eliminating x;
            label(d) = label(c_k) ∪ label(c_l);
            if (d is not quasi-syntactic redundant wrt D) {
                    E = quasi-syntactic redundant constraints in D wrt d;
                    D = D ∪ {d} − E;
/ **                                second Černikov method
                    if (d is label-subsumed in D)
                        D = D − {d};
                    else {
                        F = constraints in D label-subsumed by d;
                        D = D − F;
                    }
* * */
            }
        }
        C = D;
    }
    return C;
```

Figure 2: *Linear inequalities*

Theorem 3 *Suppose $\mathcal{A}$ is consistent and $\{\mathcal{C}_i\}$ is correct. Then $\{\mathcal{K}_i\}$ is correct.*

Proof: Note that, since $\mathcal{A}$ is consistent, $\mathcal{F}_n$ is consistent for every n, and consequently so are $\mathcal{C}_n$ and $\mathcal{K}_n$. Suppose $c \in \mathcal{R}_n$ and c depends on constraints in $\mathcal{D}_m$, $m \leq n$; say $c = \sum_i \lambda_i c_i + \sum_j \mu_j d_j$ where $d_j \in \mathcal{D}_m$ for each j and $c_i \in \mathcal{C}_m - \mathcal{D}_m$ for each i, and for some j, $\mu_j > 0$. Now for each j, since $\mathcal{C}_m \twoheadrightarrow \mathcal{D}_m$ (Lemma 2), $d_j = \sum_i \nu_{ji} c_i + (0 \leq \epsilon_j)$ where $\epsilon_j > 0$, and $c_i \in \mathcal{C}_m - \mathcal{D}_m$ for each i. Hence $c = \sum_i (\lambda_i + \sum_j \mu_j \nu_{ji}) c_i + (0 \leq \epsilon')$ where $\epsilon' = \sum_i \mu_j \epsilon_j$ and $\epsilon' > 0$. Let $c' = \sum_i (\lambda_i + \sum_j \mu_j \nu_{ji}) c_i$ so that $c = c' + (0 \leq \epsilon')$.

Now $\mathcal{F}_m \rightarrow c'$ since every $c_i \in \mathcal{F}_m$. Furthermore $\mathcal{F}_n \rightarrow c'$ since $var(c') = var(c) \subset \{y_{n+1}, y_{n+2}, \ldots\}$ (since $c \in \mathcal{R}_n$). Since $\{\mathcal{C}_i\}$ is correct, $\mathcal{C}_n \rightarrow c'$, that is, $\mathcal{C}_n \twoheadrightarrow c$.

By applying this argument for every c depending on $\mathcal{D}_m$ and every $m \leq n$, $\mathcal{C}_n \twoheadrightarrow c$ for every $c \in \mathcal{R}_n$. By Lemma 2, $\mathcal{C}_n \leftrightarrow \mathcal{C}_n - \mathcal{R}_n$. But $\mathcal{C}_n - \mathcal{R}_n = \mathcal{K}_n$. Hence $\mathcal{K}_n \leftrightarrow \mathcal{C}_n$ and, since $\{\mathcal{C}_i\}$ is correct, $\mathcal{K}_n \leftrightarrow \mathcal{F}_n$. □

This result extends to sets of constraints containing both strict and non-strict inequalities. Fourier's algorithm and Černikov's modification extend straightforwardly. The definition of $\twoheadrightarrow$ stands, but it is no longer equivalent to $\mathcal{C} \rightarrow c'$ and $c = c' + (0 \leq \epsilon)$, for some $\epsilon > 0$.

Before discussing our algorithm, we briefly outline the costs of various redundancy elimination procedures. Let $\mathcal{C}$ be a set of m inequalities involving n variables, obtained in a Fourier-based method from m_0 original inequalities. Full redundancy elimination using the simplex algorithm has exponential worst-case complexity, although in the average case it is $O(m^3 n)$. Strict redundancy elimination has essentially the same cost as full redundancy elimination. Quasi-syntactic redundancy elimination on the constraints $\mathcal{C}$ has worst-case complexity $O(m^2 n)$. The cost of eliminating redundancy in $\mathcal{C}$ using the first Černikov method has worst-case complexity $O(m m_0)$, and it has the important advantage that a constraint can be deleted *before* the (relatively expen-

sive) process of explicitly constructing it. Application of the second Černikov method has worst-case complexity $O(m^2 m_0)$.

In [Černikov 1963] it is recommended that the first, and then the second Černikov elimination method be applied at each step. The variation in which the second method is applied only intermittently is suggested in [Kohler 1967]. If we want to incorporate the elimination of strict redundancy, the above complexity analysis suggests that quasi-syntactic redundancy elimination may be most cost-effective. The analysis also suggests that this elimination should be performed between the first and second Černikov methods.

Our tests tended to support this reasoning. Using the first Černikov method followed by quasi-syntactic redundancy elimination produced significant improvement over the first method alone. However further processing in accord with the second Černikov method only marginally reduced the number of constraints eliminated and led to an overall increase in computation time. Full redundancy elimination after each Fourier step, which is incompatible with the Černikov methods, slows computation by an order of magnitude. Full strict redundancy elimination added to the Černikov method is also unprofitable.

The algorithm is shown in Figure 2. It uses a heuristic (from [Duffin 1974]) attempting to minimize the number of new constraints generated. There remains the matter of verifying the correctness of the algorithm. It is easy to see that a step i is active in $\{\mathcal{F}_i\}$ iff it is active in $\{\mathcal{K}_i\}$ iff it is active in $\{\mathcal{C}_i\}$. Thus the first Černikov method is Fourier-based, and the corresponding part of the algorithm implements this method, and so is correct. The second part of the algorithm deletes some remaining quasi-syntactic redundancies and, by the previous theorem, is correct. If the third part, which is commented out in Figure 2, is included in the algorithm then theorem 3 does not apply directly. However it is not difficult to show that this algorithm is equivalent to eliminating some of the constraints eliminable by applying the second Černikov method *en bloc* and then eliminating some strictly redundant (not necessarily quasi-syntactic redundant) constraints. Thus the theorem applies and the algorithm is correct.

4 Constraints over Trees

The constraints at hand are equations involving uninterpreted functors, the functor equations. As in PROLOG systems a straightforward way of printing these equations is to print an equation between each target variable and its value.

Consider equivalence classes of variables obtained as the reflexive, symmetric and transitive closure of the relation: $\{(x, y) : x \text{ is bound to } y\}$; write $rep(x)$ to denote the variable of highest priority equivalent to x. Now define the printable value of a variable as:

$$value(f(t_1, \cdots, t_n)) = f(value(t_1), \cdots, value(t_n));$$

$$value(x) = \begin{cases} value(t), & \text{if } x \text{ is bound to a term } t; \\ rep(x), & \text{if } x \text{ is unbound} \end{cases}$$

The output is a set of equations of the form $x = value(x)$ for each target variable x, excepting those variables x for which $value(x)$ is x itself. We remark that most PROLOG systems do not use equivalence classes as above, and thus for example, the binding structure $x \mapsto \_1, y \mapsto \_1$ is generally not printed as $x = y$.

One well-known drawback of the above output method is that the output can be exponentially larger than the original terms involved. For example, the output of $x_1 = f(x_2, x_2), x_2 = f(x_3, x_3), \ldots x_{n-1} = f(x_n, x_n), x_n = a$, where $x_1, \ldots, x_n$ are target variables, is such that the binding of x_1 is a term of size $O(2^n)$. This exponential blowup can be avoided by other methods [Paterson and Wegman 1978], but in practice it occurs rarely. Hence the binding method is adopted in the CLP($\mathcal{R}$) system.

The output of functor equations in the context of other (arithmetic) constraints raises another issue. Recall that is not always possible to eliminate non-target variables appearing in functor equations (e.g. eliminating z in $x = f(z)$). Consequently, arithmetic constraints which affect these unavoidable non-target variables must also be output. We resolve this issue by augmenting the problem description sent to the linear constraint output module (c.f. Section 3) as follows: the target variables now consist of the original target variables and the unavoidable non-target variables, with the latter having priority intermediate between the original target variables and remaining variables.

These secondary target variables are given lower priority than the original target variables in order to minimize their occurrence in the output. The lower priority ensures that such variables appear on the left hand side of arithmetic equations as much as possible. We can then substitute the right hand side of the equation for the variable and omit the equation, thus eliminating the variable. For example, if x and y are the target variables in $x = f(z), y = z + 2$ the output is $x = f(y - 2)$. We discuss this further in section 6.

5 Nonlinear Constraints

In general all nonlinear constraints need to be printed, regardless of the target variables, because omitting them may result in an output which is satisfiable when the original set of constraints is not. For example, given the constraints $x < 0, y * y = -2$ and target variable x, we cannot simply output $x < 0$. This problem arises since we have no guarantee that the nonlinear constraints are satisfiable. When the only nonlinear constraints are caused by multiplication the auxiliary variables in the nonlinear constraints can, in theory, be eliminated. However this approach is not practical with current algorithms [Collins 1982] and not possible once trigonometric functions are introduced. Thus, as with functor equations, the nonlinear equations contribute additional target variables. These are simply all the variables which remain in the nonlinear constraints, and we give them priority lower than the target variables but higher than the variables added from functor equations.

However, there is one observation which can significantly reduce the number of nonlinear equations printed and the number of additional target variables: Suppose a non-target variable y occurs exactly once in the constraints, say in the constraint c, and $p(\tilde{x})$ implies $\exists y\ c(\tilde{x}, y) \leftrightarrow c'(\tilde{x})$, for some constraint c' and some condition p, then c can be replaced by c', provided the remaining constraints imply that $p(\tilde{x})$ holds. Some specific applications of this observation follow.

If y occurs in the form $y = \mathrm{f}(\tilde{x})$ then this constraint can be eliminated provided f is a total function on the real numbers (this excludes functions such as exponentiation and division[5]). If y occurs as $y = x^z$ then we can delete the constraint, provided we know that $x > 0$ or z is an integer other than 0. Similarly, we can delete $x = z^y$ provided that $x > 0$, $z > 0$ and $z \neq 1$, and delete $x = y^z$ provided $x > 0$ and $z \neq 0$. A constraint $x = |y|$ can be replaced by $x \geq 0$; $x = \sin y$ (and $x = \cos y$) can be replaced by $-1 \leq x \leq 1$; $x = min(y, z)$ can be replaced by $x \leq z$ (and similarly for max). A constraint $x = y * z$ (equivalently $y = x/z$) can be eliminated, provided it is known that $z \neq 0$. In this latter case, which can be expected to occur more often than most of the other delayed constraints, we can use linear programming techniques on the linear constraints to test whether z is constrained to be non-zero. Specifically, we add the constraint $z = 0$ to the linear constraint solver and if the solver finds that the resulting set of constraints is inconsistent then we delete $x = y * z$. We undo the effects of the additional constraint using the same mechanism as used for backtracking during execution of a goal.

[5]Strictly speaking, division is not a function, since $y = x/z$ is defined to be equivalent to $x = y * z$ and so $0/0$ can take any value.

There is a significant complication due to the linear constraints which are generated as a result of simplifying nonlinear constraints. As each such linear constraint is generated, it is passed to the linear constraint solver so that a consistency check can be performed[6]. If the resulting constraint system is not consistent then the simplifications are undone and the system backtracks to the nearest choice-point as it normally does after executing a failure.

6 Summary of the Output Module

We now present the output algorithm in its entirety, a collation of the various sub-algorithms described above corresponding to the different kinds of constraints. Note that the order in which the sub-algorithms are invoked is important; essentially, the processing of functor and nonlinear equations must be done first in order to determine the set of secondary target variables. Then the linear constraints are processed in such a way as to maximize the number of secondary target variables that can be eliminated. Step V below, not previously described, performs this elimination. It suffers the same drawback as processing functor equations – potentially the size of output is exponential in the size of the original equations.

Step I
Process the functor equations, in order to obtain the secondary target variables. These are essentially the non-target variables appearing in the bindings of the primary target variables. Obtain a (possibly empty) collection of functor equations.

Step II
Simplify the nonlinear equations, and expand the set of secondary target variables to include all the variables in the simplified collection. Obtain a collection of nonlinear equations. This step might also produce additional linear equations.

Step III
Process the linear equations (Figure 1) with respect to the primary and secondary target variables, using some priority such that the primary variables are higher priority than the secondary variables and the auxiliary variables are of the lowest priority. Obtain a collection of final linear equations involving only target variables.

Step IV
Process the linear inequalities (Figure 2), and note that these may have been modified as a result of

[6]Thus the output module implements a more powerful constraint solver than that used during run-time.

Step III above, using the primary and secondary target variables. Obtain a collection of linear inequalities involving only target variables.

Step V

For each secondary target variable y appearing in a linear equation of the form $y = t$, substitute t for y everywhere, and remove the equation. For each secondary target variable y appearing in a nonlinear equation of the form $y = t$, where y appears elsewhere but not in t, substitute t for y everywhere, and remove the equation.

Step VI

Output all the remaining constraints.

7 Conclusion

The output module of CLP($\mathcal{R}$) has been described. While a large part of the problem coincides with the classical problem of projection in linear constraints, dealing with functor and nonlinear equations, and working in the context of a CLP runtime structure, significantly increase the problem difficulty.

The core element of our algorithm deals with projecting linear constraints; it extends the Fourier/Černikov algorithm with strict redundancy removal. The rest of the paper deals with functor and nonlinear equations and how they are output together with the linear constraints. What is finally obtained is an output module for CLP($\mathcal{R}$) which has proved to be both practical and effective.

We finally remark that the introduction of meta-level facilities [Heintze *et al.* 1989] in a future version of CLP($\mathcal{R}$) significantly complicates the output problem, since the constraint domain is expanded to include representations/codings of constraints.

References

[Černikov 1963] S.N. Černikov. Contraction of Finite Systems of Linear Inequalities (In Russian). *Doklady Akademiia Nauk SSSR*, Vol. 152, No. 5 (1963), pp. 1075 – 1078. (English translation in *Soviet Mathematics Doklady*, Vol. 4, No. 5 (1963), pp. 1520–1524.)

[Collins 1982] G.E. Collins. Quantifier Elimination for Real Closed Fields: a Guide to the Literature. In *Computer Algebra: Symbolic and Algebraic Computation*, Computing Supplement #4, B. Buchberger, R. Loos and G.E. Collins (Eds), Springer-Verlag, 1982, pp. 79–81.

[Duffin 1974] R.J. Duffin. On Fourier's Analysis of Linear Inequality Systems. *Mathematical Programming Study*, Vol. 1 (1974), pp. 71–95.

[Fourier 1824] J-B.J. Fourier. Reported in: Analyse des travaux de l'Acadamie Royale des Sciences, pendant l'annee 1824, Partie mathematique, *Histoire de l'Academie Royale des Sciences de l'Institut de France*, Vol. 7 (1827), pp. xlvii-lv. (Partial English translation in: D.A. Kohler. Translation of a Report by Fourier on his work on Linear Inequalities. *Opsearch*, Vol. 10 (1973), pp. 38-42.)

[Heintze *et al.* 1989] N.C. Heintze, S. Michaylov, P.J. Stuckey and R. Yap. Meta-programming in CLP($\mathcal{R}$). In *Proc. North American Conf. on Logic Programming*, Cleveland, 1989. pp. 1-19.

[Huynh *et al.* 1990] T. Huynh, C. Lassez and J-L. Lassez. Practical Issues on the Projection of Polyhedral Sets. *Annals of Mathematics and Artificial Intelligence*, to appear. (Also: IBM Research Report RC 15872, IBM T.J. Watson Research Center, 1990.)

[Jaffar *et al.* 1990] J. Jaffar, S. Michaylov, P. Stuckey and R. Yap. The CLP($\mathcal{R}$) Language and System, *ACM Transactions on Programming Languages*, to appear. (Also: IBM Research Report RC 16292, IBM T.J. Watson Research Center, 1990.)

[Jaffar and Lassez 1986] J. Jaffar and J-L. Lassez. Constraint Logic Programming. Technical Report 86/73, Dept. of Computer Science, Monash University (June 1986). (An abstract appears in: *Proc.* 14^{th} *Principles of Programming Languages*, Munich, 1987, pp. 111–119.)

[Kohler 1967] D.A. Kohler. Projections of Polyhedral Sets. Ph.D. Thesis, Technical report ORC-67-29, Operations Research Center, University of California at Berkeley (August 1967).

[Lassez *et al.* 1989] J-L. Lassez, T. Huynh and K. McAloon. Simplification and Elimination of Redundant Linear Arithmetic Constraints. In *Proc. North American Conference on Logic Programming*, Cleveland, 1989. pp. 35–51.

[Lassez and McAloon 1988] J-L. Lassez and K. McAloon. Generalized Canonical Forms for Linear Constraints and Applications. In *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, 1988. pp. 703–710.

[Lassez and Maher 1988] J-L. Lassez and M. Maher. On Fourier's Algorithm for Linear Arithmetic Constraints. *Journal of Automated Reasoning*, to appear.

[Paterson and Wegman 1978] M.S. Paterson and M.N. Wegman. Linear Unification. *Journal of Computer and System Sciences*, Vol. 16, No. 2 (1978), pp. 158–167.

[Tarski 1951] A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Berkeley, USA, 1951.